Algoritmo branch-and-bound para o problema do *k*-plex máximo

Mauro Roberto Costa da Silva
Curso de Engenharia de Software, Universidade Federal do Ceará
Campus de Quixadá, Quixadá, Ceará, CEP 63900-000
maurorcsc@gmail.com

Wladimir Araújo Tavares, Fabio Carlos Sousa Dias
Curso de Ciência da Computação, Universidade Federal do Ceará
Campus de Quixadá, Quixadá, Ceará, CEP 63900-000
wladimir@lia.ufc.br, fabiocsd@lia.ufc.br

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

RESUMO

Apresentamos um algoritmo branch-and-bound, chamado BITPLEX, para o problema do k-plex máximo. Sua estratégia de ramificação é baseada em uma heurística de coloração com operações que exploram o paralelismo de bits. Além disso, o BITPLEX utiliza listas saturadas para acelerar o processo de atualização do conjunto candidato, conjunto formado pelos vértices que podem ampliar o k-plex atual. Finalmente, BITPLEX foi comparado com o algoritmo do estado da arte RDPLEX proposto por [Trukhanov et al.(2013)]. As instâncias da DIMACS foram utilizadas para a realização dos testes. O BITPLEX obteve melhor resultado em grafos de densidade maior que 0.5.

PALAVRAS CHAVE. Problema do k-plex máximo. Paralelismo de Bits. Coloração de Grafos

Área principal: Teoria e Algoritmos em Grafos, Otimização Combinatória

ABSTRACT

We present a branch-and-bound algorithm, called BITPLEX, for the maximum k-plex problem. Its branching strategy is based in coloring heuristic with operations that exploit the bit-level paralelism. In addition, BITPLEX uses saturated lists to accelerate the process of updating the candidate set, set formed by the vertices that can enlarge the current k-plex set. Finally, BITPLEX was compared with state-of-art algorithm, called RDPLEX, proposed by [Trukhanov et al.(2013)]. The instances of DIMACS were used to perfom the tests. The BITPLEX obtained better results in graphs of density greater than 0.5.

KEYWORDS. Maximum k-plex problem. Bit Paralelism. Bit-Parallelism. Graph Coloring

Main Area: Theory and Algorithms in Graphs. Combinatorial Optimization.

1. Introdução

Um grafo completo, ou clique, é um grafo tal que todos os seus vértices são adjacentes entre si. O problema da clique máxima (PCM) consiste em encontrar a clique de cardinalidade máxima em uma grafo. O PCM é um problema NP-difícil clássico e tem importância fundamental em otimização combinatória [Karp(1972)]. Em termos práticos, ele possui um grande número de aplicações em diversas áreas que podem ser encontradas em [Pardalos and Xue(1994), Bomze et al.(1999), Wu and Hao(2015)].

O PCM está bastante relacionado com o problema de detecção de coesão em redes sociais. Uma rede social é um grafo em que seus vértices representam atores e as arestas indicam relacionamento entre os atores. Subgrafos coesos são subconjuntos de atores bastante relacionados que tendem a ter um comportamento homogêneo. Neste contexto surge o estudo do relaxamento de algumas propriedades da clique para o estudo de aspectos não capturados por sua estrutura. Uma dessas relaxações é conhecida como k-plex [Seidman and Foster(1978)]. No problema do k-plex máximo, desejamos encontrar um subconjunto S dos vértices do grafo com cardinalidade máxima tal que $\forall v \in S, |N(v) \cap S| \geq |S| - k$, para um inteiro $k \geq 1$, onde $N(v)$ representa o conjunto de vértices adjacentes de v. É fácil ver que toda clique é um 1-plex.

Em [Seidman and Foster(1978)], o conceito de k-plex é introduzido para capturar aspectos ainda não tratados em redes sociais. Em [Balasundaram et al.(2011)], a formulação de programação inteira para o problema do k-plex máximo e um estudo do seu poliedro derivando desigualdades válidas e facetas é realizado e a NP-completude da versão de decisão do problema k-plex é estabelecida. Além disso, duas versões de algoritmos de branch-and-cut são implementadas: **BC-MIS** e **BC-C2PLEX**. A primeira incorporando cortes de conjunto independente máximo e a segunda incorporando cortes de co-k-plex. Em [McClosky and Hicks(2012)], dois algoritmos combinatórios exatos para o problema são apresentados. O primeiro é baseado no algoritmo branch-and-bound apresentado em [Carraghan and Pardalos(1990)] e o segundo é baseado no algoritmo de bonecas russas apresentado em [Ostergard(2002)]. O segundo supera o primeiro em todas as instâncias utilizadas. Em [Trukhanov et al.(2013)], um arcabouço para resolução de várias relaxações do problema da clique máxima baseado no método das bonecas russas é proposto. Este arcabouço foi especializado para resolver os problema do k-plex máximo e da clique s-deficiente máxima. Neste arcabouço, o método das bonecas russas é estendido, incorporando um procedimento de verificação incremental usado para a definição dos vértices que podem entrar em um k-plex construído. Este algoritmo será referenciado no texto por RDPLEX. Experimentos computacionais mostraram que o algoritmo RDPLEX supera os dois algoritmos de branch-and-cut **BC-MIS** e **BC-C2PLEX** na maioria das instâncias. Além disso, o algoritmo RDPLEX apresenta um melhor desempenho em tempo de execução comparado com os algoritmo de [McClosky and Hicks(2012)]. Em [Gschwind et al.(2015)], imprecisões sobre procedimento de verificação para k-plex, apresentado em [Trukhanov et al.(2013)], são resolvidas.

Neste artigo, apresentaremos um algoritmo de branch-and-bound para o problema do k-plex máximo que combina e/ou aperfeiçoa alguns ingredientes utilizados em algoritmos existentes, como segue:

- Limite superior baseado em coloração de vértices usando Paralelismo de bits [Segundo et al.(2010)].
- Estratégia de ramificação baseada em uma coloração parcial inspirada em [Segundo and Tapia(2014)]
- Uso das chamadas listas de vértices saturados que facilita a verificação se um dado conjunto é uma k-plex. [Trukhanov et al.(2013)]

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.

As seguintes notações serão utilizadas neste artigo:

- $grau_G[v]$ é o grau do vértice v em G e $\Delta(G)$ é o valor do grau máximo de G.
- $N(v) = \{u \in V : (v, u) \in E\}$ é a vizinhança do vértice v em G.
- Se $U \subseteq V$, então $G[U] = (U, E[U])$ denota o subgrafo de G induzido por U, onde $E[U] = \{\{u, v\} \in E : u, v \in U\}$.
- S é um conjunto independente de G se somente se $\forall u, v \in S, \{u, v\} \notin E(G)$.
- Uma partição de um conjunto A é uma coleção de conjuntos $\{A_1, A_2, \ldots, A_n\}$ tal que $\forall i A_i \subseteq A$, $\bigcup_{i=1}^{n} A_i = A$ e $\forall i \neq j, A_i \cap A_j = \emptyset$
- Uma l-coloração de G é uma partição de $V(G)$ em conjuntos independentes $\mathbb{S} = \{S_1, S_2, \ldots, S_l\}$.
- $C \subseteq V$ induz uma clique de G se somente se $\forall v \in C, |N(v) \cap C| \geq |C| - 1$.
- $\omega(G)$ denota o cardinalidade da maior clique de G.
- $C \subseteq V$ induz um co-k-plex se $\Delta(G[C]) \leq k - 1$. É fácil ver que um co-1-plex é um conjunto independente.
- Uma l-coloração em co-k-plex de G é uma partição de $V(G)$ em co-k-plex $\mathbb{S} = \{S_1, S_2, \ldots, S_l\}$
- $S \subseteq V$ induz um k-plex de G se somente se $\forall v \in S, |N(v) \cap S| \geq |S| - k$.
- $\omega_k(G)$ denota o cardinalidade do maior k-plex de G.

Um limite superior para o k-plex máximo pode ser obtido utilizando o seguinte lema:

Proposição 1. *[McClosky and Hicks(2012)] Todo grafo G satisfaz $\omega_k(G) \leq \Delta(G) + k$.*

Suponha que existe um k-plex $C \subseteq G$ tal que $|C| > \Delta(G) + k$. Escolha um vértice $v \in C$. Observe que $grau_{G[C]}(v) \geq |C| - k$ pela definição de k-plex. Portanto, $grau_{G[C]}(v) \geq |C| - k > \Delta(G)$, é uma contradição desde que $C \subseteq G$.

O seguinte corolário é obtido quando G é um conjunto independente.

Corolário 1. *Todo conjunto independente C satisfaz $\omega_k(C) \leq k$*

A partir de uma l-coloração de G, podemos derivar um limite superior para $\omega_k(G)$.

Proposição 2. *Seja l-coloração de G. Então $\omega_k(G) \leq lk$.*

Seja K um k-plex máximo de G e $\mathbb{S}$ uma l-coloração de G. Pelo Corolário 1, $\forall S \in \mathbb{S}, |K \cap S| \leq k$. Logo,

$$w_k(G) = \sum_{\{S \in \mathbb{S} \mid |K \cap S| \neq \emptyset\}} |K \cap S| \leq \sum_{\{S \in \mathbb{S}\}} k = lk \quad (1)$$

Proposição 3. *[McClosky and Hicks(2012)] Dado um grafo G e um inteiro $m \geq 1$. Suponha a_m denota o número de vértices $v \in V$ tal que $grau_G[v] \geq m$. Se $j = max\{a_m : a_m \geq k + m\}$ então $\omega_k(G) \geq k + j$*

Suponha que G contém um k-plex K tal que $|K| \geq k + j + 1$. Pela definição de k-plex,

$$|K \cap N(v)| \geq |K| - k \geq j + 1 \quad \forall v \in K$$

Podemos dizer que K contém pelos menos $k + j + 1$ vértices v tal que $grau_G[v] \geq j + 1$. Fato que contradiz a definição de j.

Lema 1. *[Balasundaram et al.(2011)] Seja k um número par. Se C é um co-k-plex então $\omega_k(C) \leq 2k - 2$.*

Suponha que C é co-k-plex tal que $|C| \geq 2k - 1$. Assuma que existe um k-plex S tal que $|S| = 2k - 1$. Pela definição de k-plex,

$$|S \cap N(v)| \geq |S| - k = 2k - 1 - k = k - 1 \quad \forall v \in S$$

Pela definição de co-k-plex,

$$|S \cap N(v)| \leq k - 1 \quad \forall v \in S$$

As duas condições implicam que $G[S]$ é um grafo regular com todos os vértices com grau $k - 1$. Mas não existe um grafo com um número ímpar de vértices com grau ímpar. Fato que contradiz a existência de S.

Lema 2. *[Balasundaram et al.(2011)] Seja k um número ímpar. Se C é um co-k-plex então $\omega_k(C) \leq 2k - 1$.*

Suponha que C é co-k-plex tal que $|C| \geq 2k$. Assuma que existe um k-plex S tal que $|S| = 2k$. Pela definição de k-plex,

$$|S \cap N(v)| \geq |S| - k = 2k - k = k \quad \forall v \in S$$

Pela definição de co-k-plex,

$$|S \cap N(v)| \leq k - 1 \quad \forall v \in S$$

Não existe um grafo em que o grau dos vértices seja maior e menor ou igual $k - 1$. Fato que contradiz a existência de S.

A seguinte proposição é uma consequência direta dos lemas anteriores.

Proposição 4. *[Balasundaram et al.(2011)] Se C é um co-k-plex então $\omega_k(G) \leq 2k - 2 + (k \mod 2)$*

Proposição 5. *[McClosky and Hicks(2012)] Seja l-coloração em co-k-plex $\mathbb{S} = \{S_1, S_2, \ldots, S_l\}$ de G então $\omega_k(G) \leq \sum_{S \in \mathbb{S}} min\{2k - 2 + (k \mod 2), k + j_i, \Delta(G[S_i]) + k, |S_i|\}$, onde $j_i = max\{m : a_m \geq k + m\}$ para cada S_i.*

Seja K um k-plex máximo de G e $\mathbb{S}$ uma l-coloração em co-k-plex de G. Logo,

$$w_k(G) = \sum_{\{S \in \mathbb{S} \,|\, |K \cap S| \neq \emptyset\}} |K \cap S| \leq \sum_{\{S \in \mathbb{S}\}} min\{2k - 2 + (k \mod 2), k + j_i, \Delta(G[S_i]) + k, |S_i|\}$$

Apesar do limite superior calculado em [McClosky and Hicks(2012)] ser melhor do que o custo calculado pela coloração, o custo computacional do cálculo da coloração pode ser reduzido com a utilização das operações que exploram o paralelismo de bits. Observe que não é uma tarefa trivial adaptar um algoritmo para utilizar as operações com paralelismo de bits mantendo a complexidade inicial do algoritmo.

3. Algoritmo branch-and-bound genérico

[McClosky and Hicks(2012)] apresentam um algoritmo do tipo branch-and-bound para o problema da k-plex máximo. Neste algoritmo, um subproblema do k-plex máximo é definido pela tupla (S, P, S_{max}), onde S é o k-plex que está sendo construído, P é chamado de conjunto candidato, pois é formado pelos vértices que podem entrar no k-plex S em construção e S_{max} é a cardinalidade do maior k-plex encontrado.

Um limite superior trivial para o tamanho do maior k-plex $S' \subseteq S \cup P$ é $|S| + |P|$. Logo, um subproblema (S, P, S_{max}) pode ser podado quando $|S| + |P| \leq S_{max}$, ou seja, o conjunto candidato atual não consegue produzir um k-plex que supere a melhor solução encontrada.

No Algoritmo 1, temos um algoritmo branch-and-bound genérico para o problema do k-plex máximo baseado no algoritmo de [McClosky and Hicks(2012)].

Na linha 4 do Algoritmo 1, em cada subproblema, precisamos particionar o conjunto $P = U \cup R$ tal que $\omega_k(U) \leq S_{max} - |S|$. Somente os vértices que estão em R são utilizados no processo chamado de ramificação em largura. O processo de ramificação em largura consiste em enumerar implicitamente todos os k-plexes maximais contendo um vértice $v \in R$ em $G[P]$ e depois o vértice é removido de P. Observe que após todos os vértices de R serem ramificados (e removidos de P), o conjunto P será igual a U, definindo um subproblema que pode ser podado. Esse particionamento pode ser feito utilizando algum procedimento de limite superior. Considerando o limite superior trivial, podemos escolher $U \subseteq P$ tal que $|U| = S_{max} - |S|$.

Na linha 5 definimos uma ordem em que os vértices do conjunto R serão analisados na linha seguinte. Essa ordem interfere no desempenho do algoritmo. Por exemplo, se for escolhido primeiro os vértice de grau baixo, então o número de k-plexes maximais contendo este vértice será reduzido. Por outro lado, se for escolhido primeiro os vértices de grau mais alto, aumentamos a chance de encontrar primeiro um k-plex de tamanho maior.

Na linha 8, a atualização do conjunto candidato é realizado. A rapidez deste passo depende da maneira que o processo de verificação é realizado. O Algoritmo 2 realiza a verificação com complexidade $O(|P| \times |S|^2)$.

Algoritmo 1: Algoritmo *Branch-and-Bound* para encontrar k-plexes

1 Chamada: $basicPlex(\emptyset, V(G), 0)$
função *BasicPlex (S, P, S_{max})*
2   **se** $|S| > S_{max}$ **então**
3     $S_{max} \leftarrow |S|$
4   Particione $P = U \cup R$ tal que $\omega_k(G[U]) \leq S_{max} - |S|$
5   Seja $\beta = (v_1, \ldots, v_{|R|})$ uma ordem dos vértices de R
6   **para** $v \in$ R *na ordem* β **faça**
7     $P \leftarrow P \setminus \{v\}$
8     $P' \leftarrow \{u \in P : IsPlex(S \cup \{v\}, u)\}$
9   `BasicPlex(`$S \cup \{v\}$, P', S_{max}`)`

Algoritmo 2: Algoritmo para verificar se um conjunto é k-plex

função *IsPlex(S, u)*
  para ***cada*** $v \in S$ **faça**
    se $grau_{G[S]}[u] < |S| - k$ **então**
      retorne false
  retorne true

4. Algoritmo branch-and-bound proposto

Nesta seção, apresentamos o algoritmo branch-and-bound proposto neste trabalho, baseado no Algoritmo 1, que iremos denominar de *BITPLEX*.

No *BITPLEX* , utilizamos para o processo de construção do conjunto de ramificação e a ordem utilizada(linha 4 e 5 do Algoritmo 1), uma estratégia de ramificação baseada em uma coloração parcial inspirada em [Segundo and Tapia(2014)] obtida por uma heurística de coloração de vértices com paralelismo de bits. Já para a atualização do conjunto candidato (linha 8 do Algoritmo 1) utilizamos um processo de verificação baseado em listas de vértices saturados de [Trukhanov et al.(2013)].

Como dito anteriormente, o *BITPLEX* combina e/ou aperfeiçoa alguns ingredientes utilizados em algoritmos existentes. Dentre esses ingredientes, temos a utilização do paralelismo de bits.

O paralelismo de bits é uma forma de paralelismo alcançada quando representamos os dados em vetores de bits de tamanho w, onde w é o tamanho da palavra do computador (por exemplo, 32 ou 64 bits). Operações específicas podem ser executadas para um vetor de bits de tamanho w usando uma única instrução do processador. Recentemente, este tipo de paralelismo tem sido explorado para resolver problemas de otimização combinatória como SAT [Segundo et al.(2008)], Clique Máxima [Segundo et al.(2010)], Clique Máxima Ponderada [Tavares et al.(2015), Tavares et al.(2016)], Coloração de Vértices [Komosko et al.(2015)].

Um passo anterior, à utilização do paralelismo de bits, é a definição da ordenação inicial estática dos bits nos vetores de bits utilizados. No Algoritmo *BITPLEX*, essa ordenação causa um impacto no número de conjuntos independentes produzidos pela heurística de coloração e, consequentemente, no limite superior obtido. Na Subseção 4.1, apresentamos o critério utilizado no algoritmo *BITPLEX* .

A apresentação do Algoritmo *BITPLEX* será realizada da seguinte maneira. Na Seção 4.1, apresentamos um critério de ordenação inicial dos vértices. A ordem dos bits nos vetores dos bits seguirá essa ordem. Na Seção 4.2, apresentamos a estratégia de ramificação usando uma adaptação da heurística de coloração com o paralelismo de bits. Na Seção 4.3, apresentamos o algoritmo de verificação baseado em listas saturadas. Na Seção 4.4, apresentamos o algoritmo *BITPLEX* .

4.1. Ordem inicial estática dos vértices

A ordenação inicial estática dos vértices desempenha um papel na heurística de coloração. Ela é usada como critério de seleção do próximo vértice para a construção de uma nova cor. Um esquema de ordenação dos vértices que minimiza o número de conjuntos independentes (cores) foi derivado em [Matula and Beck(1983)]. Ele pode ser descrito pelo seguinte procedimento:

1. Para $n = |V(G)|$, seja v_n escolhido para ter o menor grau em G.
2. Para $i = n-1, n-2, \ldots, 1$, seja v_i escolhido para ter o menor grau em $G_i = G[V \setminus \{v_{i+1}, v_{i+2}, \ldots, v_n\}]$

4.2. Construção do conjunto de ramificação

A construção do conjunto de ramificação é realizada pelo Algoritmo 3. Este algoritmo é baseado na heurística de coloração de vértices de [Tomita and Seki(2003)] e na sua versão com paralelismo de bits [Segundo et al.(2010)]. A função *Branching* recebe dois parâmetros: um vetor de bits P e um inteiro *limite*, onde *limite* representa o limite superior para k-plex máximo em U, ou seja, $\omega_k(G[U]) \leq limite$ tal que $P = U \cup R$. Esta função constrói implicitamente a partição do conjunto candidato $P = U \cup R$. Começamos com a seguinte partição $P = \emptyset \cup R$. A cada passo, um conjunto independente S_j é gerado. Esse conjunto independente é removido totalmente ou parcialmente de R e colocado implicitamente em U dependendo do limite ainda disponível. No final do processo, o conjunto de ramificação R é construído. As operações destacadas são executadas utilizando o paralelismo de bits.

Algoritmo 3: Algoritmo *Branching*

função *Branching (P, limite)*
  $j \leftarrow 0$
  $R \leftarrow P$
  enquanto $R \neq \emptyset$ **faça**
    se $limite \leq 0$ **então**
      retorne R
    $S_j \leftarrow \emptyset$
    $\boxed{C \leftarrow R}$
    enquanto $C \neq \emptyset$ **faça**
      $\boxed{v \leftarrow first(C)}$
      $S_j \leftarrow S_j \cup \{v\}$
      $\boxed{C \leftarrow C \setminus \{v, N(v)\}}$
    $count \leftarrow \min(|S_j|, k, limite)$
    $limite \leftarrow limite - count$
    $R \leftarrow R -$os primeiros *count* elementos de S_j
    $j \leftarrow j + 1$
  retorne R

4.3. Geração de conjuntos candidatos

Quando um novo vértice é adicionado à solução atual, é necessário gerar um novo conjunto candidato. No Algoritmo 1, essa geração do conjunto candidato é feito pelo Algoritmo 2 que possui um custo elevado. Essa operação pode ser executada de maneira mais eficiente usando as chamadas listas de vértices saturados apresentada em [Trukhanov et al.(2013)].

Dada um k-plex $S \subseteq V$, um vértice $v \in S$ é chamado de saturado quando possui $k - 1$ não-vizinhos em S. Uma lista de vértices saturados $SL \subseteq S$ é um conjunto formado pelos vértices saturados da solução S. Logo, se adicionarmos um vértice u que não é vizinho de algum vértice saturado $v \in SL$, $S \cup \{u\}$ não é k-plex.

Em [Trukhanov et al.(2013)], a criação de listas saturadas é realizada pelo procedimento *MakeSaturatedList* descrito pelo Algoritmo 4. Essa função recebe um conjunto de vértices P; um vetor *nncnt*, no qual $nncnt[v]$ é o número de não-vizinhos do vértice v; a solução atual S; e um inteiro k indicando o tipo do k-plex e devolve a lista de vértices saturados SL na solução S.

Na função *MakeSaturatedList*, o número de não-vizinhos na solução atual é atualizado com relação ao último vértice adicionado à S para os vértices que estão na solução $S - \{u\}$ como para os vértices que estão no conjunto candidato P. As Linhas 6-8 de *MakeSaturatedList* não existem no algoritmo apresentado em [Trukhanov et al.(2013)], e isso faz com que vértices que estão no conjunto candidato já saturados possam entrar na solução. A correção para esse problema foi apontada em [Gschwind et al.(2015)]. Por fim, o algoritmo constrói a lista dos vértices saturados.

Sendo SL a lista dos vértices saturados em uma solução S, é possível concluir que os únicos vértices pertencentes ao conjunto candidato P que podem entrar na solução são os que possuem adjacência com todos os vértices saturados de SL e sejam não adjacentes a no máximo $k - 1$ vértices em S. A partir disso, pode-se verificar quais vértices de P

fazem parte do novo conjunto candidato. A função para essa verificação está descrita no Algoritmo 5.

Algoritmo 4: Algoritmo para gerar lista de vértices saturados

função *MakeSaturatedList(P, nncnt, S, k)*
- $SL \leftarrow \emptyset$
- $u \leftarrow$ último vértice de S
- **para *cada*** $v \in S - \{u\}$ **faça**
  - **se** $(u, v) \notin E(G)$ **então**
    - $nncnt[v] \leftarrow nncnt[v] + 1$
- **para *cada*** $v \in P$ **faça**
  - **se** $(u, v) \notin E(G)$ **então**
    - $nncnt[v] \leftarrow nncnt[v] + 1$
- **para *cada*** $v \in S$ **faça**
  - **se** $nncnt[v] = k - 1$ **então**
    - $SL \leftarrow SL \cup \{v\}$
- **retorne** SL

Algoritmo 5: Algoritmo para verificar se um conjunto é k-plex

função *IsPlex(SL, v, nncnt)*
- **se** *nncnt[v]* $>$ *k-1* **então**
  - **retorne false**
- **para *cada*** $u \in SL$ **faça**
  - **se** $(u, v) \notin E(G)$ **então**
    - **retorne false**
- **retorne true**

A função *IsPlex* é um procedimento utilizado para verificar se um vértice de um conjunto candidato P continua no conjunto após a atualização da solução atual. Caso um vértice v de P seja adjacente a todos os vértices de uma lista saturada SL e o número máximo de não vizinhos em S seja $k - 1$, então $S \cup \{v\}$ é k-plex. Caso contrário, $S \cup \{v\}$ não é k-plex. A complexidade de *IsPlex* é da ordem de $O(|SL|)$.

O Algoritmo 6 descreve como realizamos a construção do novo conjunto candidato. A função *Generation* recebe um conjunto candidato P e, para cada vértice $v \in P$, verifica se *IsPlex(S, v, nncnt)* é verdadeiro. *Generation* retorna um conjunto R com todos os vértices pertencentes a P que satisfizeram a chamada à função *IsPlex* as duas condições verificadas por *IsPlex* e o vetor *nncnt* com valores atualizados. A complexidade de *Generation* é da ordem de $O(|U| \times |SL| + |S| + |U|)$.

Algoritmo 6: Função para gerar conjuntos candidatos

função *Generation (P, k, S, nncnt)*
- $R \leftarrow \emptyset$
- $SL \leftarrow$ `MakeSaturatedList`$(P,\ nncnt,\ S,\ k)$
- **para *cada*** $v \in P$ **faça**
  - **se** `IsPlex`$(SL,\ v,\ nncnt)$ **então**
    - $R \leftarrow R \cup \{v\}$
- **retorne** $(R,\ nncnt)$

4.4. Algoritmo *BITPLEX*

No Algoritmo 7, apresentamos o *BITPLEX*. Ele utiliza o *Branching* para gerar o conjunto de ramificação, que usa a heurística de coloração com paralelismo de bits, e o *Generation*, que usa as listas saturadas para atualização do conjunto de candidatos.

Algoritmo 7: Algoritmo BITPLEX

$S \leftarrow \emptyset$
$P \leftarrow V(G)$
$S_{max} \leftarrow 0$
$nncnt \leftarrow \{0, 0, ..., 0\}$
função *BITPLEX* $(S,\ P,\ S_{max},\ nncnt)$
  se $|S| > S_{max}$ **então**
    $S_{max} \leftarrow |S|$
  $limite \leftarrow S_{max} - |S|$
  $\boxed{R \leftarrow \texttt{Branching}(P,\ limite)}$
  enquanto $R \neq \emptyset$ **faça**
    $v \leftarrow$ primeiro vértice de R
    $\boxed{P \leftarrow P - \{v\}}$
    $\boxed{R \leftarrow R - \{v\}}$
    $(P', newNncnt) \leftarrow \texttt{Generation}(P,\ k,\ S \cup \{v\},\ nncnt)$
    $\texttt{BITPLEX}(S \cup \{v\},\ P',\ S_{max},\ newNncnt)$

5. Resultados Computacionais

Nós comparamos a perfomance computacional do *BITPLEX* com o algoritmo *RDPLEX*. O algoritmo *RDPLEX* foi implementado conforme apresentado em [Trukhanov et al.(2013)] e realizando as correções indicadas em [Gschwind et al.(2015)].

Nós avaliamos o desempenho dos algoritmos utilizando o conjunto de instâncias da DIMACS. Este conjunto de instâncias foi criado para o segundo desafio de Cliques, Satisfabilidade e Coloração de grafos.

Todos os testes foram realizados em um computador com 8GB de memória RAM, processador Intel(R) Xeon(R) CPU E31240 @ 3.30GHz, Sistema Operacional Linux e com tempo limite de 3600 segundos.

Agrupamos os grafos pela densidade ($d = \frac{2|E|}{|V||V-1|}$). Foi considerado como densidade alta valor acima de 0.5. A Tabela 1 e a Tabela 2 apresentam os resultados para os grafos com densidade alta e baixa, respectivamente. A primeira coluna indica o nome da instância, a segunda coluna (n,d) representa o número de vértices a densidade do grafo e para cada algoritmo, temos duas colunas com a melhor solução encontrada e o tempo computacional em segundos. O * indica a solução ótima.

Para cada instâncias foi destacado o resultado do algoritmo que obteve o melhor resultado. Nos casos em que o tempo limite foi alcançado por ambos algoritmos, aquele com a melhor solução viável foi considerado melhor, e quando ambos encontram a solução ótima, o algoritmo com o menor tempo foi considerado o melhor. Em todas as instâncias consideramos $k = 2$.

Grafo	(n,d)	BITPLEX		RDPlex	
		$\omega_2(G)$	segundos	$\omega_2(G)$	segundos
brock200_1	(200,0.75)	26	3600	26	3600
brock200_3	(200,0.61)	17*	211.35	**17***	**110.80**
brock200_4	(200,0.66)	20*	1115.30	**20***	**567.13**
brock400_1	(400,0.75)	**29**	**3600**	28	3600
brock400_2	(400,0.75)	28	3600	28	3600
brock400_3	(400,0.75)	**29**	**3600**	28	3600
brock400_4	(400,0.75)	**29**	**3600**	28	3600
brock800_1	(800,0.65)	**24**	**3600**	23	3600
brock800_2	(800,0.65)	**23**	**3600**	22	3600
brock800_3	(800,0.65)	**24**	**3600**	23	3600

Continua na próxima página...

Grafo	BITPLEX		RDPlex		
		$\omega_2(G)$	segundos	$\omega_2(G)$	segundos
brock800_4	(800,0.65)	**23**	**3600**	22	3600
C125.9	(125,0.90)	43	3600	43	3600
C250.9	(250,0.90)	**51**	**3600**	48	3600
C500.9	(500,0.90)	60	3600	60	3600
C1000.9	(1000,0.90)	**71**	**3600**	68	3600
C2000.9	(2000,0.90)	**79**	**3600**	72	3600
gen200_p0.9_44	(200,0.90)	**49**	**3600**	47	3600
gen200_p0.9_55	(200,0.90)	52	3600	**53**	**3600**
gen400_p0.9_55	(400,0.90)	**59**	**3600**	56	3600
gen400_p0.9_65	(400,0.90)	64	3600	64	3600
gen400_p0.9_75	(400,0.90)	74	3600	74	3600
hamming6-2	(64,0.90)	32*	31.30	**32***	**2.57**
hamming8-2	(256,0.97)	128	3600	128	3600
hamming8-4	(256,0.64)	16	3600	16	3600
hamming10-2	(1024,0.99)	512	3600	512	3600
hamming10-4	(1024,0.83)	**44**	**3600**	32	3600
johnson8-2-4	(28,0.56)	5*	0.00	5*	0.00
johnson8-4-4	(70,0.77)	14*	3.56	**14***	**2.28**
johnson16-2-4	(120,0.76)	10	3600	**10***	**2698.35**
johnson32-2-4	(496,0.88)	20	3600	20	3600
keller4	(171,0.65)	15*	448.00	**15***	**447.23**
keller5	(776,0.75)	**31**	**3600**	30	3600
MANN_a9	(45,0.93)	**26***	**0.21**	26*	0.70
MANN_a27	(378, 0.99)	**235**	**3600**	234	3600
MANN_a45	(1035, 0.99)	**661**	**3600**	660	3600
p_hat300-3	(300,0.74)	43	3600	43	3600
p_hat500-3	(500,0.75)	**60**	**3600**	59	3600
p_hat700-3	(700,0.75)	71	3600	71	3600
p_hat1000-3	(1000,0.74)	**78**	**3600**	77	3600
p_hat1500-3	(1000,0.75)	**111**	**3600**	109	3600
san200_0.7_1	(200,0.70)	30	3600	30	3600
san200_0.7_2	(200,0.70)	24	3600	24	3600
san200_0.9_1	(200,0.90)	90	3600	90	3600
san200_0.9_2	(200,0.90)	70	3600	70	3600
san200_0.9_3	(200,0.90)	48	3600	48	3600
san400_0.7_1	(400,0.70)	40	3600	40	3600
san400_0.7_2	(400,0.70)	30	3600	30	3600
san400_0.7_3	(400,0.70)	24	3600	24	3600
san400_0.9_1	(400,0.90)	100	3600	100	3600
sanr200_0.7	(200,0.70)	22	3600	**22***	**2957.70**
sanr200_0.9	(200,0.90)	**50**	**3600**	49	3600
sanr400_0.7	(400,0.70)	**26**	**3600**	25	3600

Tabela 1: Resultados para instâncias com densidade alta.

Nas instâncias de densidade alta, o *BITPLEX* foi melhor em 22 instâncias, enquanto *RDPlex* em apenas 8. Nas demais 22 instancias, os resultados de ambos foram equivalentes. Destacamos que o *RDPLEX* encontra a solução ótima em duas instâncias (johnson16-2-4 e sanr200_0.7) e o *BITPLEX* as encontra sem provar a otimalidade por alcançar o limite de tempo.

Agora, para as instâncias de densidade baixa, o *RDPlex* foi melhor em 13 instâncias, o *BITPLEX* foi melhor em 4 instâncias e em 7 foram equivalentes.

Pelas Tabelas 1 e 2, percebe-se que o *BITPLEX* se mostra melhor em instâncias com densidade alta e o *RDPLEX* em instâncias de densidade baixa, além disso, observamos que as instâncias que os algoritmos alcançaram o limite de tempo, o *BITPLEX* sempre encontra uma solução viável de melhor qualidade, exceto na instância gen200_p0.9_55.

Grafo	(n,d)	BITPLEX		RDPlex	
		$\omega_2(G)$	segundos	$\omega_2(G)$	segundos
brock200_2	(200,0.50)	13*	10.72	**13***	**7.13**
C2000.5	(2000,0.50)	**18**	**3600**	17	3600
C4000.5	(4000,0.50)	**19**	**3600**	17	3600
c-fat200-1	(200,0.08)	12*	0.02	**12***	**0.01**
c-fat200-2	(200,0.16)	4*	0.11	**24***	**0.01**
c-fat200-5	(200,0.43)	58	3600	58	3600

Continua na próxima página...

Grafo	(n,d)	BITPLEX		RDPlex	
		$\omega_2(G)$	segundos	$\omega_2(G)$	segundos
c-fat500-1	(500,0,04)	14*	0.26	**14***	**0.07**
c-fat500-2	(500,0.07)	26*	2.13	**26***	**0.08**
c-fat500-5	(500,0.19)	64	3600	**64***	**54.28**
c-fat500-10	(500,0.37)	126	3600	126	3600
hamming6-4	(64,0.35)	6*	0.01	**6***	**0.00**
p_hat300-1	(300,0.24)	10*	1.45	**10***	**0.49**
p_hat300-2	(300,0.49)	30	3600	**30***	**412.73**
p_hat500-1	(500,0.25)	12*	21.49	**12***	**8.95**
p_hat500-2	(500,0.50)	42	3600	42	3600
p_hat700-1	(700,0.25)	13*	148.06	**13***	**51.60**
p_hat700-2	(700,0.50)	51	3600	51	3600
p_hat1000-1	(1000,0.24)	13*	1133.85	**13***	**503.34**
p_hat1000-2	(1000,0.49)	**55**	**3600**	54	3600
p_hat1500-1	(1500,0.25)	14	3600	14	3600
p_hat1500-2	(1500,0.50)	**74**	**3600**	71	3600
san400_0.5_1	(400,0.50)	14	3600	14	3600
san1000	(1000,0.50)	16	3600	16	3600
sanr400_0.5	(400,0.50)	15*	2210.96	**15***	**1380.06**

Tabela 2: Resultados para instâncias com densidade baixa.

6. Conclusões e Trabalhos Futuros

Este trabalho apresenta um algoritmo do tipo branch-and-bound para o problema do k-plex máximo, que comparado ao estado da arte do problema, alcançou resultados melhores para grafos com densidade acima de 0.5. Nesses, o nosso algoritmo foi melhor em 22 de 52 instâncias testadas, contra 8 do algoritmo de [Trukhanov et al.(2013)]. Observamos também que para os grafos de densidade igual ou abaixo de 0.5, o algoritmo de [Trukhanov et al.(2013)] foi melhor em 13 de 24, contra 4.

Como trabalho futuro, uma pesquisa focada no limite superior poderia resultar em um algoritmo mais eficiente do que o *Branching*. Por questões de tempo, essa pesquisa não pôde ser realizada durante este trabalho, assim como algumas melhorias e passos. Estes serão apresentados a seguir como trabalhos futuros:

- Implementar a heurística de [McClosky and Hicks(2012)] com operações com paralelismo de bits mantendo a complexidade de tempo inicial;
- Desenvolver uma solução híbrida utilizando o método da boneca russa (RDS), paralelismo de bits e a coloração *Branching*;
- Desenvolver heurísticas para obtenção de uma solução inicial de qualidade.

Além desses trabalhos futuros, iremos testar o BITPLEX para instâncias com $k > 2$. Acreditamos que assim como o RDPLEX mostrou-se eficiente, como relatado em [Trukhanov et al.(2013)], o BITPLEX terá desempenho semelhante ao encontrado nesse trabalho.

Referências

[Balasundaram et al.(2011)] **Balasundaram, B., Butenko, S., and Hicks, I. V.** (2011). Clique relaxations in social network analysis: The maximum k-plex problem. *Oper. Res.*, 59(1):133–142.

[Bomze et al.(1999)] **Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M.** (1999). The maximum clique problem. *Handbook of Comb. Optim.*, 4:1–74.

[Carraghan and Pardalos(1990)] **Carraghan, R. and Pardalos, P. M.** (1990). An exact algorithm for the maximum clique problem. *Oper. Res. Lett.*, 9(6):375 – 382.

[Gschwind et al.(2015)] **Gschwind, T., Irnich, S., Podlinski, I., et al.** (2015). Maximum weight relaxed cliques and russian doll search revisited. Technical report.

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

[Komosko et al.(2015)] **Komosko, L., Batsyn, M., San Segundo, P., and Pardalos, P. M.** (2015). A fast greedy sequential heuristic for the vertex colouring problem based on bitwise operations. *J. of Comb. Opt.*, pages 1–13.

[Matula and Beck(1983)] **Matula, D. W. and Beck, L. L.** (1983). Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427.

[McClosky and Hicks(2012)] **McClosky, B. and Hicks, I. V.** (2012). Combinatorial algorithms for the maximum k-plex problem. *J. of Comb. Opt.*, 23(1):29–49.

[Ostergard(2002)] **Ostergard, P. R.** (2002). A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120:197–207.

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

[Segundo et al.(2010)] **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.

[Segundo and Tapia(2014)] **Segundo, P. S. and Tapia, C.** (2014). Relaxed approximate coloring in exact maximum clique search. *Computers & OR*, 44:185–192.

[Segundo et al.(2008)] **Segundo, P. S., Tapia, C., Puente, J., and Rodríguez-Losada, D.** (2008). A new exact bit-parallel algorithm for sat. In *ICTAI'08. 20th IEEE International Conference on*, volume 2, pages 59–65. IEEE.

[Seidman and Foster(1978)] **Seidman, S. B. and Foster, B. L.** (1978). A graph-theoretic generalization of the clique concept. *J. of Math. Sociology*, 6:139–154.

[Tavares et al.(2015)] **Tavares, W. A., Neto, M. B. C., Rodrigues, C. D., and Michelon, P.** (2015). Um algoritmo de branch and bound para o problema da clique maxima ponderada. *Proceedings of XLVII SBPO*, 1.

[Tavares et al.(2016)] **Tavares, W. A., Neto, M. B. C., Rodrigues, C. D., and Michelon, P.** (2016). Bitclique: um algoritmo de branch-and-bound para o problema da clique máxima ponderada. *Proceedings of XLVIII SBPO*, 1.

[Tomita and Seki(2003)] **Tomita, E. and Seki, T.** (2003). An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete math. and theoretical computer science*, pages 278–289. Springer, Dijon, France.

[Trukhanov et al.(2013)] **Trukhanov, S., Balasubramaniam, C., Balasundaram, B., and Butenko, S.** (2013). Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Opt. and Applications*, 56(1):113–130.

[Wu and Hao(2015)] **Wu, Q. and Hao, J.** (2015). A review on algorithms for maximum clique problems. *Eur. J. of Oper. Res.*, 242(3):693–709.