LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

Algoritmos para o problema da clique máxima com paralelismo de bits

Igor Alan Albuquerque de Sousa, Wladimir Araújo Tavares
Curso de Ciência da Computação, Universidade Federal do Ceará
Campus de Quixadá, Quixadá, Ceará, CEP 63900-000
igoralbuquerque102@hotmail.com, wladimirtavares@ufc.br

RESUMO

Este trabalho aborda o Problema da Clique Máxima (PCM) em grafos, um problema NP-Completo bem conhecido na teoria dos grafos e na otimização combinatória. A maioria dos algoritmos de branch-and-bound estado da arte utiliza a heurística de coloração gulosa tanto para determinar a ordem de ramificação quanto para obter limites superiores. Em muitos casos, as diferenças nos ambientes computacionais frequentemente dificultam comparações diretas de desempenho entre algoritmos. Neste estudo, realizamos uma comparação computacional do algoritmo MCS, apresentado no framework unificado para PCM de Carmo e Züge [2012], com dois algoritmos que utilizam paralelismo de bits por meio da estrutura Bitmap Roaring. Essa estrutura mostrou-se eficiente em aproveitar o paralelismo de bits proposto para os algoritmos mais recentes.

PALAVRAS CHAVE.

Área principal: Grafos, Clique máxima, Coloração, Roaring bitmaps, Ramificação e poda

ABSTRACT

This work addresses the Maximum Clique Problem (MCP) in graphs, a well-known NP-Complete problem in graph theory and combinatorial optimization. Most state-of-the-art branch-and-bound algorithms use greedy coloring heuristics to determine both branching order and upper bounds. In many cases, differences in computational environments can make comparisons less precise. In this study, we perform a computational comparison of the MCS algorithm, presented in the unified framework for MCP by Carmo e Züge [2012], with two bit-parallel algorithms using the Roaring Bitmap structure. This structure has proven efficient in leveraging bit parallelism as proposed for more recent algorithms.

KEYWORDS.

Main Area: Graphs; Maximum Clique; Coloring; Roaring bitmaps; Branch and Bound.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

1. Introdução

Dado um grafo $G = (V, E)$, uma k-clique C em G é um subconjunto de V tal que $|C| = k$ e, para todo par de vértices distintos $u, v \in C$, tem-se que $u, v \in E$. Uma clique máxima de G é a maior clique em G. O número de clique, denotado por $\omega(G)$, é o maior valor de k para o qual G possui uma k-clique. O problema da clique máxima (PCM) consiste em encontrar $\omega(G)$ em um grafo G. A versão de decisão desse problema é NP-Completo, o que implica que não há algoritmo em tempo polinomial, a menos que P = NP (Bomze et al. [1999]).

O PCM é amplamente estudado em diversas áreas, como química (Hasan et al. [2023]), bioinformática (Das et al. [2023]; Okamoto [2020]), teoria dos códigos (Battail [2019]), economia (Boginski et al. [2006]), análise de redes sociais (Balasundaram et al. [2011]), robótica (San Segundo e Rodriguez-Losada [2013]), entre outras. Além disso, o PCM aparece como subproblema de outros problemas combinatórios importantes, como coloração de vértices (Wu e Hao [2012]), partições em cliques (Wang et al. [2006]) e determinação de vencedores em leilões combinatórios (Wu e Hao [2015]).

Nos últimos anos, houve um grande esforço no desenvolvimento de algoritmos exatos eficientes na prática. Muitos dos algoritmos exatos desenvolvidos utilizam a técnica de branch-and-bound (B&B), onde heurísticas de coloração de vértices são aplicadas tanto para obter limites superiores quanto para guiar a ramificação. Carmo e Züge [2012] realizaram uma comparação computacional implementando oito diferentes algoritmos de B&B em um arcabouço conceitual comum, com o objetivo de realizar uma análise mais aprofundada dos algoritmos. No entanto, os autores não consideraram algoritmos mais recentes que incorporam técnicas como paralelismo de bits (San Segundo et al. [2011]), recoloração de vértices (San Segundo et al. [2013]) e filtragem baseada em propagação de restrições (San Segundo et al. [2023]).

A principal contribuição deste trabalho é a integração de técnicas recentes de paralelismo de bits, utilizando a biblioteca RoaringBitmap Chambi et al. [2016], em um algoritmo de B&B e em um algoritmo de Bonecas Russas para o PCM. Os testes computacionais demonstram que essa estrutura permite expandir o arcabouço conceitual unificado proposto por Carmo e Züge [2012] para incluir os algoritmos mais modernos. Na subseção 2.1, apresentamos testes computacionais que mostram que outras estruturas de dados disponíveis não se mostraram adequadas para essa extensão.

Este artigo está organizado da seguinte forma. Na Seção 2, apresentamos os conceitos preliminares e a estrutura de dados RoaringBitmap. Na Seção 3, descrevemos a heurística de coloração gulosa com paralelismo de bits e a sua versão relaxada. Na Seção 4, apresentamos o algoritmo de B&B com coloração relaxada. Na Seção 5, discutimos uma versão simplificada do algoritmo de Bonecas Russas proposto por Corrêa et al. [2014]. Na Seção 6, mostramos os resultados computacionais de nossos experimentos. Por fim, na Seção 7, apresentamos as conclusões do trabalho.

2. Preliminares

Nesta seção, apresentamos os conceitos fundamentais relacionados ao problema da clique máxima (PCM) e realizamos uma comparação entre o bitmap Roaring e outras implementações.

Um **grafo** G é um 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 qualquer grafo G, denotamos por $V(G)$ e $E(G)$, respectivamente, os conjuntos de vértices e arestas de G.

Utilizaremos as seguintes notações ao longo deste artigo:

- $N(v) = \{u \in V : \{v, u\} \in E\}$ é a vizinhança de um vértice v em G, formada pelos vértices adjacentes a v.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

- Se $U \subseteq V$, então $G[U] = (U, E[U])$ denota o subgrafo induzido por U, onde $E[U] = u, v \in E : u, v \in U$.
- S é um conjunto independente em G se, e somente se, $\forall u, v \in S, u, v \notin E(G)$.
- Uma k-coloração dos vértices de G é uma partição de V em k conjuntos independentes: $V = V_1, V_2, \ldots, V_k$.

Observe que o número de conjuntos independentes em uma coloração de G é um limite superior para $\omega(G)$, o tamanho da clique máxima.

Proposição 1. *Seja $V_1, \ldots, V_k$ uma k-coloração de G. Então, $k \geq \omega(G)$.*

Prova Seja K uma clique máxima de G. Como todos os vértices de K são adjacentes entre si, cada vértice de K deve estar em um conjunto independente diferente. Logo, $k \geq \omega(G)$.

2.1. Bitmap Roaring

Um bitmap é uma representação compacta de um conjunto de inteiros, onde cada elemento é representado por um bit. Se um número faz parte do conjunto, o bit correspondente é 1; caso contrário, é 0. Essas estruturas permitem realizar operações como interseção e união de forma eficiente usando **operações bit a bit**, ou seja, operações que manipulam os bits de um tipo de dado.

Entretanto, o uso de memória por bitmaps pode ser elevado, especialmente em conjuntos dispersos. O formato **Roaring Bitmap**, proposto por Lemire et al. [2018], resolve esse problema ao dividir os dados em contêineres compactados, organizados de acordo com os 16 bits mais significativos dos números. Essa abordagem reduz o uso de memória sem comprometer o desempenho das operações.

A Figura 1 exemplifica um Roaring Bitmap com três contêineres, que agrupam números com os mesmos 16 bits mais significativos:

- **Contêiner 1 (0x0000)**: Contém os primeiros 1000 múltiplos de 62.
- **Contêiner 2 (0x0001)**: contendo todos os inteiros nos intervalos $[2^{16}, 2^{16} + 100)$, $[2^{16} + 101, 2^{16} + 201)$ e $[2^{16} + 300, 2^{16} + 400)$
- **Contêiner 3 (0x0002)**: Inclui os números pares no intervalo $[2 \times 2^{16}, 3 \times 2^{16})$.

Essa estrutura, ao agrupar números com base em padrões nos bits mais significativos, otimiza a compressão e as operações de busca, como interseção e união, em comparação com bitmaps tradicionais. No estudo de Lemire et al. [2018], o Roaring Bitmap demonstrou ser mais eficiente tanto em tempo quanto em uso de memória quando comparado a outras técnicas de compressão de bitmap, como WAH e Concise.

2.2. Comparação Computacional

Para avaliar a eficiência do Roaring Bitmap, conduzimos experimentos comparativos entre essa estrutura de dados e outras amplamente utilizadas, como `set` com `sort`, `bitstring`[1], `numpy`[2], e o próprio Roaring Bitmap[3]. O experimento foi realizado no contexto de um algoritmo branch-and-bound (B&B) para a resolução do problema da clique máxima. Cada nó da árvore de

[1] `https://pypi.org/project/bitstring/`
[2] `https://numpy.org/`
[3] `https://www.roaringbitmap.org/`

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

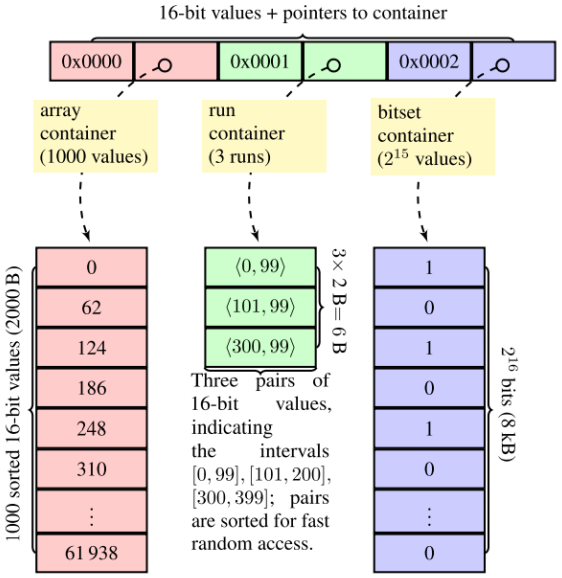


Figura 1: Bitmap Roaring contendo três contêineres: o primeiro contendo a lista dos primeiros 1000 múltiplos de 62, o segundo contendo todos os inteiros nos intervalos $[2^{16}, 2^{16} + 100)$, $[2^{16} + 101, 2^{16} + 201)$ e $[2^{16} + 300, 2^{16} + 400)$, e o terceiro contendo todos os inteiros pares em $[2 \times 2^{16}, 3 \times 2^{16})$.
Fonte: Lemire et al. [2018].

busca é representado por uma tupla (C, P, LB), onde C é a clique atual, P é o conjunto de vértices candidatos a expandir a clique, e LB é o tamanho da maior clique encontrada até o momento.

Em cada nó, aplicamos uma heurística de coloração para determinar um limite superior para o tamanho da maior clique. Essa heurística, juntamente com a atualização do conjunto de vértices candidatos, envolve operações de interseção e diferença de conjuntos, que podem ser significativamente aceleradas por meio do uso de bitmaps.

A Tabela 1 apresenta os resultados da comparação. A coluna ”Instância”identifica o grafo utilizado no teste, n representa o número de vértices, p representa a densidade do grafo, ”Nós da árvore”mostra a quantidade de subproblemas explorados, e as demais colunas indicam o tempo de execução para cada estrutura de dados.

instância	n	p	nós da árvore	set + sort	bitstring	numpy	roarbitmap
brock200_1	200	0.74	301735	83.36	117.47	80.84	7.00
brock200_2	200	0.49	3777	0.53	1.01	0.73	0.08
brock200_3	200	0.60	13411	2.73	4.67	3.09	0.27
brock200_4	200	0.65	53313	9.20	16.42	11.45	0.95

Tabela 1: Resultados dos testes computacionais das estruturas para representação de conjunto de vértices

Os resultados indicam uma diferença marcante no desempenho entre as estruturas de dados testadas. O Roaring Bitmap apresentou um tempo de execução significativamente menor, especialmente em grafos mais densos, como `brock200_1`. Isso ocorre porque, em grafos densos, a quantidade de subproblemas cresce exponencialmente, e a eficiência nas operações de interseção e união impacta diretamente o tempo total de execução.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

Em termos de desempenho, o Roaring Bitmap se destaca como a melhor escolha, superando inclusive o `numpy`, que tem forte suporte para operações vetoriais. Embora a `bitstring` seja uma estrutura otimizada para bitmaps, ela não consegue alcançar o desempenho da abordagem Roaring, devido à sua menor eficiência nas operações de compressão e manipulação de grandes conjuntos de dados.

Esses resultados confirmam que o Roaring Bitmap não só reduz o uso de memória, mas também melhora substancialmente o tempo de execução em algoritmos de B&B que fazem uso intensivo de operações sobre conjuntos de vértices.

3. Heurística de Coloração Gulosa

Nesta seção, apresentamos a heurística de coloração gulosa com paralelismo de bits, bem como uma versão relaxada.

3.1. Inicialização do Bitmap

No início do algoritmo, os vértices do grafo G são ordenados utilizando a ordem *smallest last order*, conforme proposto por Matula e Beck [1983]. Nessa ordenação, os vértices com menor grau, ou seja, menos restritos em termos de conectividade, recebem os maiores rótulos. Esse critério de ordenação tende a minimizar o número de cores utilizadas pela heurística de coloração gulosa. Essa ordenação será utilizada tanto para indexar o bitmap quanto para criar os bitmaps que representam a vizinhança de cada vértice.

3.2. Heurística de Coloração Gulosa

Dado um grafo $G = (V, E)$ e uma ordenação dos vértices $(v_1, v_2, \ldots, v_n)$, a heurística de coloração gulosa gera uma sequência de cores $\pi = (\pi_1, \ldots, \pi_n)$ tal que $color[\pi_1] \leq color[\pi_2] \leq \ldots \leq color[\pi_n]$, onde $color[\pi_i]$ representa o número de cores utilizadas para colorir o subgrafo $G[\pi_1, \ldots, \pi_i]$.

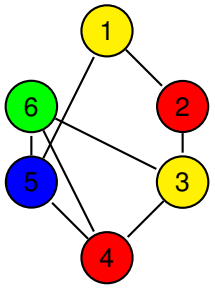


Figura 2: Coloração obtida pela heurística de coloração gulosa utilizando a sequência $(1, 2, 3, 4, 5, 6)$

Na Figura 2, temos a coloração $\{V_1 = \{1, 3\}, V_2 = \{2, 4\}, V_3 = \{5\}, V_4 = \{6\}\}$. A Tabela 2 apresenta a ordem de coloração obtida pela heurística de coloração. É importante destacar que a ordem é determinada ao considerar os vértices em cada classe de cor.

Além da ordem de ramificação, o vetor $c[\pi_i]$ fornece um limite superior para a quantidade de cores usadas para colorir o subgrafo induzido $G[\pi_1, \ldots, \pi_i]$, o que também serve como um limite superior para o tamanho da maior clique nesse subgrafo. Observe que a heurística de coloração ajusta o limite superior dinamicamente à medida que os vértices com os maiores índices são removidos, fornecendo uma estimativa mais precisa em cada etapa do algoritmo.

π	1	3	2	4	5	6
$c[\pi_i]$	1	1	2	2	3	4

Tabela 2: Ordem de ramificação e limite superior dado pela coloração gulosa

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

O Algoritmo 1 apresenta a versão da heurística de coloração gulosa com paralelismo de bits aplicada a um subgrafo $G[U]$. O conjunto de vértices U é representado por um bitmap Roaring. As operações realizadas com esses bitmaps aproveitam o paralelismo de bits para acelerar a construção das classes de cores. Enquanto na heurística convencional os vértices são coloridos sequencialmente com a menor cor disponível, na versão com paralelismo de bits as classes de cores são formadas iterativamente a partir dos primeiros vértices disponíveis, seguindo uma ordem fixa. Nas linhas 8-14, uma classe de cor é construída selecionando-se o primeiro vértice disponível usando a função $S.min()$.

Algoritmo 1: Algoritmo BITCOLOR

```
1  Function BITCOLOR(G, U):
2      π ← [];
3      color ← [];
4      R ← U.copy();
5      k ← 1;
6      while |R| > 0 do
7          S ← R.copy();
8          while |S| > 0 do
9              v ← S.min();
10             π.append(v);
11             color.append(k);
12             S ← S \ G.N(v);
13             S ← S \ {v};
14             R ← R \ {v};
15         k ← k + 1;
16     return π, color;
```

3.3. Heurística de Coloração Relaxada

Ao contrário da heurística anterior, a heurística de coloração relaxada apresentada no Algoritmo 2 recebe um inteiro k, que representa o número máximo de classes de cores a serem construídas. A heurística devolve dois conjuntos de vértices: L, que contém os vértices coloridos nas k classes geradas, e R, que contém os vértices não coloridos. O conjunto R será utilizado para a etapa de ramificação. A ideia principal deste método é reduzir o tempo de coloração, permitindo uma transição mais rápida para a fase de geração de subproblemas.

4. Algoritmo de Branch-and-Bound com Coloração Relaxada

O algoritmo de Branch-and-Bound (B&B) com heurística de coloração relaxada foi apresentado em San Segundo e Tapia [2014]. A principal ideia deste algoritmo é utilizar uma heurística de coloração que, embora forneça menos informações detalhadas, tem como vantagem um tempo de execução mais rápido. Essa eficiência é alcançada ao se definir um limite no número de cores utilizadas pela heurística de coloração.

O Algoritmo 3 descreve o B&B utilizando a heurística de coloração relaxada para o problema da clique máxima (PCM). Cada subproblema é representado por uma tripla (U, S, S_{max}), onde U é o conjunto de vértices candidatos a serem adicionados à clique atual, S é a clique parcial atual e S_{max} é a maior clique encontrada até o momento. Em cada subproblema (U, S, S_{max}), aplica-se uma heurística de coloração relaxada sobre o conjunto U, resultando em dois subconjuntos: L, contendo os vértices que foram coloridos, e R, contendo os vértices não coloridos, de forma que o número de cores $\omega(G[L])$ seja limitado a k (Linha 7).

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

Algoritmo 2: Algoritmo separa

```
1  Function separa(G, U, k):
2      R ← U.copy();
3      L ← Bitmap([]);
4      l ← 1;
5      while |R| > 0 and l ≤ k do
6          S ← R.copy();
7          while |S| > 0 do
8              v ← S.min();
9              S ← S \ G.N(v);
10             S ← S \ {v};
11             R ← R \ {v};
12             L ← L ∪ {v};
13         l ← l + 1;
14     return L, R;
```

Nas linhas 8-16, os vértices do conjunto R são escolhidos para a geração de novos subproblemas. Na Linha 9, o vértice v de maior índice, de acordo com a ordenação inicial dos vértices, é selecionado para ser adicionado à clique atual S (Linha 10). Em seguida, o conjunto candidato é atualizado (Linha 11). O novo subproblema gerado consiste em $(U \cap N(v), S \cup \{v\}, S_{max})$ (Linha 12), onde a busca prossegue de forma recursiva. Após resolver esse subproblema, o vértice v é removido do conjunto candidato U (Linha 13) e também da clique atual S (Linha 14), permitindo que outros vértices de R sejam considerados.

Algoritmo 3: Algoritmo BITCLIQUERELAXED

```
1  Function BITCLIQUERELAXED(G, U, S, S_max):
2      if U = ∅ then
3          if |S| > |S_max| then
4              S_max ← S;
5      else
6          k ← |S_max| − |S|;
7          L, R ← separa(G, U, k) ;
8          while R ≠ ∅ do
9              v ← R.max();
10             S ← S ∪ {v};
11             U_v ← U ∩ N(v);
12             BITCLIQUERELAXED(G, U_v, S, S_max) ;
13             U ← U \ {v};
14             S ← S \ {v};
```

5. Método das Bonecas Russas

O método das bonecas russas é uma alternativa interessante para resolução do PCM. O método foi primeiramente aplicado por Östergård [2002]. Dado um grafo $G = (V, E)$ e uma ordem dos vértices $V = \{v_1, v_2, \ldots, v_n\}$. O algoritmo busca iterativamente a clique máxima nos subgrafos $G[v_1], G[v_1, v_2], G[v_1, v_2, v_3], \ldots, G[v_1, v_2, \ldots, v_n]$. A intuição do método é usar soluções dos

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

problemas em grafos menores como limites para resolução dos problemas para grafos maiores.

Vaskelainen et al. [2010] aplica o método das bonecas russas em alguns problemas de otimização como o problema da cobertura tripla de Steiner (Östergård e Vaskelainen [2011]) e o problema do subtorneio transitivo máximo (Kiviluoto et al. [2016]). Gschwind et al. [2018] aplicou o método da bonecas russas no problema da k-plex máxima.

Corrêa et al. [2014] aplicaram o método das bonecas russas para resolver o problema PCM. No método das bonecas russas, resolvemos iterativamente o problema de determinar $\omega(G_i)$ para $G_i = G[v_1, \ldots, v_i]$. Note que resolver o problema $\omega(G_i)$ corresponde em decidir se $\omega(G_i) = \omega(G_{i-1})$ ou $\omega(G_i) = \omega(G_{i-1}) + 1$. Além disso, $\omega(G_i) = \omega(G_{i-1}) + 1$ somente se v_i está presente em toda clique de G_i. As principais contribuições desse trabalho são a utilização de um heurística de coloração gulosa parcial e a utilização da estratégia das bonecas russas para a resolução de cada problema de decisão (boneca). Nas próximas subseções, apresentaremos uma versão simplificada do algoritmo apresentado em Corrêa et al. [2014].

5.1. Algoritmo das Bonecas Russas

O Algoritmo 4 apresenta a estratégia de busca do algoritmo das bonecas russas para o Problema da Clique Máxima (PCM). Inicialmente, o conjunto R começa vazio (Linha 3) e o conjunto S contém todos os vértices (Linha 4). As Linhas 5-10 descrevem o processo de geração de subproblemas que serão resolvidos recursivamente. Na Linha 6, escolhe-se o primeiro vértice de S para encontrar uma clique contendo v (Linha 7). Como o vértice v pertence à clique, o conjunto candidato para expandir a clique será $R \cap N(v)$ (Linha 8). Em seguida, o vértice v é removido de S (Linha 9) e adicionado ao conjunto R (Linha 10).

Algoritmo 4: Algoritmo BONECAS RUSSAS

```
1  Function BONECA (G = (V, E)) :
2      MAX ← ∅;
3      R ← ∅;
4      S ← V;
5      while S ≠ ∅ do
6          v ← S.min();
7          C_q ← C_q + v;
8          BONECA (G, R ∩ N(v), C_q, MAX) ;
9          S ← S − v;
10         R ← R + v;
```

O Algoritmo 5 detalha a resolução de cada subproblema gerado pela estratégia de bonecas. A heurística de coloração relaxada retorna dois subconjuntos L e R, garantindo que $G[L]$ não contém uma clique maior que a atual (Linha 10). Nas Linhas 11-20, ocorre o processo de geração das bonecas menores. Um aspecto crucial é que, ao encontrar uma solução na vizinhança de v, o processo de busca pode ser interrompido imediatamente.

6. Resultados

Nesta seção, apresentamos os resultados do experimento computacional que compara o algoritmo MCS, descrito em Carmo e Züge [2012] e disponível em Carmo e Züge [2024], com o algoritmo de B&B com coloração relaxada (Seção 4) e o algoritmo de Bonecas Russas (Seção 5).

Todos os experimentos foram implementados em Python e executados em um computador com processador Intel Core i7-9750H, 16 GB de RAM e sistema operacional Ubuntu 22.04 64 bits, rodando no WSL 2.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

Algoritmo 5: Algoritmo BONECA

```
1  Function BONECA(G, U, S, S_max):
2      if U = ∅ then
3          if |S| > |S_max| then
4              S_max ← S;
5              return True;
6          else
7              return False;
8      else
9          k ← |S_max| − |S|;
10         L, R ← separa(G, U, k);
11         while R ≠ ∅ do
12             v ← R.min();
13             R ← R \ {v};
14             L ← L ∪ {v};
15             S ← S ∪ {v};
16             U_v ← L ∩ N(v);
17             flag ← BONECA(G, U_v, S, S_max);
18             if flag then
19                 return True;
20             S ← S \ {v};
```

Vértices	Densidade	B&B	Bonecas Russas	MCS	B&B Nós	Boneca Russas nós
2000	0.25	54.32	61.33	235.88	2,19E+06	2,45E+06
500	0.5	17.51	19.24	31.52	9,60E+05	9,65E+05
250	0.75	85.91	93.79	111.5	3,48E+06	3,40E+06
150	0.9	18.16	15.58	21.05	5,04E+05	3,77E+05

Tabela 3: Numero de nós na árvore de busca e tempo de execução

A Tabela 3 compara os tempos médios de execução (em segundos) e a quantidade média de nós da árvore de busca explorados pelos algoritmos B&B e Bonecas Russas.

Os experimentos foram realizados em 40 instâncias artificiais, divididas em 4 grupos, com 10 instâncias por grupo. As instâncias foram classificadas de acordo com o número de vértices e a densidade dos grafos.

Os algoritmos desenvolvidos neste trabalho mostraram-se superiores ao MCS em todas as instâncias, mesmo quando consideradas individualmente. O MCS apresentou tempos significativamente piores em instâncias com um grande número de vértices e baixa densidade.

O algoritmo de B&B mostrou-se superior na maioria das instâncias, mas em grafos extremamente densos, o algoritmo de Bonecas Russas teve um desempenho melhor. Esse comportamento também foi observado no número de nós explorados na árvore de busca: o algoritmo de Bonecas Russas foi mais eficiente em cortar subproblemas em grafos muito densos.

Uma diferença crucial entre os dois algoritmos reside na abordagem de ramificação: no algoritmo de Bonecas Russas, a expansão dos problemas é incremental, enquanto no algoritmo de B&B é decremental. Esse contraste na estratégia de ramificação pode explicar a vantagem de tempo do algoritmo de Bonecas Russas em grafos superdensos. Estudos como o de Tavares et al.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

[2016] observaram uma variação significativa no tempo de execução semelhante quando alteraram a ordem dos vértices para resolver o problema da clique máxima ponderada em grafos com densidade superior a 90%.

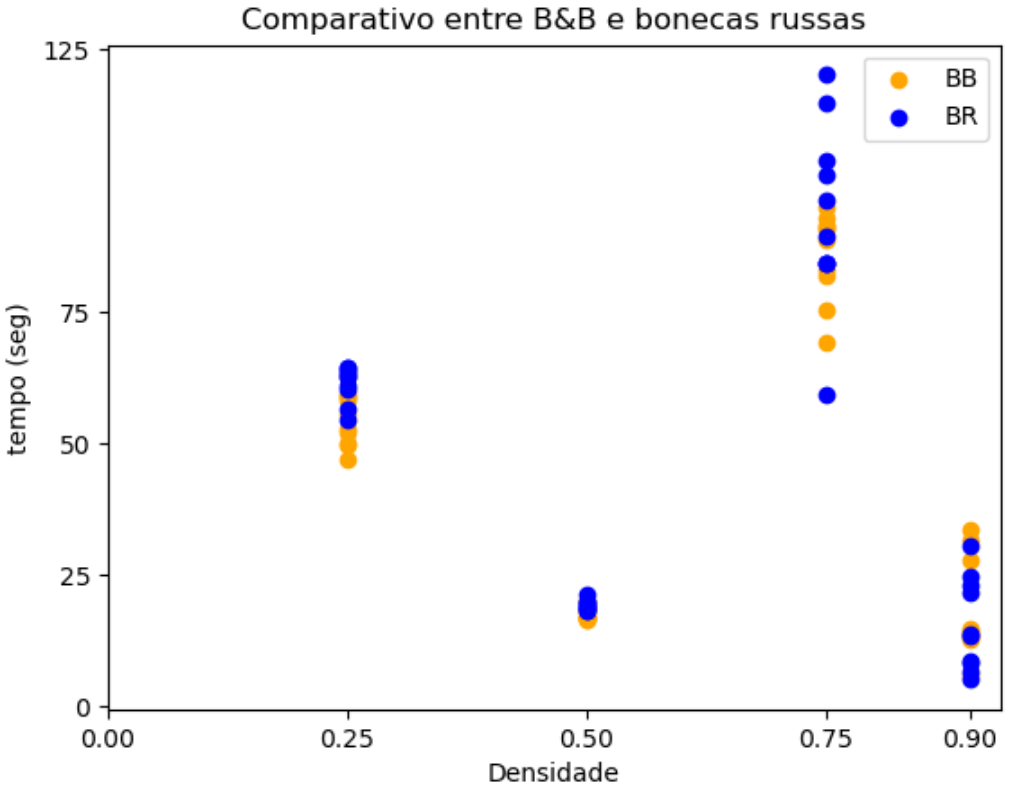


Figura 3: Gráfico de dispersão da densidade em relação ao tempo para os algoritmos B&B e bonecas russas.

A Figura 3 ilustra a dispersão dos tempos de execução em função da densidade das instâncias para os algoritmos B&B versão relaxada (BB) e Bonecas Russas (BR). Os resultados completos e o código-fonte deste trabalho podem ser acessados em Albuquerque e Tavares [2024].

7. Conclusão

Neste trabalho, comparamos o desempenho de três algoritmos para a resolução do problema da clique máxima: o algoritmo MCS, o Branch-and-Bound (B&B) com coloração relaxada e o algoritmo de Bonecas Russas. Utilizando um ambiente de teste padronizado, nossos experimentos revelaram diferenças significativas no desempenho desses algoritmos.

Os principais resultados são os seguintes:

- **Eficiência da Biblioteca Roaring Bitmaps:** Os algoritmos implementados utilizando a biblioteca de paralelismo de bits Roaring Bitmaps demonstraram um desempenho superior em comparação aos algoritmos que não a utilizaram.

- **Desempenho dos Algoritmos em Diferentes Tipos de Instâncias:** O algoritmo B&B apresentou um desempenho superior na maioria das instâncias testadas. No entanto, em grafos extremamente densos, o algoritmo de Bonecas Russas superou o B&B em termos de eficiência.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

Para trabalhos futuros, sugerimos expandir a pesquisa de Carmo e Züge [2012] implementando os algoritmos de B&B e Bonecas Russas descritos neste artigo dentro das especificações do framework unificado proposto por eles. Além disso, recomendamos explorar a aplicação da biblioteca Roaring Bitmaps a outros problemas de otimização combinatória, aproveitando seu potencial para aprimorar o desempenho.

Referências

Albuquerque, I. e Tavares, W. (2024). Cliquemaximabitmap. URL `https://github.com/IgorAlanAlbuquerque/CliqueMaximaBitmap`. GitHub repository.

Balasundaram, B., Butenko, S., e Hicks, I. V. (2011). Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142.

Battail, G. (2019). Error-correcting codes and information in biology. *BioSystems*, 184:103987.

Boginski, V., Butenko, S., e Pardalos, P. M. (2006). Mining market data: A network approach. *Computers & Operations Research*, 33(11):3171–3184.

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

Carmo, R. e Züge, A. (2012). Branch and bound algorithms for the maximum clique problem under a unified framework. *Journal of the Brazilian Computer Society*, 18:137–151.

Carmo, R. e Züge, A. (2024). maxcliquebb. URL `https://gitlab.c3sl.ufpr.br/apzuge/maxcliquebb`. Accessed: 2024-06-09.

Chambi, S., Lemire, D., Kaser, O., e Godin, R. (2016). Better bitmap performance with roaring bitmaps. *Software: practice and experience*, 46(5):709–719.

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

Das, N. R., Chaudhury, K. N., e Pal, D. (2023). Improved nmr-data-compliant protein structure modeling captures context-dependent variations and expands the scope of functional inference. *Proteins: Structure, Function, and Bioinformatics*, 91(3):412–435.

Gschwind, T., Irnich, S., e Podlinski, I. (2018). Maximum weight relaxed cliques and russian doll search revisited. *Discrete Applied Mathematics*, 234:131–138.

Hasan, M., Islam, M. R., e Mugdha, A. G. (2023). Solving maximum clique problem using chemical reaction optimization. *OPSEARCH*, 60(3):1230–1266.

Kiviluoto, L., Östergård, P. R., e Vaskelainen, V. P. (2016). Algorithms for finding maximum transitive subtournaments. *Journal of Combinatorial Optimization*, 31(2):802–814.

Lemire, D., Kaser, O., Kurz, N., Deri, L., O'Hara, C., Saint-Jacques, F., e Ssi-Yan-Kai, G. (2018). Roaring bitmaps: Implementation of an optimized software library. *Software: Practice and Experience*, 48(4):867–895.

Matula, D. W. e Beck, L. L. (1983). Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536

Okamoto, Y. (2020). Finding a maximum common subgraph from molecular structural formulas through the maximum clique approach combined with the ising model. *ACS omega*, 5(22):13064–13068.

Östergård, P. R. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207.

Östergård, P. R. e Vaskelainen, V. P. (2011). Russian doll search for the steiner triple covering problem. *Optimization Letters*, 5:631–638.

San Segundo, P., Furini, F., Álvarez, D., e Pardalos, P. M. (2023). Clisat: A new exact algorithm for hard maximum clique problems. *European Journal of Operational Research*, 307(3):1008–1025.

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

San Segundo, P. e Rodriguez-Losada, D. (2013). Robust global feature based data association with a sparse bit optimized maximum clique algorithm. *IEEE Transactions on Robotics*, 29(5):1332–1339.

San Segundo, P., Rodríguez-Losada, D., e Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581.

San Segundo, P. e Tapia, C. (2014). Relaxed approximate coloring in exact maximum clique search. *Computers & Operations Research*, 44:185–192.

Tavares, W. A., Neto, M. B. C., Rodrigues, C. D., e Michelon, P. (2016). Bitclique: um algoritmo de branch-and-bound para o problema da clique máxima ponderada. *Anais XLVIII SBPO*, p. 2429–2440.

Vaskelainen, V. et al. (2010). Russian doll search algorithms for discrete optimization problems.

Wang, H., Alidaee, B., Glover, F., e Kochenberger, G. (2006). Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*, 18:77–97.

Wu, Q. e Hao, J.-K. (2012). Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2):283–290.

Wu, Q. e Hao, J.-K. (2015). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, 42(1):355–365.

DOI: 10.59254/sbpo-2024-193536
http://live-proceedings.pantheonsite.io/p/193536