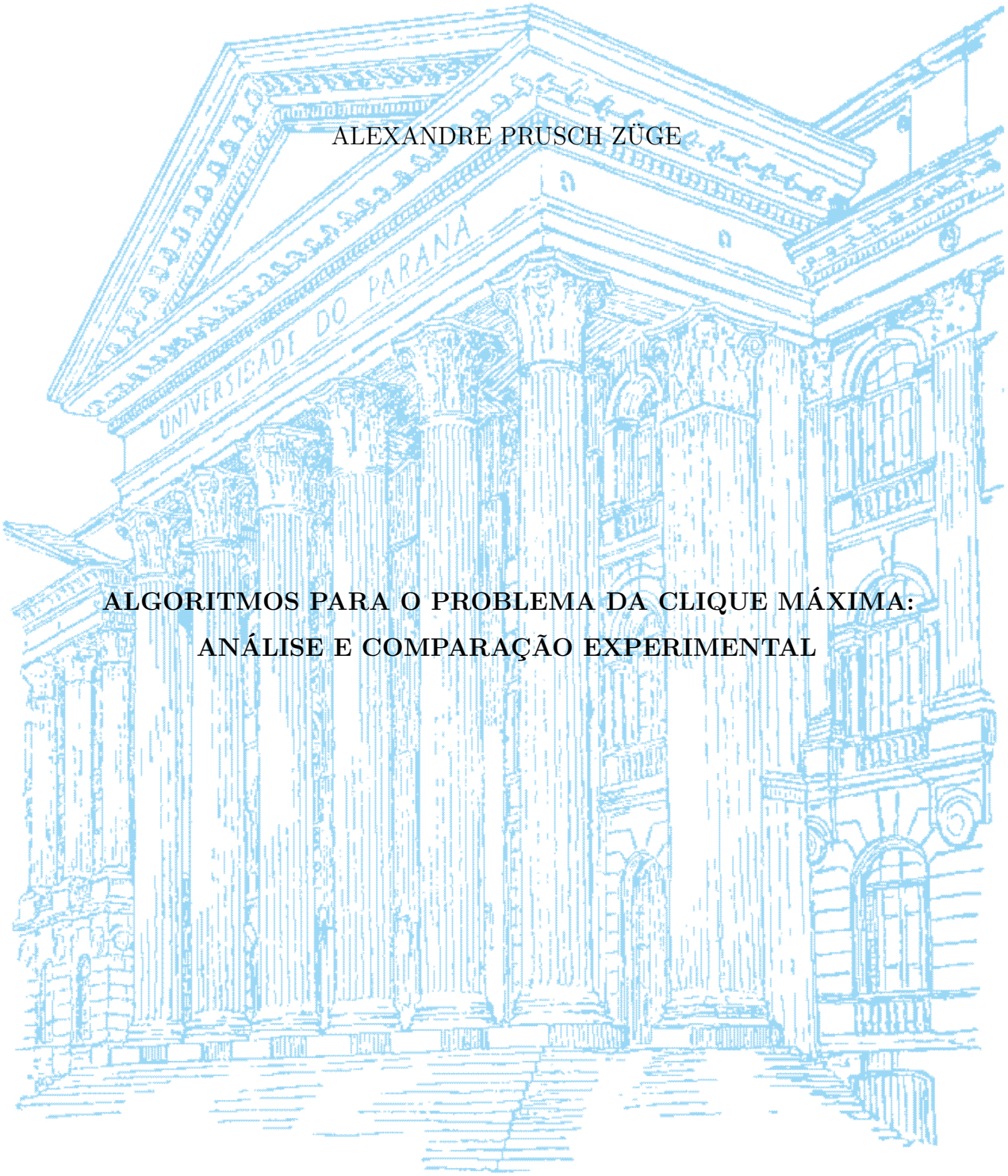
UNIVERSIDADE FEDERAL DO PARANÁ

ALEXANDRE PRUSCH ZÜGE



ALGORITMOS PARA O PROBLEMA DA CLIQUE MÁXIMA: ANÁLISE E COMPARAÇÃO EXPERIMENTAL

CURITIBA
2017

ALEXANDRE PRUSCH ZÜGE

ALGORITMOS PARA O PROBLEMA DA CLIQUE MÁXIMA: ANÁLISE E COMPARAÇÃO EXPERIMENTAL

Tese apresentada como requisito parcial para a obtenção do título de Doutor em Ciência da Computação no Programa de Pós-Graduação em Ciência da Computação, Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Prof. Dr. Renato Carmo

CURITIBA
2017

Ficha catalográfica elaborada pela Seção de Referência e Informação – UFPR – Biblioteca do Campus Jandaia do Sul (PR)

Züge, Alexandre Prusch
Z93a Algoritmos para o problema da clique máxima: análise e comparação experimental / Alexandre Prusch Züge.
Curitiba: 2017.
113 p.;il.

Orientador Renato Carmo
Tese (Doutorado em Ciência da Computação) - Universidade Federal do Paraná. Setor de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação .

1. Análise de algoritmos. 2. Comparação experimental. 3. Solução exata. Carmo, Renato, orient. II. Unversidade federal do Paraná. Setor de Ciências Exatas. Ciência da Computação. III. Título.

CDD: 005.1

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós-Graduação INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **ALEXANDRE PRUSCH ZUGE** intitulada: **Algoritmos para o Problema da Clique Máxima: análise e comparação experimental**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 28 de Setembro de 2017.

RENATO JOSÉ DA SILVA CARMO
Presidente da Banca Examinadora

JAIR DONADELLI JUNIOR
Avaliador Externo

JAYME LUIZ SZWARCFITER
Avaliador Externo

JAIME COHEN
Avaliador Externo

ANDRÉ LUIZ PIRES GUEDES
Avaliador Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

- A Deus, que me dá força e sabedoria em todos os momentos de minha vida.
- À minha esposa, Luana, por todo amor e compreensão, por me ajudar em várias etapas, por tentar entender o meu trabalho, pelos excelentes conselhos e por estar sempre ao meu lado.
- Aos meus pais, Edgar e Marilene, que sempre me apoiaram e acreditaram em mim muito além do que eu jamais irei acreditar.
- À minha irmã, Jaqueline, por todo incentivo e apoio prestados.
- Aos meus sogros, Celita e Frederico, e ao meu cunhado, Rudolfo, por todo incentivo e apoio prestados.
- Ao meu orientador, Prof. Dr. Renato Carmo, por infindáveis horas de atenção e bons conselhos.
- Aos professores das bancas de defesa e de qualificação, por sugestões, exemplos e críticas fundamentais ao meu trabalho.
- Ao meu colega Cleverson Sebastião dos Anjos pela colaboração em diversos trabalhos.
- Ao meu colega Ricardo Tavares de Oliveira pela colaboração nos trabalhos relacionados a satisfatibilidade.
- Aos colegas da UFPR – Jandaia do Sul, por todo incentivo e ajuda; em especial ao Prof. Carlos Beleti pelas contribuições no texto.
- Aos meus alunos da UFPR – Jandaia do Sul, pelo incentivo e participação nos momentos decisivos.
- A Mike Orion, Pérola, Bruce e Pequeno Schrödi: miau, miau, miau, ronrom, miau!

"I wish it need not have happened in my time," said Frodo.

"So do I," said Gandalf, "and so do all who live to see such times. But that is not for them to decide. All we have to decide is what to do with the time that is given us."

J.R.R. Tolkien

RESUMO

O problema da Clique Máxima (CM) é um problema fundamental e há uma grande motivação pela busca de algoritmos tão eficientes quanto possível para resolvê-lo de forma exata. Como esperado para um problema $\mathcal{NP}$–difícil, os melhores algoritmos com desempenho de pior caso conhecido tem custo de tempo exponencial. Por outro lado, resultados experimentais encontrados na literatura indicam que instâncias de tamanho considerável podem ser resolvidas usando algoritmos baseados na técnica de branch–and–bound. Com isso, observa-se uma distância entre os melhores resultados analíticos e os melhores resultados experimentais. Uma possível explicação para discrepância aparente entre teoria e prática foi encontrada pela análise de instâncias aleatórias. Diversos algoritmos de branch–and–bound para a solução exata do CM foram estudados, analisados e implementados. Com base nos resultados analíticos é proposta uma metodologia para comparação experimental de algoritmos, que tem como principal ponto positivo o fato de que algoritmos podem ser comparados independente de detalhes de implementação e execução. Vários algoritmos foram testados como prova de conceito. Também foram estudadas instâncias de pior caso para algoritmos de branch–and–bound que só utilizam coloração como limitante superior, resultando em um custo exponencial de tempo para estes algoritmos. Uma nova família de algoritmos foi desenvolvida, capaz de resolver tais instâncias em tempo polinomial. Recentemente, técnicas de resolvedores para problemas de satisfatibilidade têm sido aplicadas em algoritmos para CM. Tais técnicas dependem de uma redução entre os dois problemas, mas o significado em termos do grafo fica obscurecido nas descrições originais. Algumas técnicas foram estudadas e convertidas para uma descrição que não usa termos referentes aos problemas de satisfatibilidade. A implementação de vários algoritmos estudados foi disponibilizada em um repositório de acesso público.

Palavras-chave: Solução exata. Branch–and–bound. Análise de algoritmos. Comparação experimental.

ABSTRACT

The Maximum Clique problem (CM) is a fundamental problem and there is a great motivation for the development of efficient exact algorithms to solve it. As expected for a $\mathcal{NP}$–hard problem, the best algorithms where worst case analyses have been conducted present exponential running times. On the other hand, experimental results available in the literature show that instances of considerable size can be solved by branch and bound algorithms. Therefore, there is an apparent gap between the best theoretical results and the best experimental results. One possible explanation for this discrepancy between theory and practice was found through the analyses of random instances. Several exact branch and bound algorithm for CM were studied, analyzed and implemented. Based on these analytical results, a new methodology for the comparison of algorithms is proposed, where algorithms can be tested and compared regardless of implementation and execution details. Several algorithms were tested as a proof of concept. Worst case instances for some branch and bound algorithms were studied, namely algorithms that adopt only coloring-based bounding techniques to reduce the search space. These algorithms present exponential time cost for the studied instances. A new family of algorithms was developed, which is able to solve the mentioned instances in polinomial time. Recently, techniques from satisfiability solvers have been used in algorithms for CM. Such techniques depend on a reduction between the problems, and the original descriptions in terms of propositional calculus obscures their graph theoretic meaning. Some of these techniques were studied and converted to a description that uses only graph theory terminology. The implementation of several algorithms was made available in a public access repository.

Keywords: Exact solution. Branch–and–bound. Analysis of algorithms. Experimental comparison.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O problema da Clique Máxima (CM) consiste em encontrar uma clique de tamanho máximo em um grafo dado. É um problema $\mathcal{NP}$–difícil e portanto não se conhece um algoritmo polinomial que permita solucioná-lo. Ao mesmo tempo, por se tratar de um problema fundamental, existe uma grande motivação pela busca de algoritmos tão eficientes quanto possível, mesmo que não polinomiais, e suas aplicações são encontradas em múltiplas áreas (BOMZE et al., 1999).

Observando a literatura recente, existe um consenso aparente de que os melhores algoritmos para o CM são baseados na técnica de branch–and–bound. Tais trabalhos podem ser classificados em duas grandes categorias: *trabalhos analíticos*, que apresentam algoritmos com análise detalhada de complexidade; e *trabalhos experimentais*, onde algoritmos são implementados, executados e cujos resultados são reportados.

Considerando os trabalhos analíticos, os melhores algoritmos para o CM com desempenho de pior caso conhecido são algoritmos de branch–and–bound com tempo no pior caso $O(2^{0.288n})$ e $\Omega(2^{0.142n})$ e espaço polinomial (FOMIN; GRANDONI; KRATSCH, 2006) e $O(2^{n/4})$ e espaço exponencial (ROBSON, 2001) em grafos com n vértices. Todos os algoritmos de tais trabalhos possuem tempo exponencial no número de vértices do grafo no pior caso.

Do lado experimental, em Züge (2011) são apresentados um algoritmo genérico de branch–and–bound para solução do CM, oito algoritmos da literatura como modificações deste algoritmo, chamados coletivamente de família MCBB de algoritmos, e resultados experimentais para comparação entre eles. Tais algoritmos são capazes de resolver instâncias de utilidade prática, por vezes com centenas de vértices, em poucos segundos. Entretanto, não é conhecida a complexidade de tempo dos algoritmos desta família.

Assim, observa-se uma distância entre as melhores análises e os melhores resultados experimentais, e pouco se sabe sobre os motivos da existência desta distância. Com isso, torna-se aparente a necessidade da realização de estudos sobre o tema na busca da compreensão e eventual eliminação desta distância.

1.1 OBJETIVO

O objetivo deste trabalho é explorar o problema da Clique Máxima, incluindo suas instâncias e soluções, visando colaborar no entendimento da complexidade dos algoritmos, principalmente no que tange à distância observada entre análises e resultados experimentais.

1.2 PRINCIPAIS CONTRIBUIÇÕES

A principal contribuição do presente trabalho é o estudo analítico estruturado de algoritmos práticos de branch–and–bound para solução exata do problema da Clique Máxima. Todas as contribuições são derivadas de tal estudo.

Pouco está disponível na literatura sobre a complexidade de tempo dos trabalhos experimentais, e seus resultados práticos parecem ficar distantes do esperado em vista dos trabalhos analíticos. Neste trabalho é oferecida uma explicação parcial para tal fenômeno, pelo exame detalhado de várias famílias de instâncias. De particular importância são os resultados referentes a grafos aleatórios, muito utilizados para testes experimentais. Foi determinado que mesmo o pior algoritmo prático tem tempo esperado subexponencial nestes grafos. Este algoritmo realiza uma enumeração de todas as cliques do grafo.

Os resultados em grafos aleatórios foram aplicados na criação de uma metodologia para experimentação e comparação de algoritmos. A metodologia proposta tem a vantagem inédita de que resultados obtidos por diferentes implementações em diferentes ambientes computacionais podem ser comparados diretamente. Esta abordagem combate um problema comum na literatura em que resultados apresentados por diferentes autores não podem ser comparados de forma direta, um problema que é agravado pelo fato de que as implementações em geral não estão disponíveis.

Para o algoritmo que enumera todas as cliques, foram determinados os custos de tempo no pior caso, no melhor caso, no caso médio e também na família de grafos que possui o maior número possível de cliques maximais.

A adição de um limitante trivial para reduzir o número de cliques enumeradas foi examinada e os custos de tempo encontrados já apresentam uma diferença significativa em comparação com o algoritmo que gera todas as cliques.

Também foi estudado o comportamento de algoritmos de branch–and–bound que usam coloração como limitante superior, que é o caso em diversos trabalhos experimentais. Vários destes algoritmos apresentam tempo exponencial em uma família de grafos junção

disponível na literatura. Este resultado foi convertido e está reapresentado dentro do esquema de branch–and–bound deste trabalho. Foi elaborado um algoritmo que resolve tais instâncias em tempo polinomial. Levando isto em consideração, foi desenvolvida a família de algoritmos MCBBJ. Todos os algoritmos desta família podem ser apresentados e implementados como instâncias de um "meta-algoritmo". A família MCBBJ inclui os algoritmos da família MCBB e permite que eles sejam usados como base para novos algoritmos que tratam grafos junção de forma especial. Como prova de conceito, foi desenvolvido e avaliado experimentalmente o algoritmo MCJ.

Recentemente, técnicas provenientes de algoritmos de branch–and–bound para o problema da Satisfatibilidade Máxima Parcial (PMaxSat) têm sido adaptadas para algoritmos para o CM. No primeiro trabalho publicado na literatura neste ramo, uma técnica descrita em termos da lógica proposicional é adotada e suas implicações no grafo estão obscurecidas. No presente trabalho tal técnica foi estudada, relações com o grafo foram estabelecidas e ela foi recriada removendo a dependência com os problemas de satisfatibilidade. Um segundo algoritmo também foi estudado e está reapresentado aqui, visando colaborar com futuros desenvolvimentos na área. Além disso, o potencial deste ramo de pesquisa foi investigado experimentalmente pelo teste de instâncias de CM convertidas para PMaxSat em resolvedores contemporâneos.

Por fim, também é uma contribuição a implementação e disponibilização dos algoritmos estudados em um contexto unificado.

1.3 ORGANIZAÇÃO DO TRABALHO

A seguir, são apresentadas as definições e a notação adotada ao longo do texto.

Uma revisão da literatura é apresentada no Capítulo 2, incluindo resultados referentes à complexidade do problema e de alguns algoritmos, a descrição do esquema de branch–and–bound da família MCBB e de vários destes algoritmos.

Várias famílias de instâncias estudadas são compostas por grafos junção, apresentados no Capítulo 3, que são os grafos gerados pela aplicação da operação de junção de grafos. Tal operação foi implementada no *software* livre SageMath, e a implementação foi submetida e integrada à distribuição oficial do sistema. Um limitante inferior exponencial para o tempo no pior caso de alguns algoritmos, obtido pelo estudo de uma família de grafos junção, é apresentado na Seção 3.2. Como estes grafos podem ser instâncias desafiadoras

para algoritmos de branch–and–bound para CM, uma forma de tratá-los é estudada na Seção 3.3. Tal estudo resultou na criação da família MCBBJ de algoritmos, que abrange todos os algoritmos de branch–and–bound para CM apresentados neste trabalho.

No Capítulo 4, é realizado um estudo analítico sobre os algoritmos de branch–and–bound para solução do CM. Inicialmente, é dado foco para o algoritmo que executa uma enumeração exaustiva de cliques no grafo, que é o pior algoritmo razoável possível. Como esperado, é confirmado que tal algoritmo apresenta tempo exponencial no pior caso. O resultado mais importante é dado na Seção 4.1.3, onde é visto que este algoritmo tem desempenho de tempo *subexponencial* no caso médio. Este fato é substancial pois boa parte dos testes experimentais é realizada em grafos aleatórios, o que o torna um passo na compreensão da distância observada na literatura entre resultados obtidos analiticamente e resultados experimentais. Em seguida, são estudados algoritmos que utilizam técnicas para evitar enumerar todas as cliques do grafo de entrada. Um algoritmo bastante simples é estudado na Seção 4.2. Vários algoritmos de branch–and–bound usam coloração para redução do seu espaço de busca; tais algoritmos são estudados na Seção 4.3.

Recentemente, heurísticas originárias de resolvedores para o problema PMaxSat têm sido aplicadas em algoritmos branch–and–bound para o CM. Estas melhorias dependem, assim, de uma redução entre os dois problemas. No Capítulo 5, as melhorias dadas por dois algoritmos que fazem uso deste tipo de técnica são interpretadas em termos da teoria dos grafos e reescritas, permitindo que possam ser usadas diretamente em qualquer algoritmo que utilize coloração para redução do espaço de busca, sem necessidade de implementação de um resolvedor para PMaxSat. O uso de resolvedores diretamente também é investigado experimentalmente no mesmo capítulo.

Como dito anteriormente, é comum serem apresentados na literatura resultados experimentais usando grafos aleatórios. Porém, não há um padrão para a escolha dos parâmetros a serem adotados na geração dos grafos, além de ficar oculta a motivação para tal escolha. Além disso, tipicamente autores reportam o tempo de execução de uma determinada implementação em uma máquina específica, o que praticamente inviabiliza a comparação entre algoritmos. No Capítulo 6 é proposta uma metodologia para comparação experimental de algoritmos de branch–and–bound usando grafos aleatórios, cuja pontuação é baseada na análise do comportamento esperado de uma classe de algoritmos para estes grafos. A metodologia inclui tanto os parâmetros para elaboração de experimentos, como a forma de avaliação de tais experimentos, de modo que qualquer algoritmo da família

MCBB pode ser comparado diretamente, independente de características específicas de implementação e execução.

Considerações finais são apresentadas no Capítulo 7.

1.4 DEFINIÇÕES E NOTAÇÃO

Dados um conjunto S e um inteiro k, o conjunto de subconjuntos de S de tamanho k é denotado $\binom{S}{k}$. É importante observar que $\left|\binom{S}{k}\right| = \binom{|S|}{k}$. O conjunto de subconjuntos de S é denotado 2^S. Dados inteiros a, b e k, o conjunto dos primeiros k inteiros positivos é denotado $[k]$ e o conjunto de todos os inteiros entre a e b, inclusive, é denotado $[a..b]$, isto é, $[a..b] := \{z \in \mathbb{Z} \mid a \leq z \leq b\}$ e $[k] := [1..k]$. O logaritmo binário é denotado $\lg x$, isto é, $\lg x := \log_2 x$.

Um *grafo* G é um par $(V(G), E(G))$, onde $V(G)$ é um conjunto finito e $E(G) \subseteq \binom{V(G)}{2}$. Cada elemento de $V(G)$ é chamado *vértice* de G, e cada elemento de $E(G)$ é chamado *aresta* de G. O *tamanho* do grafo G é $|V(G)| + |E(G)|$. Ao longo do texto, o número de vértices em um grafo é denotado por n e o número de arestas por m.

Um vértice v é *vizinho* de u em G se $\{u, v\}$ é uma aresta de G. A *vizinhança* do vértice v em G é o conjunto de seus vizinhos em G e é denotado $\Gamma_G(v)$. O *grau* do vértice v em G é o número de vizinhos de v em G.

Dado um conjunto $Q \subseteq V(G)$, a *vizinhança comum* dos vértices de Q, denotada $\Gamma^{\cap}_G(Q)$, é o conjunto de vértices que são vizinhos de todos os vértices em Q, isto é, $\Gamma^{\cap}_G(Q) := \bigcap_{v \in Q} \Gamma_G(v)$. O caso $Q = \emptyset$ é tratado de forma especial como $\Gamma^{\cap}_G(\emptyset) := V(G)$.

Um grafo H é dito *subgrafo* de G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Dado um conjunto de vértices $S \subseteq V(G)$, o *subgrafo induzido por S em G* é o grafo $G[S] := \left(S, E(G) \cap \binom{S}{2}\right)$.

O grafo G é *completo* se $E(G) = \binom{V(G)}{2}$. Uma *clique* em G é um subconjunto de $V(G)$ que induz um subgrafo completo. Em particular, consideramos o conjunto vazio como uma clique. Um grafo completo com n vértices é denotado K_n.

Uma clique em G é *maximal* se não é subconjunto próprio de nenhuma outra clique de G. Uma clique em G é *máxima* se possui o maior número de vértices dentre todas as cliques de G. O conjunto de todas as cliques de G é denotado $\mathcal{C}(G)$. O tamanho de uma clique máxima de G é denotado $\omega(G)$, isto é, $\omega(G) := \max\{|C| \mid C \in \mathcal{C}(G)\}$.

Um conjunto de vértices S é *independente* em G se $E(G[S]) = \emptyset$. Dado um inteiro

k, uma *k-coloração* de G é uma função sobrejetora $f : V(G) \to [k]$ tal que $f^{-1}(c)$ é independente em G, para todo $c \in [k]$. Cada conjunto $f^{-1}(c)$ é chamado de *cor* c e o valor $f(v)$ é a cor de v. Uma *coloração* de G é uma k-coloração de G para algum k. O *número de cores* k em uma coloração f de G é denotado $f(G)$. O *número cromático* de G é o menor k tal que G possui uma k-coloração e é denotado $\chi(G)$. Um *coloração mínima* de G é uma k-coloração com $k = \chi(G)$. É importante observar que $\omega(G) \leq k$ para toda k-coloração de G.

Um grafo G é *perfeito* se $\omega(H) = \chi(H)$ para todo subgrafo induzido H de G.

O *grafo complementar* de G é o grafo $\overline{G} := \left(V(G), \binom{V(G)}{2} - E(G)\right)$. Observamos que uma clique S em G é um conjunto independente em $\overline{G}$. O *grafo sem arestas* com n vértices, denotado $\overline{K}_n$, é o grafo com conjunto de arestas vazio, ou seja, $\overline{K}_n := \overline{K_n}$.

O modelo $\mathcal{G}_{n,p}$ de grafos aleatórios (BOLLOBÁS, 2001) é o espaço de probabilidades formado pelos grafos cujo conjunto de vértices é $[n]$ e onde cada aresta tem probabilidade (independente) p de ocorrer.

Ao longo do texto, adotamos o sinal de ponto como separador decimal (por exemplo "2.5" no lugar do usual "2,5"), para concordar com a literatura que está praticamente na totalidade em língua inglesa.

Definimos os seguintes problemas computacionais:

Clique Máxima (CM)
Instância : Grafo G.
Resposta : Uma clique máxima de G.

k–Clique (Clique)
Instância : Grafo G, inteiro k.
Resposta : SIM, caso G contenha clique de tamanho k; NÃO, caso contrário.

Conjunto Independente Máximo (CIM)
Instância : Grafo G.
Resposta : Um conjunto independente máximo de G.

2 REVISÃO DA LITERATURA

O CM é um problema $\mathcal{NP}$–difícil (GAREY; JOHNSON, 1979). Aproximá-lo para um fator melhor que $O(n^{1-\varepsilon})$ também é $\mathcal{NP}$–difícil, para todo $\varepsilon > 0$ (ZUCKERMAN, 2006).

O problema de decisão associado ao CM é conhecido como o problema da Clique e é o problema de, dados um grafo G e um inteiro k, decidir se G tem uma clique de tamanho (pelo menos) k. O problema da Clique é um problema $\mathcal{NP}$–completo e faz parte da lista clássica de 21 problemas $\mathcal{NP}$–completos de Karp (1972). Além disso, o problema resultante de sua parametrização natural, onde k é o parâmetro, é $W[1]$–completo (DOWNEY; FELLOWS, 2012).

Um grafo com n vértices pode ter até $3^{n/3}$ cliques maximais, para n múltiplo de três (MOON; MOSER, 1965), portanto qualquer algoritmo que enumere todas as cliques maximais tem complexidade de tempo $\Omega(3^{n/3})$ no pior caso. O algoritmo CLIQUES (TOMITA; TANAKA; TAKAHASHI, 2006), que é baseado no clássico algoritmo BK (BRON; KERBOSCH, 1973) para enumeração de cliques maximais, tem complexidade de tempo de pior caso $O(3^{n/3})$. Recentemente, foi proposta uma versão refinada do algoritmo CLIQUES com melhor desempenho prático, sem perdas na complexidade de tempo (NAUDÉ, 2015). O algoritmo GP tem tempo $O(|\mathcal{C}(G)| \cdot m)$ e foi proposto junto com algumas variações, incluindo versões que usam processamento paralelo (HOU et al., 2016). Também existem algoritmos que possuem tempo polinomial por clique maximal enumerada, que podem ter melhor desempenho quando o número de cliques maximais no grafo não é próximo do pior caso. O primeiro algoritmo proposto com tal característica é o algoritmo MIS (TSUKIYAMA et al., 1977), cujo tempo para gerar cada clique maximal é $O(nm)$.

Para a solução do CM não é necessária uma enumeração completa de cliques maximais. Algumas cliques podem ser descartadas quando não forem promissoras como candidatas à clique máxima, usando a técnica chamada de *branch–and–bound* (KREHER; STINSON, 1999).

Em Tarjan e Trojanowski (1976) é apresentado um algoritmo com complexidade de tempo $O(2^{n/3})$ para o problema do Conjunto Independente Máximo (CIM). Essa complexidade foi melhorada para $O(2^{0.304n})$ (JIAN, 1986) e em seguida para $O(2^{0.296n})$ usando espaço

polinomial e $O(2^{0.276n})$ usando espaço exponencial (ROBSON, 1986). Atualmente, o melhor algoritmo disponível com custo polinomial de espaço tem complexidade de tempo no pior caso $O(2^{0.288n})$ e $\Omega(2^{0.142n})$ (FOMIN; GRANDONI; KRATSCH, 2006) e o melhor com custo de espaço exponencial é um algoritmo gerado com auxílio de um computador de tempo $O(2^{n/4})$ (ROBSON, 2001).

Em resumo, todos os algoritmos para os quais análises detalhadas de tempo no pior caso foram conduzidas estão na classe de tempo $O(2^{cn})$ para alguma constante $c > 0$. Além disso, não é relatada qualquer tentativa de implementação e avaliação experimental nos trabalhos em que tais algoritmos foram propostos.

2.1 APLICAÇÕES DO CM

Existem aplicações do CM em diversos domínios. Por exemplo, em Konc e Janežič (2007b) é apresentado um método para, dadas duas proteínas, prever que tipos de ligações elas podem realizar e em Tomita e Kameda (2007) é apresentado o algoritmo MCR, que foi aplicado para solução de problemas em bioinformática, processamento de imagens, projeto de circuitos quânticos e computação biomolecular. Outras aplicações podem ser encontradas em Bomze et al. (1999).

Uma implementação independente do algoritmo MCR foi usada no estudo de sistemas distribuídos em Duarte Jr. et al. (2010), onde milhares de instâncias com centenas de vértices foram resolvidas, geradas a partir de dados coletados no sistema para experimentação distribuída PlanetLab.

O PlanetLab consiste em uma rede de computadores conectados pela internet e espalhados geograficamente por todo o mundo. A estabilidade da conexão pode ser diferente para cada par de computadores no sistema. Experimentos que requerem o melhor desempenho possível da rede devem ser executados em um conjunto de computadores com conexão estável.

Para a descoberta de conjuntos estáveis no PlanetLab, em Duarte Jr. et al. (2010) foram geradas e resolvidas instâncias com até 461 vértices. Nestas instâncias, os vértices representam computadores e uma aresta é inserida caso a conexão entre dois computadores seja adequada segundo um determinado padrão de qualidade.

2.2 ALGORITMOS DE BRANCH–AND–BOUND PARA CM

No contexto do presente texto, a técnica de branch–and–bound consiste em uma enumeração sistemática de possíveis soluções para um problema de otimização (KREHER; STINSON, 1999). Os algoritmos de branch–and–bound empregam limitantes inferiores e superiores na busca da redução do número de soluções geradas. De modo genérico consistem em algoritmos que subdividem um problema em subproblemas menores (*branching*) e eliminam conjuntos de subproblemas que não podem levar à solução ótima com base nos limitantes superiores e inferiores (*bounding*).

Em um problema de maximização, como o CM, um *limitante superior* é um valor estimado para a solução de determinado subproblema, que é igual ou maior ao valor real de uma solução ótima. Para o passo de *bounding*, deve-se calcular um limitante superior para cada subproblema e caso este valor seja menor que um limitante inferior para o problema global, o subproblema pode ser descartado. O limitante inferior pode ser a melhor solução para algum subproblema já resolvido ou uma estimativa.

Diversos algoritmos de branch–and–bound estão disponíveis para solução do CM. Em Züge (2011) e Carmo e Züge (2012), é apresentado um contexto unificado que permite descrever, implementar e comparar o desempenho de tais algoritmos, na forma do Algoritmo MaxCliqueBB. Na realidade, este algoritmo é um meta-algoritmo e os algoritmos que resultam de instanciá-lo são coletivamente chamados de *algoritmos* MCBB.

MaxCliqueBB(G)

Entrada: um grafo G.
Saída: uma clique máxima de G.

```
1  Pré-processe o grafo G e inicialize a clique C e a pilha de estados S
2  Enquanto S ≠ ∅ faça
3  |  Desempilhe um estado de S e guarde em (Q, K)
4  |  Processe o estado (Q, K)
5  |  Enquanto K ≠ ∅ e |C| < |Q| + upper-bound(G, K, C) faça
6  |  |  Remova um vértice de K e guarde em v
7  |  |  Empilhe o estado (Q, K) em S
8  |  |  (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
9  |  |  Processe o estado (Q, K)
10 |  Se |C| < |Q| então
11 |  |  C ← Q
12 Pós-processe G
13 Devolva C
```

No Algoritmo MaxCliqueBB, o parâmetro G é o grafo de entrada do CM, cada par (Q, K) processado nas linhas 4 e 9 é chamado *estado*, a variável $\mathcal{S}$ é uma pilha de estados e o conjunto C guarda a maior clique encontrada até o momento. Cada estado contém dois conjuntos, a *clique atual* Q e o conjunto de *candidatos* K. O conjunto Q contém uma clique em G e o conjunto K contém vértices que podem ser adicionados a Q de modo a formar cliques maiores em G. Cada vértice em K é vizinho de todos os vértices de Q, isto é, $K \subseteq \Gamma_G^{\cap}(Q)$. O vértice v da linha 6 é chamado de *pivô*.

O texto indicado em itálico como "*Pré-processe*" na linha 1 representa o que na realidade são chamadas a funções que são parâmetros de MaxCliqueBB. Cada algoritmo MCBB é uma especialização de tais funções e da função upper-bound(G, K, C) da linha 5.

Seja $\mathcal{A}$ um algoritmo MCBB. As relações entre os estados definem naturalmente uma árvore $T_{\mathcal{A}}(G)$, chamada *árvore de estados* ou *espaço de busca* do algoritmo $\mathcal{A}$ com entrada G.

A árvore $T_{\mathcal{A}}(G)$ é uma árvore estritamente binária (todo nó tem zero ou dois filhos) e cada nó de $T_{\mathcal{A}}(G)$ é um estado em $\mathcal{S}$ ao longo da execução do algoritmo $\mathcal{A}$. É importante observar que uma árvore estritamente binária com x folhas tem $x - 1$ nós internos.

Mais precisamente,

1. A raiz de $T_{\mathcal{A}}(G)$ é o estado (Q, K) obtido na primeira iteração da linha 3.
2. Cada nó de $T_{\mathcal{A}}(G)$ que não é folha na árvore tem dois filhos.
   a) O *filho esquerdo* é o estado (Q, K) obtido na execução da linha 9.
   b) O *filho direito* é o estado acrescentado a $\mathcal{S}$ na linha 7 (posteriormente removido na linha 3 e processado na linha 4).

Os filhos esquerdo e direito do estado (Q, K) são denotados $E(Q, K)$ e $D(Q, K)$, respectivamente. Alternativamente, dado um estado $X = (Q, K)$, os filhos esquerdo e direito também são denotados $E(X)$ e $D(X)$. O estado (Q, K) é chamado *pai* dos estados $E(Q, K)$ e $D(Q, K)$.

Assim, dado um estado não folha $X = (Q, K)$, temos

$$E(X) = E(Q, K) = (Q \cup \{v\}, K \cap \Gamma_G(v)),$$
$$D(X) = D(Q, K) = (Q, K - \{v\}).$$

Um estado Y é *descendente* de um estado X se Y está na subárvore de $T_{\mathcal{A}}(G)$ enraizada por X. Neste caso, o estado X é um *antecedente* de Y.

O *descendente mais à direita* de um estado X, denotado $D^*(X)$, é a folha em $T_{\mathcal{A}}(G)$ dada por

$$D^*(X) := \begin{cases} X, & \text{se } X \text{ é folha,} \\ D^*(D(X)), & \text{caso contrário.} \end{cases}$$

De modo análogo, o *descendente mais à esquerda* de um estado X, denotado $E^*(X)$, é a folha em $T_{\mathcal{A}}(G)$ dada por

$$E^*(X) := \begin{cases} X, & \text{se } X \text{ é folha,} \\ E^*(E(X)), & \text{caso contrário.} \end{cases}$$

O *tamanho* de $T_{\mathcal{A}}(G)$ é o número de nós em $T_{\mathcal{A}}(G)$ e é denotado $|T_{\mathcal{A}}(G)|$.

A Figura 1 ilustra o esquema de branch–and–bound pela representação de um estado e seus filhos.

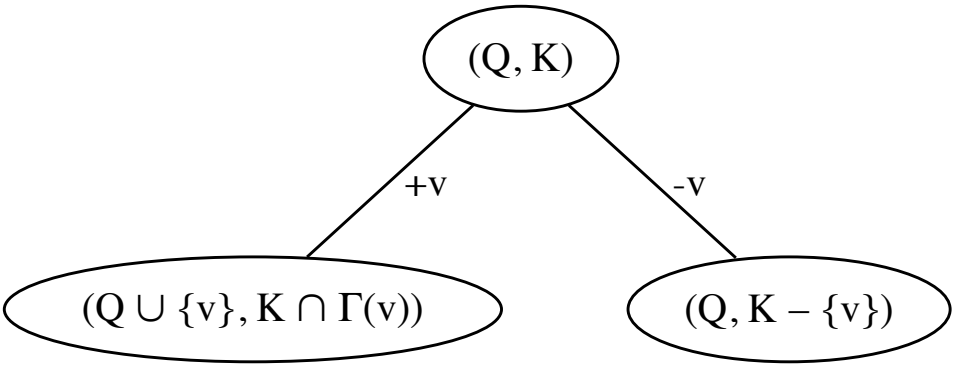


Figura 1 – Esquema de branch–and–bound para CM. O estado (Q, K) possui dois filhos, v é o pivô, "$+v$" indica o filho onde v é inserido na clique e "$-v$" o filho onde v não é inserido na clique.

2.2.1 Enumerando todas as cliques: o algoritmo nobound

O algoritmo mais simples na família MCBB é o algoritmo nobound, que consiste na ausência do cálculo de limitante superior e de uma regra específica para escolha de pivôs. O algoritmo nobound não realiza nenhuma tentativa de redução do espaço de busca e, por isso, enumera todas as cliques do grafo de entrada. Mais especificamente, cada clique C aparece exatamente uma vez em algum estado $(C, \emptyset)$, que é folha da árvore de estados.

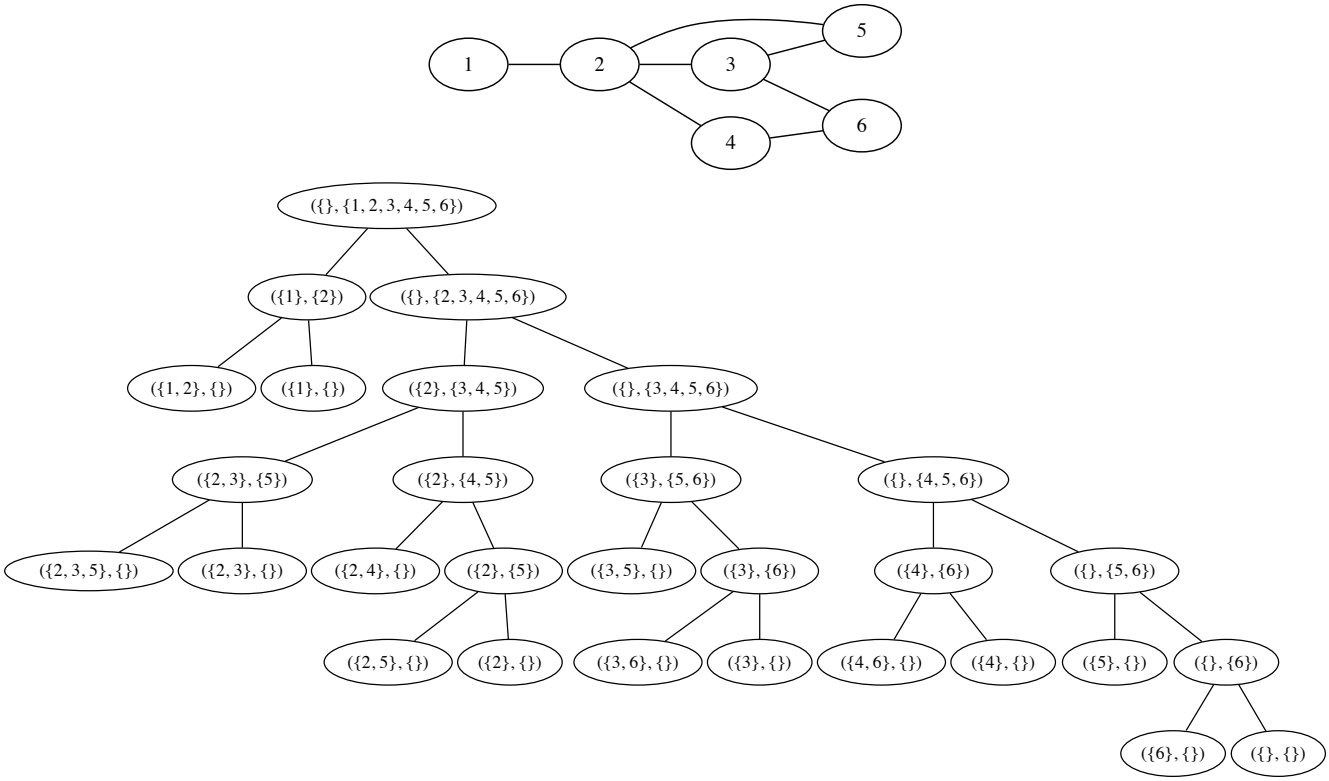


Figura 2 – Um grafo G e a árvore de estados $T_{\mathsf{nobound}}(G)$.

A Figura 2 mostra um grafo G e a árvore de estados resultante da execução de nobound para G. Para facilitar o entendimento do exemplo, o pivô escolhido é sempre o de menor valor, mas isso não é regra para o algoritmo nobound. Cada nó da árvore é um estado.

Na raiz da árvore do exemplo, é encontrado um estado $(Q = \emptyset, K = \{1, 2, 3, 4, 5, 6\})$, que representa que a clique atual (Q) está vazia e que, naturalmente, ela pode ser aumentada pela inclusão de qualquer vértice do grafo, ou seja, todos estão no conjunto de candidatos (K). Nesse estado raiz, o vértice pivô é o vértice de número 1, por isso em seu filho esquerdo encontra-se o estado $E = (\{1\}, \{2\})$, visto que o vértice de número 2 é o único vizinho do pivô, enquanto que seu filho direito é o estado $D = (\emptyset, \{2, 3, 4, 5, 6\})$. A subárvore enraizada no estado E enumera todas as cliques do grafo que contém o vértice de número 1, enquanto que a subárvore enraizada em D enumera todas as cliques que não o contém. Assim, sistematicamente todas as cliques do grafo são enumeradas e podem ser encontradas como o conjunto Q dos estados nas folhas da árvore.

Como uma especialização do Algoritmo MaxCliqueBB, o algoritmo nobound está definido no pseudocódigo a seguir.

nobound(G)

Entrada: um grafo G.
Saída: uma clique máxima de G.

```
1  C ← ∅
2  S ← pilha com um único estado (∅, V(G))
3  Enquanto S ≠ ∅ faça
4  |  Desempilhe estado de S e guarde em (Q, K)
5  |  Enquanto K ≠ ∅ faça
6  |  |  Remova vértice de K e guarde em v
7  |  |  Empilhe (Q, K) em S
8  |  |  (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
9  |  Se |C| < |Q| então
10 |  |  C ← Q
11 Devolva C
```

2.2.2 Algoritmos que empregam limitantes superiores

Para se obter um algoritmo rápido o objetivo natural é cortar o maior número possível de estados. Estratégias variadas podem ser adotadas na tentativa de atingir este objetivo. Pode-se buscar limitantes superiores justos, utilizar escolhas apropriadas de vértices pivô ou tentar reduzir de forma direta o conjunto K. De modo geral, estratégias mais complexas implicam em maior investimento de tempo por estado na árvore.

O número de vértices em K é um limitante superior trivial que pode ser usado em MaxCliqueBB. Além disso, se o conjunto K for guardado como uma sequência (por exemplo, usando um vetor), o pivô pode ser o primeiro elemento da sequência. A aplicação de tal limitante e escolha de pivô resulta no algoritmo Basic.

Como uma especialização do Algoritmo MaxCliqueBB, o algoritmo Basic está definido como pseudocódigo no Algoritmo Basic.

O algoritmo CP (CARRAGHAN; PARDALOS, 1990) prioriza na escolha de pivôs vértices de grau baixo, com o intuito de examinar e descartar rapidamente da busca vértices que não participam de uma clique máxima. Como um algoritmo MCBB, consiste na adição de uma ordenação de vértices como pré-processamento de G de modo que o grau do i-ésimo vértice é o menor em $G[\{v_1, \ldots, v_i\}]$. O limitante superior é o mesmo do algoritmo Basic.

O algoritmo DF (FAHLE, 2002) tem como motivação remover rapidamente de K vértices que estarão com certeza na clique sendo buscada e vértices que não podem estar

Basic(G)

Entrada: um grafo G.
Saída: uma clique máxima de G.

```
1  C ← ∅
2  S ← pilha com um único estado (∅, V(G))
3  Enquanto S ≠ ∅ faça
4  |  Desempilhe estado de S e guarde em (Q, K)
5  |  Enquanto K ≠ ∅ e |C| < |Q| + |K| faça
6  |  |  Remova o primeiro vértice de K e guarde em v
7  |  |  Empilhe (Q, K) em S
8  |  |  (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
9  |  Se |C| < |Q| então
10 |  |  C ← Q
11 Devolva C
```

nela. Estes vértices são detectados com base nos graus em $G[K]$. Primeiramente, todos os vértices com grau inferior ao mínimo necessário para estar em uma clique maior que a maior atual são removidos. Em seguida, todos os vértices que são vizinhos de todos os outros em K, ou seja, têm grau igual a $|K| - 1$, são movidos para Q.

2.2.3 Coloração como limitante superior

Cada vértice de uma clique deve receber uma cor diferente em qualquer coloração de um grafo, então o número de cores em uma coloração é um limitante superior para o tamanho da clique máxima. O problema de encontrar uma coloração de um grafo com o menor número possível de cores é $\mathcal{NP}$–difícil (GAREY; JOHNSON, 1979), então os algoritmos que utilizam coloração como limitante superior empregam heurísticas para colorir o grafo.

Diversos algoritmos da família MCBB usam coloração como limitante superior, por vezes em conjunto com outras técnicas e outros limitantes. O algoritmo χ, apresentado juntamente com o algoritmo DF (FAHLE, 2002), computa quatro colorações e utiliza como limitante a melhor, isto é, aquela com o menor número de cores. Unindo-se os algoritmos DF e χ obtém-se o algoritmo χ + DF (FAHLE, 2002).

O primeiro algoritmo proposto da família que usa coloração é o algoritmo MCLIQ (TOMITA; KOHATA; TAKAHASHI, 1988). Esse algoritmo foi sucessivamente atualizado, dando origem aos algoritmos MCQ (TOMITA; SEKI, 2003), MCR (TOMITA; KAMEDA, 2007), MCS (TOMITA et al., 2010) e MCT (TOMITA et al., 2016), reapresentados

recentemente em um trabalho unificado (TOMITA, 2017), além do algoritmo $k5_$MCT (TOMITA et al., 2017). O algoritmo MCQ também é a base para o algoritmo MaxCliqueDyn (KONC; JANEŽIČ, 2007a).

Por sua simplicidade e por ser um "antecedente" direto de vários algoritmos disponíveis na literatura, o algoritmo MCLIQ é um bom representante da classe dos algoritmos que usam coloração e está apresentado nesta seção.

O algoritmo MCLIQ utiliza a coloração definida no Algoritmo Guloso como limitante superior e também para especificar a ordem em que os vértices são escolhidos como pivô. O Algoritmo Guloso colore um vértice de cada vez, atribuindo a cor de menor número possível. Esse algoritmo de coloração possui a característica marcante de que os vértices possuem vizinhos em todas as cores com número menor que o da sua cor. O pseudocódigo desta coloração gulosa para um grafo G está definido no Algoritmo Guloso, o qual devolve uma lista de cores onde cada cor é um conjunto de vértices.

Guloso(G)

Entrada: um grafo G.
Saída: uma coloração de G.

```
1  maiorcor ← 1
2  P[1] ← ∅
3  Para todo v ∈ V(G) faça
4  |   cor ← 1
5  |   Enquanto cor ≤ maiorcor e P[cor] ∩ Γ(v) ≠ ∅ faça
6  |   |   cor ← cor + 1
7  |   Se cor > maiorcor então
8  |   |   maiorcor ← cor
9  |   |   P[maiorcor] ← ∅
10 |   P[cor] ← P[cor] ∪ {v}
11 Devolva P
```

Para um vértice fazer parte de uma clique grande, ele deve ter vizinhos em uma quantidade grande de cores, já que todos os vértices de uma clique são coloridos com cores diferentes. Assim, MCLIQ escolhe como pivô sempre um vértice da cor de maior número, na tentativa de encontrar cliques grandes rapidamente. Escolher o pivô desta maneira também resulta em mais uma vantagem: o filho direito de um estado não precisa ser colorido, pois a coloração é igual a do pai exceto pelo fato de o pivô ter de ser removido. Assim, o descendente mais à direita de um estado $(Q, K = (k_1, k_2, \ldots, k_i))$ é um estado $(Q, K = (k_1, k_2, \ldots, k_j))$, com $j \leq i$, e as cores dos vértices $k_1, k_2, \ldots, k_j$ são as mesmas

em ambos os estados. Assim, os números das cores de cada vértice deve ser guardado e o Algoritmo Guloso só precisa ser chamado na raiz da árvore de estados e nos estados que são filhos esquerdos.

Inicialmente, MCLIQ ordena os vértices de G em ordem não crescente de graus. Esta ordenação é realizada porque o Algoritmo Guloso tende, empiricamente, a precisar de um número menor de cores desta forma do que para ordens arbitrárias. Além disso, vértices que compartilham a mesma cor em um estado e em seu filho esquerdo são mantidos no estado filho na mesma ordem relativa como estavam no estado pai, objetivando propagar a ordem inicial ao longo da busca.

O algoritmo MCLIQ associa um número a cada vértice v de um conjunto K, denotado $N(K, v)$, que contém a cor de v na coloração de $G[K]$. Este número é utilizado para simplificar a função upper-bound(G, K, C), que apenas devolve $N(K, u)$, onde u é o último vértice de K. Os estados são coloridos e processados como no Algoritmo Processa.

MCLIQ(G)

Entrada: um grafo G.
Saída: uma clique máxima de G.

```
1  C ← ∅
2  S ← pilha com um único estado (∅, V(G))
3  Processa(G, ∅, V(G))
4  Enquanto S ≠ ∅ faça
5  |  Desempilhe estado de S e guarde em (Q, K)
6  |  Enquanto K ≠ ∅ e |C| < |Q| + max{N(K, u), u ∈ K} faça
7  |  |  Remova o último vértice de K e guarde em v
8  |  |  Empilhe (Q, K) em S
9  |  |  (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
10 |  |  Processa(G, Q, K)
11 |  Se |C| < |Q| então
12 |  |  C ← Q
13 Devolva C
```

Os algoritmos MCQ, MaxCliqueDyn, MCR, MCS, MCT e $k5_$MCT são desenvolvidos a partir de mudanças pontuais no algoritmo MCLIQ. Nenhum destes algoritmos computa a coloração no estado raiz da árvore, isto é, para o grafo inteiro. O algoritmo MCR melhora a ordenação de vértices no estado raiz. O algoritmo MaxCliqueDyn reordena os vértices antes de computar a coloração em alguns estados selecionados. O algoritmo MCS apresenta melhorias baseadas na ordenação introduzida pelo algoritmo MCR, e também modificações

Processa(G, Q, K)

Entrada: um grafo G, uma clique Q em G, um conjunto K de vértices em G.

```
1 P ← Guloso(G[K])
2 i ← 0
3 Para cor ← 1 até |P| faça
4 |   Para todo v ∈ P[cor] faça
5 |   |   K[i] ← v
6 |   |   N(K, v) ← cor
7 |   |   i ← i + 1
```

na coloração. O algoritmo MCT atualiza o algoritmo MCS de duas formas: o conjunto C é inicializado no início do algoritmo com uma clique encontrada usando o algoritmo de busca local KLS (KATAYAMA; HAMAMOTO; NARIHISA, 2005) e o conjunto de candidatos é reordenado em alguns estados selecionados antes de ser colorido. Por fim, o algoritmo $k5$_MCT é obtido a partir do algoritmo MCT por uma alteração no algoritmo de coloração.

2.2.4 Limitantes infra-cromáticos

Apesar de bastante utilizado, é importante observar que o limitante superior dado por uma coloração pode não ser justo, pelo menos por dois motivos distintos. Em primeiro lugar, um grafo pode ter o número cromático arbitrariamente afastado do tamanho da menor clique (GYÁRFÁS; SEBŐ; TROTIGNON, 2012). Em segundo lugar, como o problema de encontrar uma coloração mínima é $\mathcal{NP}$–difícil, valor dado pelo limitante superior pode também estar distante do valor ótimo.

Para tentar superar as limitações inerentes ao uso de colorações, recentemente foi proposto o uso de limitantes superiores derivados de técnicas oriundas de resolvedores de problemas de satisfatibilidade, como apresentado no Capítulo 5. Como tais limitantes podem ser mais justos que o número cromático do grafo, eles têm sido chamados de *limitantes infra-cromáticos* (SAN SEGUNDO; NIKOLAEV; BATSYN, 2015). Vários trabalhos estão disponíveis na literatura motivados por tal abordagem (LI; QUAN, 2010b; LI; QUAN, 2010a; LI; FANG; XU, 2013; LI; JIANG; XU, 2015; MASLOV; BATSYN; PARDALOS, 2014; SAN SEGUNDO; TAPIA; LOPEZ, 2013; SAN SEGUNDO et al., 2016b; SAN SEGUNDO et al., 2016a; SAN SEGUNDO et al., 2017; LI; JIANG; MANYÀ, 2017).

3 INSTÂNCIAS DE PIOR CASO PARA ALGORITMOS BASEADOS EM COLORAÇÃO: ANÁLISE E SOLUÇÕES

Várias famílias de grafos de interesse para estudo do CM são geradas usando a operação de *junção de grafos* (do inglês, *graph join*), apresentada na Seção 3.1. Por exemplo, os grafos de Moon–Moser, que são gerados usando junção e estão analisados nas Seções 4.1.2, 4.2.2 e 4.3.2, são os grafos com o maior número de cliques maximais para um determinado número de vértices (MOON; MOSER, 1965).

Diversos algoritmos da família MCBB usam apenas coloração como limitante superior. Para essa classe de algoritmos, há uma família de grafos gerados usando junção que exige a criação de uma árvore de estados de tamanho exponencial no número de vértices no grafo de entrada (LAVNIKEVICH, 2013). Por consequência, todos estes algoritmos executam em tempo exponencial no pior caso. Esta família de instâncias é analisada dentro do esquema de branch–and–bound de MaxCliqueBB na Seção 3.2.

Apesar de representar um limitante inferior exponencial para os algoritmos acima citados, estes grafos podem ser resolvidos em tempo polinomial por algoritmos que sejam capazes de detectá-los. Uma nova família de algoritmos que expande a família MCBB e é capaz de evitar o comportamento exponencial para estes grafos é desenvolvida na Seção 3.3.

3.1 JUNÇÃO DE GRAFOS

A junção de grafos é uma operação binária entre grafos que cria um grafo a partir de dois grafos e será denotada pelo símbolo "+". Dados dois grafos disjuntos G_1 e G_2, a junção de G_1 e G_2 é o grafo $G_1 + G_2$ que contém todos os vértices de G_1 e G_2 e todas as arestas de G_1 e G_2, além de arestas de modo que todos os vértices de G_1 são vizinhos de todos os vértices de G_2 (HARARY, 1969). Ou seja,

$$V(G_1 + G_2) := V(G_1) \cup V(G_2),$$
$$E(G_1 + G_2) := E(G_1) \cup E(G_2) \cup \{\{v_1, v_2\} \mid v_1 \in V(G_1), v_2 \in V(G_2)\}.$$

O grafo resultante da operação de junção é chamado *grafo junção*. A Figura 3 ilustra a operação de junção de grafos.

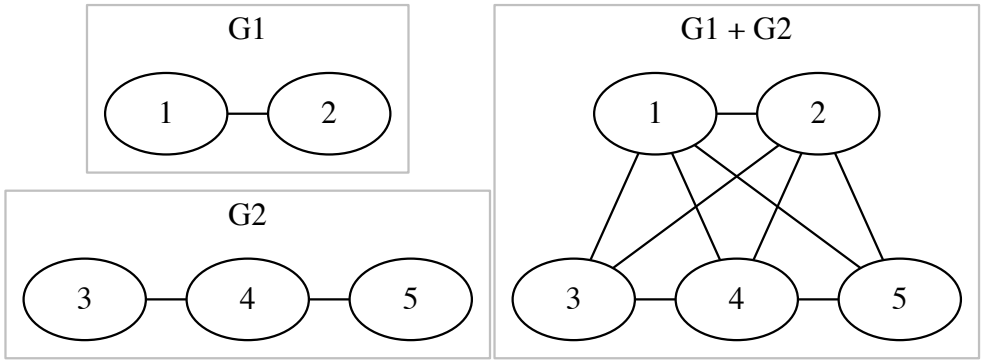


Figura 3 – Dois grafos e a junção entre eles.

No grafo junção $G_1 + G_2$, todos os vértices provenientes de G_1 passam a ser vizinhos de todos os vértices provenientes de G_2. Portanto, em uma coloração deste grafo, nenhuma cor aplicada em algum vértice oriundo de G_1 pode ser aplicada em vértices oriundos de G_2, e vice-versa. Além disso, uma clique máxima do grafo $G_1 + G_2$ é uma união de cliques máximas de G_1 e G_2. Assim, o grafo $G_1 + G_2$ tem as seguintes características:

$$|V(G_1 + G_2)| = |V(G_1)| + |V(G_2)|,$$
$$\omega(G_1 + G_2) = \omega(G_1) + \omega(G_2),$$
$$\chi(G_1 + G_2) = \chi(G_1) + \chi(G_2).$$

A operação de junção de grafos é comutativa, ou seja,

$$G_1 + G_2 = G_2 + G_1.$$

A operação de junção de grafos é associativa, ou seja,

$$(G_1 + G_2) + G_3 = G_1 + (G_2 + G_3).$$

Como a junção de grafos é comutativa e associativa, a operação de junção pode ser generalizada para $k > 2$ grafos, o que resulta no grafo junção $\sum_{i=1}^{k} G_i := G_1 + G_2 + \ldots + G_k$ com os seguintes conjuntos:

$$V\left(\sum_{i=1}^{k} G_i\right) := \bigcup_{i\in[k]} V(G_i),$$
$$E\left(\sum_{i=1}^{k} G_i\right) := \bigcup_{i\in[k]} E(G_i) \cup \bigcup_{i\in[k-1],j\in[i+1..k]} \{\{v_x, v_y\} \mid v_x \in V(G_i), v_y \in V(G_j)\}.$$

No grafo junção $G_1 + G_2 + \ldots + G_k$, cada vértice de um grafo G_i é vizinho do todo vértice de outro grafo G_j, então o grafo resultante da junção tem as seguintes

características:

$$\left|V\left(\sum_{i=1}^{k} G_i\right)\right| = \sum_{i=1}^{k} |V(G_i)|,$$
$$\omega\left(\sum_{i=1}^{k} G_i\right) = \sum_{i=1}^{k} \omega(G_i),$$
$$\chi\left(\sum_{i=1}^{k} G_i\right) = \sum_{i=1}^{k} \chi(G_i).$$

O número de cliques de um grafo junção é necessário para alguns resultados do Capítulo 4 e está determinado no Teorema 1.

Teorema 1. *Seja G_i um grafo para cada $i \in [k]$, então*

$$\left|\mathcal{C}\left(\sum_{i=1}^{k} G_i\right)\right| = \prod_{i=1}^{k} |\mathcal{C}(G_i)|.$$

Demonstração. Cada clique $C \in \mathcal{C}(G_1 + G_2 + \ldots + G_k)$ é formada pela união de uma clique de cada grafo original G_i, isto é,

$$C = C_1 \cup C_2 \cup \ldots \cup C_k, \text{para algum } C_1 \in \mathcal{C}(G_1), C_2 \in \mathcal{C}(G_2), \ldots, C_k \in \mathcal{C}(G_k).$$

Assim, existe uma bijeção natural entre as cliques de $G_1 + G_2 + \ldots + G_k$ e o produto cartesiano $\mathcal{C}(G_1) \times \mathcal{C}(G_2) \times \ldots \times \mathcal{C}(G_k)$, e o tamanho do conjunto de cliques é

$$|\mathcal{C}(G_1 + G_2 + \ldots + G_k)| = \prod_{i=1}^{k} |\mathcal{C}(G_i)|.$$

□

Dados um grafo G e um inteiro k, a *k-cópia* de G é o grafo G^k dado por

$$V(G^k) := \{k\} \times V(G),$$
$$E(G^k) := \{\{(k,u),(k,v)\} \mid \{u,v\} \in E(G)\}.$$

Para cada $q > 0$, o grafo qG é o grafo resultante da junção de q cópias distintas de G, isto é,

$$qG := \sum_{i=1}^{q} G^i.$$

É importante observar que

$$|V(qG)| = q|V(G)|,$$
$$\omega(qG) = q\omega(G),$$
$$\chi(qG) = q\chi(G).$$

Além disso, pelo Teorema 1, o número de cliques em qG é

$$|\mathcal{C}(qG)| = |\mathcal{C}(G)|^q.$$

3.1.1 Implementação da operação no SageMath

O SageMath (THE SAGE DEVELOPERS, 2016) é um *software* matemático que possui recursos em diversas áreas, como teoria dos grafos, álgebra, combinatória, cálculo e estatística. A missão do SageMath é "criar uma alternativa livre de código aberto viável a Magma, Maple, Mathematica e Matlab". Foi desenvolvido a partir da linguagem Python pela inclusão de diversos pacotes já existentes, como NumPy, SciPy e Sympy, e também pela adição de ferramentas como matplotlib, Maxima, GAP e R. O SageMath é um *software* livre distribuído sob licença GPL.

A operação de junção de grafos não estava disponível no SageMath, como pode ser visto no *Graph Theory Roadmap*[1] do sistema. Por ser *software* livre, o SageMath é desenvolvido colaborativamente e qualquer usuário do sistema pode alterá-lo. O desenvolvimento do SageMath é gerenciado com o sistema trac[2].

A necessidade de inclusão da operação de junção de grafos foi registrada como uma "solicitação de melhoria" no trac[3]. A operação de junção foi implementada na forma do método `join()` da classe `Graph` e submetida para inserção na distribuição oficial do SageMath. Esta implementação foi incluída na versão 5.12 do SageMath, lançada em 23 de outubro de 2013. Detalhes sobre o uso são encontrados na documentação[4].

3.2 UM LIMITANTE INFERIOR PARA A COMPLEXIDADE DE TEMPO DE UMA CLASSE DE ALGORITMOS

Diversos algoritmos da família MCBB usam apenas coloração como limitante superior. Para esta classe de algoritmos, um limitante inferior para a complexidade de tempo de pior caso é apresentado em Lavnikevich (2013). Nessa seção, tal limitante é expressado dentro do esquema de branch–and–bound de MaxCliqueBB.

Os grafos usados para a análise de pior caso são a família qC_5, apresentada a seguir.

---

[1] ver <http://trac.sagemath.org/sage_trac/wiki/GraphTheoryRoadmap>.
[2] ver <http://trac.edgewall.org/>.
[3] ver <http://trac.sagemath.org/ticket/14837>.
[4] ver <http://www.sagemath.org/doc/reference/graphs/sage/graphs/graph.html#sage.graphs.graph.Graph.join>.

3.2.1 A família de grafos qC_5

O grafo circuito com cinco vértices é denotado C_5, isto é,

$$V(C_5) = \{1, 2, 3, 4, 5\},$$
$$E(C_5) = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 1\}\}.$$

A família qC_5 são os grafos resultantes da junção de q grafos C_5. Assim,

$$|V(qC_5)| = 5q,$$
$$\omega(qC_5) = 2q,$$
$$\chi(qC_5) = 3q.$$

Qualquer coloração mínima de C_5 tem três cores no total, onde duas cores contêm dois vértices e uma cor é unitária, isto é, contém um único vértice. Portanto, qualquer coloração mínima de qC_5 tem $2q$ cores com dois vértices e q cores unitárias.

3.2.2 Análise dos algoritmos MCBB que usam coloração como limitante no grafo qC_5

Queremos provar a Proposição 1.

Proposição 1. *Seja T a árvore de estados resultante da execução de uma especialização de* MaxCliqueBB(qC_5) *onde,*

1. *após a execução da linha 1, C é uma clique máxima de qC_5,*
2. upper-bound$(qC_5, K, C) = \chi(qC_5[K])$,
3. *o pivô da linha 6 vem de uma cor unitária de uma coloração mínima.*

Então $|T| = 2^{q+1} - 1$.

Como queremos obter um limitante inferior para o pior caso do algoritmo, permitimos que ele obtenha a clique máxima logo na linha 1 e que o limitante superior do algoritmo seja uma coloração mínima. Além disso, todo pivô é escolhido em uma cor unitária, o que garante que o limitante superior dos filhos é mais justo do que o do pai. Assim, qualquer algoritmo que use apenas coloração como limitante superior pode apresentar resultados iguais ou piores que os obtidos aqui, mas não melhores.

Por simplicidade, dizemos que o *número de cores de um estado* (Q, K), denotado $\chi(Q, K)$, é o número de cores da coloração mínima do grafo induzido por K. Ou seja, $\chi(Q, K) := \chi(qC_5[K])$.

A raiz de T é $(\emptyset, V(qC_5))$.

Então

$$\mathsf{upper\text{-}bound}(qC_5, V(qC_5), C) = \chi(qC_5[V(qC_5)]) = \chi(qC_5) = 3q,$$

e

$$E(\emptyset, V(qC_5)) = (\{v\}, \Gamma_{qC_5}(v)),$$
$$D(\emptyset, V(qC_5)) = (\emptyset, V(qC_5) - \{v\}).$$

onde v está em uma cor unitária.

Consequentemente, o número de cores dos filhos de $(\emptyset, V(qC_5))$ é menor do que o de $(\emptyset, V(qC_5))$, isto é,

$$\chi(\{v\}, \Gamma_{qC_5}(v)) \leq 3q - 1,$$
$$\chi(\emptyset, V(qC_5) - \{v\}) \leq 3q - 1.$$

3.2.2.1 Propriedades do filho esquerdo de um estado

O filho esquerdo de um estado qualquer de T tem como propriedades importantes os fatos de que o conjunto Q tem um vértice a mais que o estado pai e que o número de cores é dois a menos que o do pai, como apresentado no Lema 2.

Lema 2. *Seja (Q, K) um estado de T e seja $(Q_E, K_E) = E(Q, K)$ o filho esquerdo de (Q, K). Então*

$$|Q_E| = |Q| + 1,$$
$$\chi(Q_E, K_E) = \chi(Q, K) - 2.$$

Demonstração. O tamanho de Q aumenta em uma unidade pois v é inserido na clique.

O número de cores diminui de dois devido à estrutura do grafo e do fato de que v é obrigatoriamente escolhido em uma cor unitária. Seja C_5^i a i-cópia de C_5 tal que $V(C_5^i) = \{v, u_1, u_2, u_3, u_4\}$ e $E(C_5^i) = \{\{v, u_1\}, \{u_1, u_2\}, \{u_2, u_3\}, \{u_3, u_4\}, \{u_4, v\}\}$. Os vértices de $V(C_5^i)$ são coloridos no processamento de (Q, K) em três cores, representadas

pelos conjuntos $\{v\}$, $\{u_1, u_3\}$ e $\{u_2, u_4\}$. No filho esquerdo, os vértices v, u_2 e u_3 não estão presentes em K_E, e os vértices u_1 e u_4 são coloridos com apenas uma cor. Como o restante da coloração é igual no pai e no filho esquerdo, o número de cores no estado filho reduz de dois em comparação com o pai. □

Como exemplo das propriedades apresentadas no Lema 2, considere o grafo

$$C_5 = (\{v_1, v_2, v_3, v_4, v_5\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_5\}, \{v_5, v_1\}\}).$$

No estado raiz, $(Q, K) = (\emptyset, \{v_1, v_2, v_3, v_4, v_5\})$, o grafo C_5 requer três cores na coloração, por exemplo, como nos conjuntos $\{v_1, v_3\}$, $\{v_2, v_4\}$ e $\{v_5\}$. Como o pivô é escolhido em uma cor unitária, necessariamente ele deve ser v_5.

Seja $(Q_E, K_E) = E(\emptyset, \{v_1, v_2, v_3, v_4, v_5\}) = (\{v_5\}, \{v_1, v_4\})$. Assim, os vértices v_2 e v_3 não estão presentes no conjunto K_E e os vértices v_1 e v_4 podem ser coloridos com a mesma cor. Ou seja, o número de cores no estado (Q_E, K_E) é igual a um, assim como o tamanho de Q_E, isto é, $\chi(Q_E, K_E) = \chi(Q, K) - 2$ e $|Q_E| = |Q| + 1$.

3.2.2.2 Propriedades do filho direito de um estado

No filho direito de um estado qualquer de T, o conjunto Q tem um vértice a mais que o estado pai e que o número de cores é um a menos que o do pai, como apresentado no Lema 3.

Lema 3. *Seja (Q, K) um estado de T e seja $(Q_D, K_D) = D(Q, K)$ o filho direito de (Q, K). Então*

$$|Q_D| = |Q|,$$
$$\chi(Q_D, K_D) = \chi(Q, K) - 1.$$

Demonstração. Como v foi escolhido em cor unitária, o lema segue diretamente do fato de que $D(Q, K) = (Q, K - \{v\})$. □

3.2.2.3 Limitante inferior para o pior caso

O Teorema 4 prova o que queríamos com a Proposição 1. Como consequência, o Corolário 5 mostra que todo algoritmo da família MCBB que adote apenas coloração como limitante superior tem tempo exponencial no pior caso na ordem de $\Omega(2^{n/5})$.

Teorema 4. *A árvore de estados T possui $2^{q+1} - 1$ nós.*

Demonstração. Sejam (Q, K) um estado de T, (Q_E, K_E) e (Q_D, K_D) os filhos de (Q, K), $(Q_E, K_E) = E(Q, K)$ e $(Q_D, K_D) = D(Q, K)$, e seja $L(Q, K) := |Q| + \mathsf{upper\text{-}bound}(G, K, C)$ a soma da linha 5 do Algoritmo MaxCliqueBB.

Pelo Lema 2,

$$L(Q_E, K_E) = L(Q, K) - 1$$

e pelo Lema 3,

$$L(Q_D, K_D) = L(Q, K) - 1$$

isto é, em todo estado filho a soma $L(Q, K)$ é reduzida de um em relação ao pai.

A raiz de T é o estado $(\emptyset, V(qC_5))$ e

$$L(\emptyset, V(qC_5)) = 3q. \tag{3.1}$$

Os filhos da raiz são os estados $E(\emptyset, V(qC_5))$ e $D(\emptyset, V(qC_5))$ e

$$L(E(\emptyset, V(qC_5))) = L(D(\emptyset, V(qC_5))) = 3q - 1. \tag{3.2}$$

Seja $P(Q, K)$ o pai do estado (Q, K) e $N(Q, K)$ o *nível* do estado na árvore, definido por

$$N(Q, K) := \begin{cases} 0, & \mathsf{se}\ (Q, K) = (\emptyset, V(qC_5)), \\ N(P(Q, K)) + 1, & \text{caso contrário.} \end{cases}$$

Assim, há um estado no nível zero (raiz), dois estados no nível um (filhos da raiz), quatro estados no nível dois e, em geral, 2^i estados no nível i. Em outras palavras, a árvore T é uma árvore binária completa, desde que todas as folhas estejam no mesmo nível.

Além disso, generalizando as equações (3.1) e (3.2), obtém-se

$$L(Q, K) = 3q - N(Q, K). \tag{3.3}$$

Todo estado (Q', K') em que

$$L(Q', K') = \omega(qC_5) = 2q \tag{3.4}$$

não possui filhos pela condição da linha 5 e, portanto, é folha na árvore. Substituindo-se (3.4) em (3.3) obtém-se

$$2q = 3q - N(Q', K') \quad \Rightarrow \quad N(Q', K') = q.$$

Assim, todas as folhas estão no nível q e total de nós é

$$|T| = \sum_{i=0}^{q} 2^i = 2^{q+1} - 1.$$

□

Com isso, o Teorema 4 prova o que queríamos na Proposição 1.

Corolário 5. *Qualquer algoritmo da família* MCBB *que adote apenas coloração como limitante superior tem tempo no pior caso* $\Omega(2^{n/5})$.

Demonstração. Pela Proposição 1, o melhor algoritmo possível que adota apenas coloração como limitante superior tem número total de estados $2^{q+1} - 1$ nas instâncias qC_5. Seja $\mathcal{M}$ tal algoritmo. Como $q = |V(qC_5)|/5$, o número de estados para $\mathcal{M}$ no grafo $G = qC_5$ é

$$|T_{\mathcal{M}}(G)| = 2^{|V(G)|/5+1} - 1.$$

O algoritmo $\mathcal{M}$ tem custo de tempo $\Omega(1)$ para cada estado na árvore, então o custo total no pior caso é $\Omega(2^{n/5})$. □

3.3 A FAMÍLIA DE ALGORITMOS MCBBJ

Grafos junção podem ser instâncias difíceis para algoritmos da família MCBB. Como visto no Corolário 5, existem grafos junção com n vértices que obrigam qualquer algoritmo que usa apenas coloração como limitante superior a criar uma árvore com tamanho exponencial em n, o que resulta em um tempo de execução também exponencial.

Entretanto, como visto na Seção 3.1, a clique máxima de um grafo junção é a união das clique máximas dos grafos originais na junção. Ou seja, para encontrar a clique máxima em um grafo $G = G_1 + G_2$, basta encontrar as cliques máximas de G_1 e G_2 em separado e uní-las em seguida, supondo-se que é possível detectar que G é um grafo junção e decompô-lo nos grafos originais. O mesmo vale para grafos que são a junção de mais de dois grafos. Se H é o grafo junção $\sum_{i=1}^{k} H_i$, a clique máxima de H é a união das cliques máximas de cada um dos grafos H_i, com $i \in [k]$.

Em particular, a clique máxima de um grafo na família qC_5 pode ser facilmente encontrada em tempo polinomial decompondo-se o grafo em cada um dos grafos da junção e testando se cada fragmento é uma cópia de C_5.

Assim, alguns grafos junção aparentemente difíceis se tornam instâncias fáceis, desde que tratados de maneira apropriada. Porém, uma instância difícil pode ser criada

disfarçando-se um grafo junção de algum modo que o faça não ser mais um grafo junção. Por exemplo, é suficiente a inclusão de um vértice isolado em qualquer grafo para que ele não seja um grafo junção. Em geral, um grafo pode não ser um grafo junção, mas conter subgrafos que são.

Nesta seção, é apresentado um algoritmo para detecção de grafos junção e decomposição do grafo nos grafos originais com complexidade de tempo quadrática no número de vértices do grafo. Em seguida, uma atualização do Algoritmo MaxCliqueBB é apresentada, definindo uma nova família de algoritmos para solução exata do CM que analisam se o grafo induzido pelo conjunto de candidatos é um grafo junção. Por fim, um novo algoritmo desta nova família baseado no algoritmo MCQ é apresentado, juntamente com resultados experimentais.

3.3.1 Detecção e decomposição de grafos junção

Grafos junção podem facilmente ser detectados a partir do Lema 6, cuja demonstração resulta em um algoritmo imediato para decomposição de grafos junção.

Lema 6. *O complemento de um grafo junção é desconexo.*

Demonstração. Seja $G = G_1 + G_2$, então não existe em $\overline{G}$ aresta $\{v_1, v_2\}$ onde $v_1 \in V(\overline{G_1})$ e $v_2 \in V(\overline{G_2})$. Assim, $\overline{G_1}$ é um componente conexo em $\overline{G}$, ou ainda, uma coleção de componentes conexos. Caso $\overline{G_1}$ seja desconexo, G_1 também é um grafo junção. O mesmo vale para G_2. No caso geral, tem-se $G = \sum_{i=1}^{k} G_i^*$, onde $\overline{G_i^*}$, com $i \in [k]$, são os componentes conexos de $\overline{G}$. □

Pelo Lema 6, um grafo junção pode ser decomposto pela execução de qualquer algoritmo de percorrimento no complemento do grafo.

Para um melhor desempenho prático, o ideal é evitar a criação explícita do complemento do grafo, o que é possível caso o grafo seja representado por matriz de adjacências. Tanto o percorrimento em largura quanto em profundidade podem ser adaptados para percorrer o complemento do grafo simplesmente interpretando os valores como o inverso do que está guardado na matriz. Em um grafo G com n vértices, a decomposição de G com qualquer um desses algoritmos tem custo de tempo $\Theta(n^2)$.

3.3.2 Criando uma nova família de algoritmos

Decompor o grafo e executar um algoritmo da família MCBB para cada fragmento é suficiente para resolver todos os grafos qC_5 em tempo polinomial, porém, como visto acima, esses grafos podem ser disfarçados e continuar exigindo tempo exponencial de qualquer algoritmo na família MCBB que use apenas coloração no cálculo do limitante superior.

Uma forma mais abrangente de tratar tais grafos que pode, inclusive, apresentar benefícios em grafos que não foram especificamente construídos usando a operação de junção é verificar não apenas toda a entrada se é um grafo junção, mas também os subgrafos induzidos pelos conjuntos de candidatos ao longo da execução de um algoritmo de branch–and–bound. Essa ideia é a base para o meta-algoritmo MaxCliqueBBJ1. A família de algoritmos baseados em MaxCliqueBBJ1 é chamada de MCBBJ1.

Seja (Q, K) um estado em uma árvore de um algoritmo MCBB tal que o subgrafo induzido por K é um grafo junção de k grafos distintos, isto é, $G[K] = \sum_{i=1}^{k} G_i^K$. Na família MCBB os algoritmos escolhem um pivô $v \in K$ e ramificam a árvore nos filhos esquerdo e direito, onde o pivô é incluído ou não em Q. Já em um algoritmo baseado em MaxCliqueBBJ1, ao invés de continuar a busca na árvore, o algoritmo decompõe o grafo $G[K]$, busca a maior clique em cada um dos fragmentos e une as cliques ao final.

Em comparação com o Algoritmo MaxCliqueBB, o Algoritmo MaxCliqueBBJ1 adiciona análise e processamento de grafos junção em dois locais diferentes. Logo no início, nas linhas 1 a 6, o grafo inteiro é analisado. Caso G seja grafo junção, ele é decomposto e a problema é resolvido de maneira independente para cada fragmento do grafo. Já nas linhas 15 a 19, o subgrafo induzido pelo conjunto de candidatos é analisado.

Cada chamada a MaxCliqueBBJ1 dá origem a uma árvore de estados, onde cada estado é um par (Q, K) de modo análogo à árvore de estados do Algoritmo MaxCliqueBB. Como pode haver mais de uma chamada a MaxCliqueBBJ1 para uma determinada instância G, o Algoritmo MaxCliqueBBJ1 gera, assim, uma floresta de árvores estritamente binárias de estados. Porém, a utilização de uma floresta como espaço de busca dificulta o entendimento da relação entre o algoritmo e o espaço de busca, visto que as raízes ficam desconectadas no espaço de busca, tornando impossível discernir qual estado originou qual chamada recursiva. Assim, para que o espaço de estados seja uma árvore, definimos que todo estado (Q, K) onde $G[K]$ é grafo junção será denotado $[Q, K]$ e que as raízes das árvores geradas

MaxCliqueBBJ1(G)

Entrada: um grafo G.
Saída: uma clique máxima de G.

```
   ▷ Caso G seja grafo junção, chama MaxCliqueBBJ1 para cada fragmento
 1 P ← fragmentos de G
 2 Se |P| > 1 então
 3 |   C ← ∅
 4 |   Para cada X ∈ P faça
 5 |   |   C ← C ∪ MaxCliqueBBJ1(G[X])
 6 |   Devolva C
 7 Pré-processe G e inicialize C e S
 8 Enquanto S ≠ ∅ faça
 9 |   Desempilhe estado de S e guarde em (Q, K)
10 |   Processe estado(Q, K)
11 |   Enquanto K ≠ ∅ e |C| < |Q| + upper-bound(G, K, C) faça
12 |   |   Remova vértice de K e guarde em v
13 |   |   Empilhe (Q, K) em S
14 |   |   (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
   |   |   ▷ Análise de subgrafos junção - início
15 |   |   P ← fragmentos de G[K]
16 |   |   Se |P| > 1 então
17 |   |   |   Para cada X ∈ P faça
18 |   |   |   |   Q ← Q ∪ MaxCliqueBBJ1(G[X])
19 |   |   |   K ← ∅
   |   |   ▷ Análise de subgrafos junção - fim
20 |   |   Processe estado (Q, K)
21 |   Se |C| < |Q| então
22 |   |   C ← Q
23 Pós-processe G
24 Devolva C
```

pelas chamadas recursivas são filhos do estado $[Q, K]$. Além disso, caso G seja grafo junção, adicionamos um estado raiz $[\emptyset, V(G)]$ e todas as raízes das árvores das chamadas recursivas são conectadas a $[\emptyset, V(G)]$. Com isso, o espaço de busca continua sendo uma árvore, com a diferença de que possivelmente tal árvore não é binária, visto que o um grafo junção pode ter mais de dois fragmentos.

Cada algoritmo MCBB pode ser usado como base para um algoritmo na família MCBBJ1, visto que as mesmas chamadas de função estão presentes no Algoritmo MaxCliqueBB e no Algoritmo MaxCliqueBBJ1. Assim, o algoritmo que não realiza nenhum tipo de corte na árvore de estados e seleciona o pivô de forma arbitrária é chamado noboundJ1.

3.3.2.1 Subgrafos repetidos

Nos algoritmos da família MCBBJ1, é possível que o mesmo subgrafo apareça como fragmento de um grafo junção para diversos estados diferentes na árvore. Assim, tal subgrafo é resolvido diversas vezes. De fato, isso pode fazer com que o tamanho da árvore seja maior do que o da árvore para um algoritmo comparável na família MCBB.

Essa situação em que um subproblema é resolvido repetidas vezes durante a solução recursiva de um problema não é incomum, e há uma técnica de projeto de algoritmos, chamada *memoização*, específica para esses casos. A memoização, cujo nome é derivado das "funções memo" propostas em Michie (1968), consiste em guardar em um dicionário valores já calculados de uma função de modo que, caso a função seja chamada novamente com os mesmos parâmetros, basta que o valor seja recuperado do dicionário.

Adicionando-se a memoização no Algoritmo MaxCliqueBBJ1, obtém-se o Algoritmo MaxCliqueBBJ2.

No Algoritmo MaxCliqueBBJ2, o dicionário M é um vetor associativo que relaciona chaves, que são conjuntos de vértices, a valores, que também são conjuntos de vértices. As chaves são os vértices de cada fragmento X dos grafos junção referentes ao subgrafo induzido $G[K]$ que aparecerem nos estados (Q, K). O valor associado a uma chave X é a clique máxima em $G[X]$. A notação $X \notin M$ é usada para indicar que uma chave X não está no dicionário. O valor associado a uma chave X é denotado $M[X]$. A tabela M contém subconjuntos de $V(G)$ e, portanto, não pode conter mais de 2^n elementos em um grafo com n vértices. Acesso em tempo linear em n a qualquer elemento pode ser obtido usando uma árvore balanceada para guardar as chaves. O uso de tabelas de dispersão também pode ser uma alternativa com bom desempenho prático.

3.3.2.2 Considerações sobre fragmentos pequenos na junção

Seja $[Q, K]$ um estado na árvore do Algoritmo MaxCliqueBBJ2 onde o subgrafo induzido $G[K] = \sum_{i=1}^{k} G_i^K$ é um grafo junção com k fragmentos. Fragmentos "pequenos", isto é, fragmentos com poucos vértices, podem ser analisados de forma especial e algumas implicações imediatas para a solução do CM são úteis para o algoritmo.

Se há em $G[K]$ um fragmento G_i^K com apenas um vértice v, isso significa que v está conectando por arestas a todos os outros vértices de $G[K]$. Ou seja, caso sejam encontradas no Algoritmo MaxCliqueBBJ2 fragmentos com apenas um vértice, os vértices

MaxCliqueBBJ2(G)

Entrada: um grafo G.
Saída: uma clique máxima de G.

▷ `Caso` G `seja grafo junção, chama` MaxCliqueBBJ2 `para cada fragmento`
$P \leftarrow$ fragmentos de G
Se $|P| > 1$ então
  $C \leftarrow \emptyset$
  Para cada $X \in P$ faça
    $C \leftarrow C \cup$ MaxCliqueBBJ2($G[X]$)
  Devolva C
$M \leftarrow$ dicionário global para memoização
Pré-processe G e inicialize C e $\mathcal{S}$
Enquanto $\mathcal{S} \neq \emptyset$ faça
  Desempilhe estado de $\mathcal{S}$ e guarde em (Q, K)
  Processe estado (Q, K)
  Enquanto $K \neq \emptyset$ e $|C| < |Q| +$ upper-bound(G, K, C) faça
    Remova vértice de K e guarde em v
    Empilhe (Q, K) em $\mathcal{S}$
    $(Q, K) \leftarrow (Q \cup \{v\}, K \cap \Gamma_G(v))$
    ▷ `Análise de subgrafos junção - início`
    $P \leftarrow$ fragmentos de $G[K]$
    Se $|P| > 1$ então
      Para cada $X \in P$ faça
        Se $X \notin M$ então
          $M[X] \leftarrow$ MaxCliqueBBJ2($G[X]$)
        $Q \leftarrow Q \cup M[X]$
      $K \leftarrow \emptyset$
    ▷ `Análise de subgrafos junção - fim`
    Processe estado (Q, K)
  Se $|C| < |Q|$ então
    $C \leftarrow Q$
Pós-processe G
Devolva C

destes fragmentos podem ser imediatamente inseridos na clique atual Q e removidos do conjunto de candidatos K.

Fragmentos com dois vértices também podem ser resolvidas diretamente. Seja G_j^K um fragmento de $G[K]$ com dois vértices v_1 e v_2. Neste caso, v_1 e v_2 estão conectados com todos os outros vértices de G. Porém, não existe aresta $\{v_1, v_2\}$ em $G[K]$, pois caso existisse, v_1 e v_2 estariam em dois fragmentos separados de tamanho um. Em outras palavras, todo grafo que pode aparecer como fragmento com dois vértices é isomorfo ao grafo $\overline{K}_2 = (\{1, 2\}, \emptyset)$. Assim, caso o Algoritmo MaxCliqueBBJ2 encontre algum fragmento

de tamanho dois, um dos vértices pode ser inserido em Q, desde que ambos os vértices sejam removidos de K.

A partir de três vértices, podem aparecer arestas nos fragmentos de $G[K]$. Um fragmento com três vértices é um grafo isomorfo a um dos seguintes grafos: $\overline{K}_3$, $(\{1,2,3\},\{\{1,2\}\})$, $(\{1,2,3\},\{\{1,2\},\{2,3\}\})$. Em outras palavras, um fragmento com três vértices não pode ser um triângulo, isto é, um grafo isomorfo a K_3. Se o fragmento fosse um triângulo, os vértices formariam três fragmentos com um vértice em cada. Assim, o Algoritmo MaxCliqueBBJ2 pode tratar fragmentos de tamanho três realizando uma análise de casos. Os três vértices devem ser removidos de K e, caso exista alguma aresta no subgrafo, os dois vértices da aresta devem ser adicionados em Q; caso contrário, apenas um vértice deve ser adicionado a Q.

Com quatro vértices o número de casos possíveis como fragmento no grafo junção aumenta substancialmente, podendo até mesmo haver um triângulo no fragmento. Assim, a maior clique em um fragmento de quatro vértices pode ter um, dois ou três vértices.

3.3.2.3 A família de algoritmos MCBBJ

Como a análise de grafos junção realiza uma travessia no grafo, com custo de tempo quadrático no número de vértices, informações adicionais podem ser coletadas durante a travessia sem impacto assintótico e que podem resultar em ganho de desempenho. Por exemplo, em algoritmos de branch–and–bound que usam coloração como limitante superior, uma coloração mais apertada pode ser encontrada ordenando-se o conjunto de candidatos com base nos graus no subgrafo induzido antes dos vértices serem coloridos. A maioria dos algoritmos não realiza tal operação pois, apesar de o número de estados diminuir, o tempo total acaba aumentando. Contudo, durante a travessia, os graus podem ser calculados praticamente sem custo adicional de tempo, tornando essa ordenação muito mais interessante em caráter prático.

Para possibilitar esse tipo de processamento e também a atualização direta de Q e K durante e após a análise de subgrafos junção como sugerido na seção anterior, as chamadas de função que decompõem os grafos junção também devem ser parametrizadas por algoritmos concretos.

Levando tudo isso em consideração, é proposto o Algoritmo MaxCliqueBBJ. Os algoritmos gerado para definição de cada uma das funções são chamados coletivamente de

família MCBBJ. Nesta versão final do meta-algoritmo, os processamentos foram unificados, dando grande liberdade aos algoritmos.

É importante observar que o conjunto P não precisa necessariamente conter todos os fragmentos do grafo sendo analisado, visto que alguns podem ser tratados diretamente. Inclusive, a família MCBBJ admite a inclusão de algoritmos que não fazem uso da análise de grafos junção, bastando que o conjunto P seja inicializado como vazio nas linhas 1 e 14. De fato, a família MCBB está contida na família MCBBJ, onde os algoritmos MCBB são exatamente aqueles onde $P = \emptyset$ e $M = \emptyset$.

```
MaxCliqueBBJ(G)
   Entrada: um grafo G.
   Saída: uma clique máxima de G.
 1 Pré-processe G, inicialize C, S e P
 2 Se |P| > 1 então
 3 |   Para cada X ∈ P faça
 4 |   |   C ← C ∪ MaxCliqueBBJ(G[X])
 5 |   Devolva C
 6 M ← dicionário global para memoização
 7 Enquanto S ≠ ∅ faça
 8 |   Desempilhe estado de S e guarde em (Q, K)
 9 |   Processe estado (Q, K)
10 |   Enquanto K ≠ ∅ e |C| < |Q| + upper-bound(G, K, C) faça
11 |   |   Remova vértice de K e guarde em v
12 |   |   Empilhe (Q, K) em S
13 |   |   (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
14 |   |   Processe ((Q, K), P)
15 |   |   Se |P| > 1 então
16 |   |   |   Para cada X ∈ P faça
17 |   |   |   |   Se X ∉ M então
18 |   |   |   |   |   M[X] ← MaxCliqueBBJ(G[X])
19 |   |   |   |   Q ← Q ∪ M[X]
20 |   |   |   K ← ∅
21 |   Se |C| < |Q| então
22 |   |   C ← Q
23 Pós-processe G
24 Devolva C
```

3.3.3 O algoritmo MCJ

O algoritmo mais simples da família MCBBJ que faz uso da análise de grafos junção é o algoritmo que não corta nenhum estado usando limitantes e escolhe o pivô

de forma arbitrária. Ou seja, tal algoritmo é equivalente a adicionar a decomposição do grafo no algoritmo nobound. Um algoritmo similar ao algoritmo Basic é obtido pela adição de uma ordem no conjunto K e do limitante trivial, isto é, fazendo com que a função upper-bound(G, K, C) devolva o tamanho de K. Assim, todo algoritmo da família MCBB pode ser utilizado como base para a criação de algoritmos na família MCBBJ. Como prova de conceito, foi desenvolvido um algoritmo como uma variação do algoritmo MCQ. Este novo algoritmo recebeu o nome de algoritmo MCJ.

O algoritmo MCJ utiliza a decomposição de grafos e trata diretamente fragmentos de tamanho menor ou igual a quatro. O algoritmo MCJ utiliza a coloração gulosa do Algoritmo Guloso da mesma forma como o algoritmo MCQ, isto é, a coloração é utilizada como limitante superior e na escolha do pivô, que é sempre um vértice da maior cor. O algoritmo MCJ reordena o conjunto K antes de computar a coloração em ordem não crescente de graus em $G[K]$.

3.3.4 Experimentos

O algoritmo MCJ foi implementado em linguagem C++, como uma modificação da implementação original do algoritmo MaxCliqueDyn (KONC; JANEŽIČ, 2007a)[5].

A decisão de implementar o algoritmo desta forma foi tomada por dois motivos principais. Em primeiro lugar, o uso de uma linguagem compilada viabiliza a execução de testes em condições parecidas com as apresentadas em grande parte dos algoritmos disponíveis na literatura, utilizando as instâncias do DIMACS Second Implementation Challenge (JOHNSON; TRICK, 1996). Em segundo lugar, a adoção de uma implementação de um algoritmo de terceiros permite que este algoritmo seja também executado e testado em condições iguais às do algoritmo proposto. Com isso, os dados podem ser comparados de forma direta entre estes dois algoritmos.

Os testes foram realizados em um computador com processador Intel Xeon E5-1650, com seis núcleos a 3.5 GHz, e 32 GB de memória, rodando a distribuição Ubuntu 16.04 do sistema operacional GNU/Linux. A execução foi automatizada utilizando o programa GNU Parallel (TANGE, 2011).

Além destes testes, outros experimentos utilizando grafos aleatórios foram realizados dentro da metodologia para comparação experimental introduzida no Capítulo 6. Os

[5] disponível em <http://insilab.org/maxclique/>.

Tabela 1 – Resultados experimentais de MCJ para grafos qC_5.

		MaxCliqueDyn		MCJ	
q	n	$\lvert T_{\mathsf{MCDyn}}(qC_5)\rvert$	Tempo (s)	$\lvert T_{\mathsf{MCJ}}(qC_5)\rvert$	Tempo (s)
1	5	9	0.0003	7	0.0005
2	10	27	0.0007	8	0.0007
3	15	169	0.0042	13	0.0010
4	20	619	0.0197	18	0.0015
5	25	4773	0.1475	23	0.0021
6	30	19 701	0.8146	28	0.0035
7	35	122 827	5.2984	33	0.0044
8	40	752 905	37.3874	38	0.0068
9	45	3 213 301	206.3587	43	0.0065

resultados destes testes adicionais são encontrados na Seção 6.4.

3.3.5 Resultados experimentais

Como a família MCBBJ foi desenvolvida especificamente para tratar de grafos junção, os primeiros testes foram realizados utilizando instâncias da família qC_5. A Tabela 1 mostra os resultados da execução dos algoritmos MaxCliqueDyn e MCJ para os grafos com q variando de um a nove, isto é, grafos com até 45 vértices. As colunas $|T_{\mathsf{MCDyn}}(qC_5)|$ e $|T_{\mathsf{MCJ}}(qC_5)|$ mostram o número de estados criados pelos algoritmos MaxCliqueDyn e MCJ, respectivamente. Também são apresentados os tempos de execução.

Como era de se esperar, as árvores criadas pelo algoritmo MaxCliqueDyn crescem de forma exponencial em relação ao número de vértices nos grafos qC_5, enquanto as árvores do algoritmo MCJ têm crescimento linear.

O algoritmo MCJ também foi testado com o conjunto de instâncias DIMACS. Para estes experimentos, foi estabelecido um limite de tempo de execução de uma hora.

As Tabelas 2 e 3 mostram os resultados para os algoritmos MCJ e MaxCliqueDyn. As colunas $|T_{\mathsf{MCDyn}}(G)|$ e $|T_{\mathsf{MCJ}}(G)|$ mostram o número de estados criados pelos algoritmos MaxCliqueDyn e MCJ, respectivamente. Também são apresentados os tempos de execução. Colunas adicionais apresentam a razão entre o número de estados dos algoritmos MCJ e MaxCliqueDyn e entre os tempos dos algoritmos.

Das 66 instâncias testadas, 52 foram resolvidas pelos dois algoritmos no tempo permitido. O algoritmo MaxCliqueDyn resolveu quatro instâncias a mais que o algoritmo MCJ. Considerando as instâncias resolvidas, o algoritmo MCJ criou menos estados que o

Tabela 2 – Resultados experimentais de MCJ para grafos DIMACS.

G	MaxCliqueDyn $\lvert T_{\mathsf{MCDyn}}(G)\rvert$	MaxCliqueDyn Tempo (s)	MCJ $\lvert T_{\mathsf{MCJ}}(G)\rvert$	MCJ Tempo (s)	Razão (MCJ/MCDyn) $\lvert T(G)\rvert$	Razão (MCJ/MCDyn) Tempo
MANN_a9	99	0.0002	69	0.0009	0.6970	4.0140
MANN_a27	48 445	1.4732	47 520	7.1826	0.9809	4.8755
MANN_a45	4 590 866	956.6900	$\geq$ 3 062 098	–		
MANN_a81	$\geq$ 356 698	–	$\geq$ 145 587	–		
brock200_1	235 172	0.4650	201 743	2.2066	0.8579	4.7459
brock200_2	3622	0.0082	3115	0.0298	0.8600	3.6367
brock200_3	12 706	0.0267	10 733	0.1202	0.8447	4.5025
brock200_4	46 755	0.0837	45 965	0.4212	0.9831	5.0338
brock400_1	118 164 924	322.4660	88 017 132	1403.3000	0.7449	4.3518
brock400_2	45 097 178	145.1470	54 184 598	926.8230	1.2015	6.3854
brock400_3	113 099 318	280.2240	49 363 435	753.4470	0.4365	2.6887
brock400_4	53 805 125	150.1040	14 198 686	287.2140	0.2639	1.9134
brock800_1	$\geq$1 207 676 781	–	$\geq$ 215 379 881	–		
brock800_2	$\geq$1 167 198 740	–	$\geq$ 201 711 823	–		
brock800_3	824 058 952	2809.5300	$\geq$ 255 358 909	–		
brock800_4	501 919 040	1964.6700	$\geq$ 205 371 133	–		
c-fat200-1	212	0.0003	8	0.0003	0.0377	1.1225
c-fat200-2	238	0.0004	12	0.0004	0.0504	1.1830
c-fat200-5	307	0.0009	30	0.0013	0.0977	1.3907
c-fat500-1	513	0.0012	11	0.0014	0.0214	1.1370
c-fat500-2	539	0.0013	24	0.0016	0.0445	1.2747
c-fat500-5	618	0.0025	41	0.0030	0.0663	1.2295
c-fat500-10	743	0.0057	2	0.0011	0.0027	0.1895
hamming6-2	62	0.0003	10	0.0003	0.1613	1.0891
hamming6-4	123	0.0002	114	0.0006	0.9268	2.9468
hamming8-2	371	0.0058	20	0.0026	0.0539	0.4408
hamming8-4	12 704	0.0347	11 625	0.2032	0.9151	5.8619
hamming10-2	2863	0.8473	21 906	25.3342	7.6514	29.9006
hamming10-4	$\geq$1 221 040 225	–	$\geq$ 153 987 448	–		

algoritmo MaxCliqueDyn em 45 grafos. O tempo de execução do algoritmo MaxCliqueDyn foi melhor na maioria dos casos, mas ambos os algoritmos tiveram tempos na mesma ordem de magnitude na grande maioria dos testes. Apenas em quatro grafos, o algoritmo MCJ foi mais de dez vezes mais lento que o algoritmo MaxCliqueDyn, em instâncias onde a árvore de estados de MCJ também é consideravelmente maior que a de MaxCliqueDyn.

Tabela 3 – Resultados experimentais de MCJ para grafos DIMACS.

G	MaxCliqueDyn $\lvert T_{\mathsf{MCDyn}}(G)\rvert$	MaxCliqueDyn Tempo (s)	MCJ $\lvert T_{\mathsf{MCJ}}(G)\rvert$	MCJ Tempo (s)	Razão (MCJ/MCDyn) $\lvert T(G)\rvert$	Razão (MCJ/MCDyn) Tempo
johnson8-2-4	50	0.0001	41	0.0002	0.8200	2.6438
johnson8-4-4	216	0.0007	138	0.0022	0.6389	3.2281
johnson16-2-4	583 334	0.2634	525 524	0.9264	0.9009	3.5167
johnson32-2-4	$\geq$2 926 171 714	–	$\geq$1 897 197 389	–		
keller4	7711	0.0133	8785	0.0746	1.1393	5.6110
keller5	$\geq$ 740 419 902	–	$\geq$ 87 115 159	–		
keller6	$\geq$ 191 612 486	–	$\geq$ 24 622 893	–		
p_hat300-1	2129	0.0036	1507	0.0119	0.7078	3.2816
p_hat300-2	7459	0.0187	4509	0.1081	0.6045	5.7798
p_hat300-3	629 203	2.1357	440 602	10.2073	0.7003	4.7794
p_hat500-1	10 956	0.0167	8606	0.0730	0.7855	4.3787
p_hat500-2	193 480	0.7224	108 709	4.0831	0.5619	5.6518
p_hat500-3	25 535 382	153.0120	23 115 349	968.1650	0.9052	6.3274
p_hat700-1	28 084	0.0584	19 243	0.2347	0.6852	4.0170
p_hat700-2	1 091 257	5.8263	701 138	37.4351	0.6425	6.4252
p_hat700-3	294 050 955	2593.5200	$\geq$ 59 512 978	–		
p_hat1000-1	170 075	0.2999	146 247	1.3366	0.8599	4.4571
p_hat1000-2	31 172 704	189.4430	30 089 184	1652.2200	0.9652	8.7215
p_hat1000-3	$\geq$ 417 607 264	–	$\geq$ 72 216 681	–		
p_hat1500-1	1 138 690	2.4950	1 046 978	11.9243	0.9195	4.7794
p_hat1500-2	$\geq$ 341 978 397	–	$\geq$ 53 214 816	–		
p_hat1500-3	$\geq$ 359 692 207	–	$\geq$ 53 378 602	–		
san200_0.7_1	1138	0.0046	69	0.0038	0.0606	0.8337
san200_0.7_2	2521	0.0071	1042	0.0153	0.4133	2.1607
san200_0.9_1	11 031	0.0437	692	0.0237	0.0627	0.5432
san200_0.9_2	49 351	0.2523	45 081	1.2345	0.9135	4.8935
san200_0.9_3	332 964	1.5104	1 867 802	37.3570	5.6096	24.7332
san400_0.5_1	2761	0.0073	436	0.0049	0.1579	0.6771
san400_0.7_1	42 579	0.2021	27 192	0.5776	0.6386	2.8580
san400_0.7_2	9182	0.0707	327 177	4.6200	35.6324	65.3293
san400_0.7_3	510 097	1.1602	957 203	10.9343	1.8765	9.4245
san400_0.9_1	628 188	11.3414	23 986 801	1143.7900	38.1841	100.8509
san1000	124 031	0.2417	34 936	0.3831	0.2817	1.5853
sanr200_0.7	106 744	0.1900	84 530	0.8402	0.7919	4.4226
sanr200_0.9	6 361 069	22.7824	4 588 398	95.9154	0.7213	4.2101
sanr400_0.5	242 343	0.4254	216 715	2.0223	0.8942	4.7540
sanr400_0.7	37 811 004	85.2271	33 045 118	405.4630	0.8740	4.7574

4 ANÁLISE DOS ALGORITMOS

Neste capítulo são apresentados resultados obtidos analiticamente relacionados ao CM, voltados ao comportamento esperado em grafos aleatórios do modelo $\mathcal{G}_{n,p}$ e à execução de alguns algoritmos de branch–and–bound para solução do problema.

O tempo de execução dos algoritmos MCBB é dominado pelo custo de processamento de cada estado. Para um algoritmo $\mathcal{A}$ da família MCBB, seja $t_{\mathcal{A}}(G)$ o custo de tempo de $\mathcal{A}$ para o grafo G, e sejam $f_{\mathcal{A}}(G, Q, K)$ o tempo gasto para processar o estado (Q, K) e $g_{\mathcal{A}}(G)$ o tempo gasto fora do laço da linha 2. Então, simplificadamente,

$$t_{\mathcal{A}}(G) = g_{\mathcal{A}}(G) + \sum_{(Q,K)\in T_{\mathcal{A}}(G)} f_{\mathcal{A}}(G, Q, K).$$

Em todos os algoritmos estudados, as funções $f_{\mathcal{A}}$ e $g_{\mathcal{A}}$ são polinomiais no tamanho de G. Seja $f^*_{\mathcal{A}}(G)$ o maior tempo gasto para processar algum estado do espaço de busca, isto é,

$$f^*_{\mathcal{A}}(G) := \max\{f_{\mathcal{A}}(G, Q, K) \mid (Q, K) \in T_{\mathcal{A}}(G)\}, \tag{4.1}$$

então

$$t_{\mathcal{A}}(G) \leq g_{\mathcal{A}}(G) + \sum_{(Q,K)\in T_{\mathcal{A}}(G)} f^*_{\mathcal{A}}(G) \tag{4.2}$$

$$= g_{\mathcal{A}}(G) + |T_{\mathcal{A}}(G)| f^*_{\mathcal{A}}(G). \tag{4.3}$$

Assim, o tamanho da árvore de estados é o principal contribuinte para a complexidade de tempo dos algoritmos MCBB, e é foco da análise dos algoritmos fazer previsões em relação ao número de estados gerados.

Como o algoritmo nobound não reduz de nenhuma forma o espaço de busca, o número de estados gerados por nobound representa uma cota superior em relação ao número de estados criados dentre os algoritmos MCBB. Levando isso em consideração, o algoritmo nobound é analisado em detalhe na Seção 4.1. Em seguida, o impacto da adição de um limitante superior trivial é estudado na Seção 4.2, pela análise do algoritmo Basic. Por fim, o uso de limitantes baseados em coloração é analisado na Seção 4.3.

4.1 ANÁLISE DO ALGORITMO NOBOUND

O algoritmo nobound enumera todas as cliques do grafo de entrada e é, assim, o algoritmo mais simples dentre os algoritmos MCBB.

Considerando o espaço de busca, cada clique aparece no conjunto Q uma vez em uma folha, isto é, em um estado onde o conjunto K é vazio. Cada clique pode aparecer diversas vezes em nós internos do espaço de busca, mas nunca repetida em folhas, visto que os vértices escolhidos como pivô não são devolvidos a K. Estes fatos são estabelecidos pelo Teorema 7 e pelo Corolário 10, que é suportado pelos Lemas 8 e 9. Com isso, o número total de estados em uma execução de nobound está linearmente relacionado com o número de cliques no grafo, como mostra o Teorema 11. O tempo de execução do algoritmo nobound para um grafo G com $|\mathcal{C}(G)|$ cliques é $O(n^2|\mathcal{C}(G)|)$ e $\Omega(|\mathcal{C}(G)|)$, pelos Corolários 12 e 13.

Seguem algumas definições necessárias para a demonstração do Teorema 7. Dados uma clique X em G e $K \subseteq \Gamma_G^{\cap}(X)$, o conjunto de cliques em G que contém todos os vértices de X é denotado $\mathcal{C}(G, X, K)$, isto é,

$$\mathcal{C}(G, X, K) := \{C \subseteq V(G) \mid C \in \mathcal{C}(G) \text{ e } X \subseteq C \subseteq X \cup K\}.$$

Dado um estado (X, K) em $T_{\mathsf{nobound}}(G)$, a subárvore de $T_{\mathsf{nobound}}(G)$ enraizada em (X, K) é denotada $T_{\mathsf{nobound}}(G, X, K)$. Além disso, o conjunto de todas as cliques enumeradas em $T_{\mathsf{nobound}}(G, X, K)$ é denotado $\mathcal{T}(G, X, K)$, isto é,

$$\mathcal{T}(G, X, K) := \bigcup_{(Q,K) \in T_{\mathsf{nobound}}(G,X,K)} \{Q\}.$$

Teorema 7. *Dado um grafo G, cada clique em $\mathcal{C}(G)$ é igual ao conjunto Q de pelo menos um estado na árvore de estados $T_{\mathsf{nobound}}(G)$.*

Demonstração. Provar que cada clique em $\mathcal{C}(G)$ é igual ao conjunto Q de pelo menos um estado em $T_{\mathsf{nobound}}(G)$ é o mesmo que provar que $\mathcal{T}(G, \emptyset, V(G)) = \mathcal{C}(G, \emptyset, V(G))$. Vamos fazer isto provando que $\mathcal{T}(G, Q, K) = \mathcal{C}(G, Q, K)$ por indução em $|K|$.

Base: Se $|K| = 0$ então $K = \emptyset$ e, neste caso, temos

$$\mathcal{T}(G, Q, K) = \{Q\},$$
$$\mathcal{C}(G, Q, K) = \{Q\}.$$

Hipótese de Indução: Seja $k \in \mathbb{N}$ tal que se $|K| \leq k$, então

$$\mathcal{T}(G, Q, K) = \mathcal{C}(G, Q, K).$$

Passo da Indução: Vamos provar que se $|K| = k + 1$ então

$$\mathcal{T}(G, Q, K) = \mathcal{C}(G, Q, K).$$

Como $K \neq \emptyset$, então para algum $v \in K$ temos

$$\mathcal{T}(G, Q, K) = \{Q\} \cup \mathcal{T}(G, Q \cup \{v\}, K \cap \Gamma_G(v)) \cup \mathcal{T}(G, Q, K - \{v\}).$$

Da Hipótese de Indução temos que

$$\mathcal{T}(G, Q \cup \{v\}, K \cap \Gamma_G(v)) = \mathcal{C}(G, Q \cup \{v\}, K \cap \Gamma_G(v)),$$
$$\mathcal{T}(G, Q, K - \{v\}) = \mathcal{C}(G, Q, K - \{v\}),$$

de forma que basta provar que

$$\mathcal{C}(G, Q, K) = \{Q\} \cup \mathcal{C}(G, Q \cup \{v\}, K \cap \Gamma_G(v)) \cup \mathcal{C}(G, Q, K - \{v\}).$$

É evidente que

$$\mathcal{C}(G, Q, K) \supseteq \{Q\} \cup \mathcal{C}(G, Q \cup \{v\}, K \cap \Gamma_G(v)) \cup \mathcal{C}(G, Q, K - \{v\}),$$

de forma que basta provar que

$$\mathcal{C}(G, Q, K) \subseteq \{Q\} \cup \mathcal{C}(G, Q \cup \{v\}, K \cap \Gamma_G(v)) \cup \mathcal{C}(G, Q, K - \{v\}).$$

Seja então $C \in \mathcal{C}(G, Q, K)$, ou seja,

$$Q \subseteq C \subseteq Q \cup K.$$

Se $v \in C$, então é imediato que $Q \cup \{v\} \subseteq C$. Pelo mesmo motivo, também é imediato que $C \subseteq Q \cup (K \cap \Gamma_G(v))$. Concluímos então que $C \in \mathcal{C}(G, Q \cup \{v\}, K \cap \Gamma_G(v))$.

Finalmente, se $v \notin C$ temos que, por um lado, $Q \subseteq C$ e por outro lado, que $(Q \cup K) - \{v\} = Q \cup (K - \{v\})$ e, portanto, $C \in \mathcal{C}(G, Q, K - \{v\})$.

□

Lema 8. *Sejam G um grafo e $E_1 = (Q_1, K_1)$ um estado em $T_{\mathsf{nobound}}(G)$, então não existe outro estado $E_2 = (Q_2, K_2)$ em $T_{\mathsf{nobound}}(G)$ onde $Q_1 = Q_2$ e $K_1 = K_2$.*

Demonstração. Por contradição, supondo que $E_1 = (Q_1, K_1)$ e $E_2 = (Q_2, K_2)$ são estados distintos em $T_{\mathsf{nobound}}(G)$ tais que $Q_1 = Q_2$ e $K_1 = K_2$.

Em primeiro lugar, E_1 não pode ser antecedente de E_2, pois o pivô de E_1 não estaria em K_2 e, consequentemente, $K_1 \neq K_2$. Pelo mesmo motivo, E_2 não pode ser antecedente de E_1. Assim, E_1 e E_2 possuem ao menos um antecedente comum em $T_{\mathsf{nobound}}(G)$.

Seja $E_A = (Q_A, K_A)$ o antecedente comum de E_1 e E_2 tal que E_1 e E_2 estão em subárvores distintas, um na subárvore enraizada em $E(Q_A, K_A)$ e outro na subárvore enraizada em $D(Q_A, K_A)$, e seja v o pivô em E_A. Sem perda de generalidade, suponha que E_1 está na subárvore enraizada em $E(Q_A, K_A)$ e E_2 está na subárvore enraizada em $D(Q_A, K_A)$. Em todo estado na subárvore enraizada em $E(Q_A, K_A)$, temos $v \in Q$, e em todo estado na subárvore enraizada em $D(Q_A, K_A)$, temos $v \notin Q$. Assim, $v \in Q_1$ e $v \notin Q_2$, o que contradiz a suposição de que $Q_1 = Q_2$. □

Lema 9. *Sejam G um grafo e $E = (Q, K)$ um estado em $T_{\mathsf{nobound}}(G)$, então existe um único estado folha $(Q, \emptyset)$ em $T_{\mathsf{nobound}}(G)$.*

Demonstração. O descendente mais à direita de E é a folha $D^*(E) = (Q, \emptyset)$ e, pelo Lema 8, é única. □

Corolário 10. *Seja G um grafo, então cada clique C em $\mathcal{C}(G)$ é igual ao conjunto Q de exatamente uma folha $(Q, \emptyset)$ em $T_{\mathsf{nobound}}(G)$.*

Demonstração. Toda clique $C \in \mathcal{C}(G)$ aparece em um estado (C, K) em $T_{\mathsf{nobound}}(G)$ pelo Teorema 7 e, pelo Lema 9, existe exatamente uma folha $(C, \emptyset)$ em $T_{\mathsf{nobound}}(G)$. □

Teorema 11. *Sejam G um grafo, então*

$$|T_{\mathsf{nobound}}(G)| = 2|\mathcal{C}(G)| - 1.$$

Demonstração. Cada clique em $\mathcal{C}(G)$ aparece em uma folha em $T_{\mathsf{nobound}}(G)$, pelo Corolário 10. Assim, a árvore $T_{\mathsf{nobound}}(G)$ tem $|\mathcal{C}(G)|$ folhas. Como se trata de uma árvore estritamente binária, $T_{\mathsf{nobound}}(G)$ tem $|\mathcal{C}(G)| - 1$ nós internos. Totalizando, o tamanho da árvore de estados é

$$|T_{\mathsf{nobound}}(G)| = |\mathcal{C}(G)| + |\mathcal{C}(G)| - 1 = 2|\mathcal{C}(G)| - 1.$$

□

Corolário 12. *O tempo de execução do algoritmo* nobound *para um grafo G com n vértices é $O(n^2|\mathcal{C}(G)|)$.*

Demonstração. Pela equação (4.1), o tempo de nobound para G é

$$t_{\mathsf{nobound}}(G) = g(G) + \sum_{(Q,K)\in T_{\mathsf{nobound}}(G)} f(G,Q,K),$$

onde $f(G,Q,K)$ é o tempo gasto para processar o estado (Q,K) e $g(G)$ o tempo gasto fora do laço da linha 3.

A função $g(G)$ é $O(n)$, visto que todos os vértices devem ser inclusos no conjunto de candidatos do único estado inicial da pilha.

A operação mais custosa para o estado (Q,K) é a interseção $K \cap \Gamma_G(v)$ na linha 8, que pode ser implementada com custo de tempo $O(|K| \times |\Gamma_G(v)|)$. Como $K \subseteq V(G)$ e $\Gamma_G(v) \subseteq V(G)$, o tempo $t_{\mathsf{nobound}}(G)$ é

$$t_{\mathsf{nobound}}(G) = O(n) + \sum_{(Q,K)\in T_{\mathsf{nobound}}(G)} O(n^2).$$

Considerando o tamanho da árvore $T_{\mathsf{nobound}}(G)$ pelo Teorema 11, obtém-se

$$t_{\mathsf{nobound}}(G) = O(n) + (2|\mathcal{C}(G)| - 1)O(n^2).$$

Portanto,

$$t_{\mathsf{nobound}}(G) = O(n^2|\mathcal{C}(G)|).$$

□

Corolário 13. *O tempo de execução do algoritmo* nobound *para um grafo G com n vértices é $\Omega(|\mathcal{C}(G)|)$.*

Demonstração. Como no Corolário 12, o tempo de nobound para G é

$$t_{\mathsf{nobound}}(G) = g(G) + \sum_{(Q,K)\in T_{\mathsf{nobound}}(G)} f(G,Q,K).$$

A função $g(G)$ é $\Omega(1)$ e, para todo estado (Q,K), a função $f(G,Q,K)$ é $\Omega(1)$. Assim, o tempo $t_{\mathsf{nobound}}(G)$ é

$$t_{\mathsf{nobound}}(G) = \Omega(1) + \sum_{(Q,K)\in T_{\mathsf{nobound}}(G)} \Omega(1).$$

Considerando o tamanho da árvore $T_{\mathsf{nobound}}(G)$ pelo Teorema 11, obtém-se

$$t_{\mathsf{nobound}}(G) = \Omega(1) + (2|\mathcal{C}(G)| - 1)\Omega(1).$$

Assim,

$$t_{\mathsf{nobound}}(G) = |\mathcal{C}(G)|\Omega(1)$$

e, portanto,

$$t_{\mathsf{nobound}}(G) = \Omega(|\mathcal{C}(G)|).$$

□

Pelo Teorema 11, o número de estados para nobound é exponencial, subexponencial ou polinomial conforme o número de cliques no grafo. Consequentemente, o mesmo ocorre com o tempo de execução, pelos Corolários 12 e 13. Nas próximas seções, o tamanho da árvore de nobound é estudada para algumas famílias de grafos.

Assim, é importante observar que o algoritmo nobound gera a maior árvore de estados dentre todos os algoritmos MCBB, como mostra o Lema 14. Desta forma, todos os outros algoritmos MCBB criam no máximo o mesmo número de estados que nobound para um determinado grafo, mas é esperado que criem menos.

Lema 14. *Seja $\mathcal{A}$ um algoritmo* MCBB*. Para todo grafo G,*

$$|T_{\mathcal{A}}(G)| \leq |T_{\mathsf{nobound}}(G)|.$$

Demonstração. Sejam $\mathcal{A}$ um algoritmo MCBB e G um grafo. Se o algoritmo $\mathcal{A}$ não aplica nenhum limitante superior, ele enumera todas as cliques em G. Neste caso, pelo Corolário 10, todas as cliques de G aparecem exatamente uma vez como folhas em $T_{\mathcal{A}}(G)$ e, portanto, $|T_{\mathcal{A}}(G)| = |T_{\mathsf{nobound}}(G)|$. Caso contrário, o algoritmo $\mathcal{A}$ aplica alguma estratégia para reduzir o espaço de busca, resultando em $|T_{\mathcal{A}}(G)| \leq |T_{\mathsf{nobound}}(G)|$. □

4.1.1 Pior caso – o grafo completo

O tamanho da árvore de estados de nobound depende unicamente da quantidade de cliques no grafo de entrada, pelo Teorema 11. Portanto, o grafo que gera a maior árvore é o grafo com o maior número de cliques. Como as cliques são subconjuntos do conjunto de vértices, o grafo com o número máximo de cliques é um grafo onde todo subconjunto de vértices é uma clique. O grafo com n vértices que atende tal propriedade é o grafo completo K_n, que possui 2^n cliques, visto que todo subconjunto de $V(K_n)$ é uma clique.

Assim, o pior caso para nobound é determinado no Teorema 15.

Teorema 15. *O tamanho da árvore de estados na execução de* nobound *para o grafo completo com n vértices é $2^{n+1} - 1$, isto é,*

$$|T_{\text{nobound}}(K_n)| = 2^{n+1} - 1,$$

e esse é o pior caso para nobound*.*

Demonstração. Como todo subconjunto de $V(K_n)$ é uma clique, o conjunto $\mathcal{C}(K_n)$ de cliques em K_n tem tamanho 2^n. Assim, pelo Teorema 11,

$$|T_{\text{nobound}}(K_n)| = 2 \times 2^n - 1 = 2^{n+1} - 1.$$

Em geral, um grafo G qualquer pode ter subconjuntos de $V(G)$ que não são cliques, então K_n é o pior caso para nobound e

$$|T_{\text{nobound}}(G)| \leq 2^{n+1} - 1.$$

□

O Teorema 15, aliado ao Corolário 12, mostra que o algoritmo nobound tem tempo exponencial no pior caso.

4.1.2 Grafos de Moon–Moser

Um outro caso onde nobound tem número de estados exponencial é o dos grafos de Moon–Moser, denotados $M(n)$, onde n é o número de vértices. Esses grafos são particularmente relevantes por apresentarem o maior número possível de cliques maximais para cada n (MOON; MOSER, 1965). A forma de gerar um grafo na família $M(n)$ depende do resto da divisão de n por três.

O grafo $M(n)$ para n múltiplo de três é o grafo junção de $n/3$ grafos $\overline{K}_3$, isto é, grafos onde cada um é composto de três vértices e nenhuma aresta. Ou seja,

$$M(n) = \tfrac{n}{3}\overline{K}_3, \text{ se } n \equiv 0 \pmod 3.$$

Naturalmente, o conjunto de vértices de cada grafo na junção é independente em $M(n)$. Assim, existem $3^{n/3}$ cliques maximais de tamanho $n/3$ em $M(n)$, que correspondem a escolher um vértice de cada conjunto independente. O número de cliques é ainda maior, pois os subconjuntos das cliques maximais também devem ser contados.

O grafo $\overline{K}_3$ possui quatro cliques: a clique vazia e três cliques de tamanho um. Assim, pelo Teorema 1, o número de cliques em $M(n)$ é

$$|\mathcal{C}(M(n))| = 4^{n/3}, \text{ se } n \equiv 0 \pmod 3.$$

Esse valor pode ser entendido de forma intuitiva visualizando-se que existem $n/3$ conjuntos independentes de tamanho três em $M(n)$ e cada clique corresponde a uma de quatro escolhas para cada conjunto independente: incluir um dos vértices ou nenhum.

Assim, pelo Teorema 11, para n múltiplo de três,

$$|T_{\mathsf{nobound}}(M(n))| = 2|\mathcal{C}(M(n))| - 1 = 2 \times 4^{n/3} - 1 = 2^{\left(\frac{2n}{3}+1\right)} - 1.$$

Quando n não é múltiplo de três, a construção do grafo $M(n)$ é adequada de forma bastante simples. Se $n \equiv 2 \pmod 3$, um dos grafos na junção possui dois vértices e nenhuma aresta. Assim, o grafo $M(n)$ nesse caso tem $(n-2)/3$ conjuntos independentes de tamanho três e um conjunto independente de tamanho dois, isto é,

$$M(n) = \overline{K}_2 + \tfrac{n-2}{3}\overline{K}_3, \text{ se } n \equiv 2 \pmod 3.$$

Se $n \equiv 1 \pmod 3$, dois grafos na junção têm dois vértices e nenhuma aresta, ou seja,

$$M(n) = 2\overline{K}_2 + \tfrac{n-4}{3}\overline{K}_3, \text{ se } n \equiv 1 \pmod 3.$$

O caso $n = 1$ é um caso especial e é constituído de um único vértice, isto é,

$$M(1) = K_1.$$

Pelo Teorema 1, o número de cliques nos grafos de Moon–Moser é dado por

$$|\mathcal{C}(M(n))| = \begin{cases} 2, & \text{se } n = 1, \\ 4^{n/3}, & \text{se } n \equiv 0 \pmod 3, \\ 9 \times 4^{\left(\frac{n-4}{3}\right)}, & \text{se } n \equiv 1 \pmod 3, \\ 3 \times 4^{\left(\frac{n-2}{3}\right)}, & \text{se } n \equiv 2 \pmod 3. \end{cases}$$

Consequentemente, pelo Teorema 11, o número de estados gerados na execução do algoritmo nobound é

$$|T_{\mathsf{nobound}}(M(n))| = \begin{cases} 3, & \text{se } n = 1, \\ 2^{\left(\frac{2n}{3}+1\right)} - 1, & \text{se } n \equiv 0 \pmod 3, \\ 9 \times 2^{\left(\frac{2n-5}{3}\right)} - 1, & \text{se } n \equiv 1 \pmod 3, \\ 3 \times 2^{\left(\frac{2n-1}{3}\right)} - 1, & \text{se } n \equiv 2 \pmod 3. \end{cases}$$

4.1.3 Grafos do modelo $\mathcal{G}_{n,p}$

Para os grafos completos e de Moon–Moser, é possível se obter o número exato de estados na execução do algoritmo nobound, pois o número de cliques existentes no grafo depende apenas do número de vértices. No modelo $\mathcal{G}_{n,p}$, o número de cliques é uma variável aleatória. Em um extremo, o grafo com n vértices pode não ter nenhuma aresta, e portanto possuir $n+1$ cliques. No outro extremo está o grafo completo com n vértices, que possui 2^n cliques.

Assim, pode-se analisar o comportamento esperado de nobound para grafos $\mathcal{G}_{n,p}$. Uma série de resultados para o modelo $\mathcal{G}_{n,p}$ é apresentada nesta seção e ao final o tamanho esperado da árvore de estados é calculado.

Lema 16. *Dados um grafo G no modelo $\mathcal{G}_{n,p}$ e um conjunto de vértices $X \subseteq V(G)$, a probabilidade de X ser clique em G é*

$$\mathbb{P}\left(X \text{ é clique em } G\right) = p^{\binom{|X|}{2}}.$$

Demonstração. Seja G um grafo no modelo $\mathcal{G}_{n,p}$. Em G, a probabilidade de cada par $\{v,u\} \in \binom{X}{2}$ ser uma aresta é igual a p, independente da existência de outras arestas. Assim, a probabilidade de todas as arestas possíveis em $G[X]$ existirem é $p^{\binom{|X|}{2}}$, ou seja, depende apenas do número de vértices em X. □

Lema 17. *Dado um grafo G no modelo $\mathcal{G}_{n,p}$, o número esperado de cliques de tamanho r em G é*

$$\mathbb{E}\left[N_r\right] = \binom{n}{r} p^{\binom{r}{2}}.$$

Demonstração. Seja G um grafo no modelo $\mathcal{G}_{n,p}$. Para cada $X \in \binom{V(G)}{r}$, seja C_X a variável indicadora que indica se X é clique em G. Então, pelo Lema 16,

$$\mathbb{P}\left(C_X = 1\right) = p^{\binom{|X|}{2}},$$

e o número de cliques de tamanho r é dado pela variável aleatória

$$N_r = \sum_{X \in \binom{V(G)}{r}} C_X,$$

e consequentemente

$$\begin{aligned}\mathbb{E}[N_r] &= \mathbb{E}\left[\sum_{X\in\binom{V(G)}{r}} C_X\right] = \sum_{X\in\binom{V(G)}{r}} \mathbb{E}[C_X] \\ &= \sum_{X\in\binom{V(G)}{r}} p^{\binom{|X|}{2}} = \sum_{X\in\binom{V(G)}{r}} p^{\binom{r}{2}} \\ &= \binom{n}{r} p^{\binom{r}{2}}.\end{aligned}$$

□

Teorema 18. *O número esperado de cliques em um grafo no modelo $\mathcal{G}_{n,p}$ é*

$$\mathbb{E}[N] = \sum_{r=0}^{n} \binom{n}{r} p^{\binom{r}{2}}.$$

Demonstração. O teorema segue direto da aplicação do Lema 17 para cada tamanho possível de clique. □

Corolário 19. *Sejam $n \geq 2$ um inteiro, $0 < p < 1$ uma constante e $c_p := 1/\lg(1/p)$. O número esperado de cliques em $\mathcal{G}_{n,p}$ é limitado por $n^{2+c_p \lg n}$.*

Demonstração. Para $n = 2$, temos do Teorema 18 que

$$\mathbb{E}[N] = \binom{2}{0} p^{\binom{0}{2}} + \binom{2}{1} p^{\binom{1}{2}} + \binom{2}{2} p^{\binom{2}{2}} = 1 + 2 + p \leq n^{2+\log_{1/p} n} = n^{2+c_p \lg n}.$$

Para $n \geq 3$, temos do Teorema 18 que

$$\mathbb{E}[N] = \sum_{r=0}^{n} \binom{n}{r} p^{\binom{r}{2}} \leq \binom{n}{0} p^{\binom{0}{2}} + (n-1)\binom{n}{r^*} p^{\binom{r^*}{2}} + \binom{n}{n} p^{\binom{n}{2}} \leq 2 + (n-1)\binom{n}{r^*} p^{\binom{r^*}{2}},$$

onde r^* é um valor de r que maximiza $\binom{n}{r} p^{\binom{r}{2}}$, isto é, r^* é o maior r para o qual

$$\frac{\binom{n}{r} p^{\binom{r}{2}}}{\binom{n}{r-1} p^{\binom{r-1}{2}}} = \frac{n-r+1}{r} p^{r-1} \geq 1.$$

Como

$$\frac{n-r+1}{r} p^{r-1} \leq np^{r-1}, \text{ para todo } r \in [n],$$

então

$$r^* \leq 1 + \log_{1/p} n.$$

Assim,

$$\begin{aligned}\mathbb{E}[N] &\leq 2 + (n-1)\binom{n}{r^*} p^{\binom{r^*}{2}} < 2 + (n-1)\binom{n}{r^*} \leq 2 + (n-1)n^{r^*} = 2 + n^{1+r^*} - n^{r^*} \\ &\leq n^{1+r^*} \leq n^{2+\log_{1/p} n} = n^{2+c_p \lg n}.\end{aligned}$$

□

Teorema 20. *O número esperado de estados na execução de* nobound *para um grafo G no modelo $\mathcal{G}_{n,p}$ é*

$$\mathbb{E}\left[|T_{\mathsf{nobound}}(G)|\right] = O\left(n^{2+\log_{1/p} n}\right).$$

Demonstração. Seja G um grafo no modelo $\mathcal{G}_{n,p}$. Pelo Teorema 11,

$$|T_{\mathsf{nobound}}(G)| = 2|\mathcal{C}(G)| - 1,$$

e pelo Corolário 19,

$$\mathbb{E}\left[|\mathcal{C}(G)|\right] = O\left(n^{2+\log_{1/p} n}\right).$$

Portanto,

$$\mathbb{E}\left[|T_{\mathsf{nobound}}(G)|\right] = O\left(n^{2+\log_{1/p} n}\right).$$

□

Assim, pelo Teorema 20 o algoritmo nobound tem número *subexponencial* de estados nos grafos do modelo $\mathcal{G}_{n,p}$. Isso significa que qualquer algoritmo MCBB deve ser subexponencial em média nesta classe de instâncias aleatórias. Esse fato, em uma primeira vista, poderia indicar que o modelo $\mathcal{G}_{n,p}$ não é adequado para avaliação de algoritmos para o CM. Em particular, uma eventual observação de que um algoritmo específico é subexponencial não apresentaria nenhum benefício, visto que o pior algoritmo de todos da família MCBB já é subexponencial.

Entretanto, o fato de os grafos aleatórios do modelo $\mathcal{G}_{n,p}$ terem comportamento previsível pode ser encarado como um benefício, se utilizados dentro de uma metodologia que leve em consideração o comportamento dos grafos. Tendo isso em mente, uma metodologia para análise experimental de algoritmos é apresentada no Capítulo 6, com base em alguns resultados apresentados a seguir.

4.1.3.1 Um pouco de história: algoritmos f–driven

Em Chvátal (1977) é apresentada uma família de algoritmos para solução do CIM, chamada de algoritmos f–driven. Como um conjunto independente no grafo G é uma clique no grafo $\overline{G}$, a redução entre o CIM e o CM é trivial.

Os algoritmos f–driven são algoritmos enumerativos que subdividem a instância em subproblemas, também chamados aqui de estados, de modo análogo ao Algoritmo MaxCliqueBB da Seção 2.2, porém em que cada subproblema contém apenas o conjunto K.

Os algoritmos f–driven não usam *bounding*, mas aplicam memoização para evitar a resolução de subproblemas duplicados. Dentro da família, cada algoritmo específico é obtido pela definição de uma função f, que é a função de escolha do vértice pivô. Ao manter o conjunto K ordenado e escolher como pivô o primeiro elemento do conjunto, obtém-se o algoritmo order–driven. Uma versão modificada para resolver CM similar ao Algoritmo MaxCliqueBB está disponível em pseudocódigo no Algoritmo order–driven–clique, que recebe um grafo G e devolve a maior clique de G.

order–driven–clique(G)

```
   Entrada: um grafo G.
   Saída: uma clique máxima de G.
 1 S ← pilha com um único estado V(G)
 2 M ← tabela vazia para memoização
 3 M[∅] ← ∅
 4 M[{v}] ← {v}, ∀v ∈ V(G)
 5 Enquanto S ≠ ∅ faça
 6 |   Desempilhe estado de S e guarde em K
 7 |   Se K ∉ M então
 8 |   |   Enquanto K ≠ ∅ e K ∉ M faça
 9 |   |   |   v ← primeiro vértice de K
10 |   |   |   Se K − {v} ∈ M e K ∩ Γ_G(v) ∈ M então
11 |   |   |   |   M[K] ← max {M[K − {v}], {v} ∪ M[K ∩ Γ_G(v)]}
12 |   |   |   Senão
13 |   |   |   |   Empilhe K em S
14 |   |   |   |   K ← K − {v}
15 |   |   |   |   Se K ∉ M então
16 |   |   |   |   |   Empilhe K em S
17 |   |   |   |   K ← K ∩ Γ_G(v)
18 Devolva M[V(G)]
```

No Algoritmo order–driven–clique, a tabela M é um vetor associativo que relaciona chaves, que são conjuntos de vértices, a valores, que também são conjuntos de vértices. As chaves são os conjuntos de candidatos do espaço de estados. O valor associado a uma chave K é a clique máxima em $G[K]$. A notação $K \in M$ é usada para indicar que uma chave K está na tabela, enquanto $K \notin M$ indica a ausência de K como chave na tabela. O valor associado a uma chave K é denotado $M[K]$. A tabela M contém subconjuntos de $V(G)$ e pode conter até 2^n elementos em um grafo de n vértices. Acesso em tempo linear em n a qualquer elemento pode ser obtido usando uma árvore balanceada para guardar as chaves.

Durante a execução do algoritmo, para um determinado estado K, se o valor já estiver memoizado em M, nenhum estado filho precisa ser criado. Caso não esteja, um pivô v é escolhido na linha 9, e o valor dos estados filhos é buscado em M. Caso os valores para os filhos ($K - \{v\}$ e $K \cap \Gamma_G(v)$) estejam em M, a clique máxima para $G[K]$ é calculada como $\max\{M[K - \{v\}], \{v\} \cup M[K \cap \Gamma_G(v)]\}$ na linha 11. Caso o valor de pelo menos um dos filhos de K ainda não tenha sido computado, o estado K é devolvido para a pilha $\mathcal{S}$, para que seu valor seja encontrado posteriormente, e os filhos necessários são criados de modo análogo aos estados dos algoritmos MCBB.

Como os subproblemas duplicados são guardados apenas uma vez via memoização, o espaço de busca dos algoritmos f–driven não forma uma árvore, mas sim um grafo acíclico direcionado. A Figura 4 apresenta o espaço de busca do algoritmo order–driven para o mesmo grafo cuja árvore de busca do algoritmo nobound está apresentada na Figura 2.

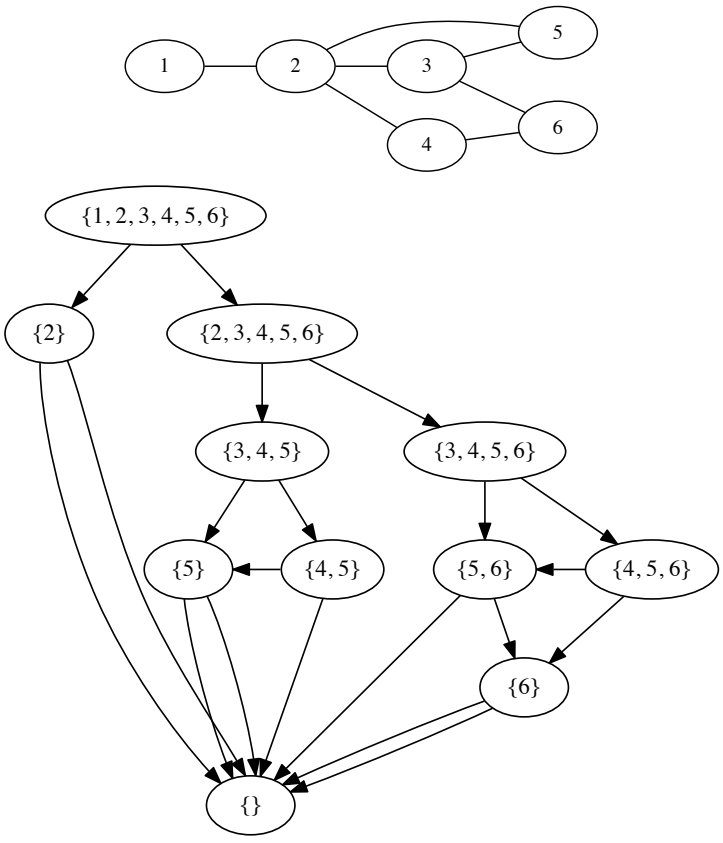


Figura 4 – Um grafo e o espaço de busca do algoritmo order–driven.

Os algoritmos f–driven não fazem parte da família MCBB, mas possuem um espaço de estados similar, então alguns resultados são relevantes apesar das diferenças algorítmicas. A Proposição 2 é a proposição de número 3.3 em Chvátal (1977) e aqui está refraseada para o contexto do CM.

Proposição 2. *Para quase todos os grafos com n vértices, o algoritmo* order–driven *cria no máximo $n^{2+\log_2 n}$ subproblemas.*

Em Pittel (1982), o comportamento dos algoritmos f–driven é estudado, porém ignorando-se a parte de memoização. Ou seja, os resultados apresentados, uma vez transpostos para CM no lugar do CIM, valem imediatamente para nobound. O principal resultado (Teorema 1) está reescrito na Proposição 3.

Proposição 3. *Para toda função para escolha de pivô f e todo $\varepsilon > 0$, seja $|T_{f\text{–driven}}(\mathcal{G}_{n,p})|$ o número de estados na execução do algoritmo f–driven em $\mathcal{G}_{n,p}$ incluindo-se estados duplicados, então,*

$$\mathbb{P}\left(n^{(0.25-\varepsilon)\log_{1/p} n} \leq |T_{f\text{–driven}}(\mathcal{G}_{n,p})| \leq n^{(0.5+\varepsilon)\log_{1/p} n}\right) \geq 1 - \exp(-c(\varepsilon)\ln^2 n),$$

onde $c(\varepsilon) > 0$.

Refraseando a Proposição 3 com alguma imprecisão, pode-se dizer que a razão

$$R_{f\text{–driven}}(\mathcal{G}_{n,p}) := \frac{\log_{1/p} |T_{f\text{–driven}}(\mathcal{G}_{n,p})|}{(\log_{1/p} n)^2}$$

está quase certamente entre $1/4$ e $1/2$.

4.1.3.2 Implicações para algoritmos MCBB

Considerando-se o exposto na seção anterior, para um grafo no modelo $\mathcal{G}_{n,p}$, o tamanho esperado da árvore de estados de nobound e, por consequência, seu tempo de execução esperado, são da ordem de $n^{c\log_{1/p} n}$, para $1/4 < c < 1/2$. É plausível, assim, conjeturar que os demais algoritmos MCBB tenham tempo esperado nesta mesma ordem. Uma metodologia para comparação de algoritmos baseada na ideia de estimar experimentalmente o valor de c é proposta no Capítulo 6. Nessa metodologia são usados grafos em $\mathcal{G}_{n,1/2}$, assim o resultado que embasa a proposta é o Corolário 23.

Teorema 21. *Seja $\mathcal{A}$ um algoritmo* MCBB *onde o pré-processamento do grafo e o processamento de cada estado é realizado em tempo polinomial em n. Seja $0 < p < 1$ uma constante e seja $c_p := 1/\lg(1/p)$. O custo de tempo médio do algoritmo $\mathcal{A}$ em grafos $\mathcal{G}_{n,p}$ é $O(n^{(c_p+\varepsilon)\lg n})$ para todo $\varepsilon > 0$.*

Demonstração. Sejam $\mathcal{A}$, p e c_p como acima. O tempo de execução de $\mathcal{A}$ em um grafo G é (4.3)

$$t_{\mathcal{A}}(G) \leq g_{\mathcal{A}}(G) + |T_{\mathcal{A}}(G)| f^*_{\mathcal{A}}(G).$$

Sejam $\alpha > 0$, $k > 0$ e $n_\alpha \in \mathbb{N}$ tais que $g_{\mathcal{A}}(G) \leq f^*_{\mathcal{A}}(G)$ e

$$f^*_{\mathcal{A}}(G) \leq \alpha |V(G)|^k, \text{ para todo } G \text{ onde } |V(G)| \geq n_\alpha.$$

Do Lema 14, temos que para todo grafo G tal que $|V(G)| \geq n_\alpha$,

$$t_{\mathcal{A}}(G) \leq g_{\mathcal{A}}(G) + |T_{\mathsf{nobound}}(G)| f^*_{\mathcal{A}}(G) \leq \alpha |V(G)|^k (1 + |T_{\mathsf{nobound}}(G)|).$$

Assim, no $\mathcal{G}_{n,p}$,

$$\mathbb{E}[t_{\mathcal{A}}] \leq \alpha n^k (1 + \mathbb{E}[|T_{\mathsf{nobound}}|]) \leq \alpha n^k (1 + 2n^{2+c_p \lg n} - 1) \leq 2\alpha n^{k+2+c_p \lg n}.$$

Por fim, para todo $\varepsilon > 0$, temos

$$n^{k+2+c_p \lg n} \leq n^{(c_p+\varepsilon)\lg n}, \text{ para todo } n \geq 2^{(k+2)/\varepsilon},$$

de modo que $\mathbb{E}[t_{\mathcal{A}}] = O(n^{(c_p+\varepsilon)\lg n})$. □

Teorema 22. *O número esperado de cliques em um grafo em $\mathcal{G}_{n,1/2}$ com $n \geq 2$ é*

$$\mathbb{E}[N] = n^{\frac{1}{2}\lg n(1+o(1))}.$$

Demonstração. Do Teorema 18 temos

$$\mathbb{E}[N] = \sum_{r=0}^{n} \binom{n}{r} p^{\binom{r}{2}} \leq \sum_{r=0}^{n} n^r p^{\binom{r}{2}} \leq 2 + (n-1) n^{r^*} p^{\binom{r^*}{2}}.$$

onde r^* é um valor de r que maximiza $n^r p^{\binom{r}{2}}$.

Para $p = 1/2$, derivando e encontrando o ponto de máximo[6] de $n^r p^{\binom{r}{2}}$, obtemos

$$r^* = -1/2 + \lg n.$$

Fixando $p = 1/2$, obtemos

$$\begin{aligned}
n^{r^*} p^{\binom{r^*}{2}} &= n^{r^*} \left(\frac{1}{2}\right)^{\binom{-1/2+\lg n}{2}} = \frac{n^{-1/2+\lg n}}{2^{\binom{-1/2+\lg n}{2}}} \leq \frac{n^{\lg n}}{2^{\frac{\lg^2 n}{2} - \lg n + \frac{3}{8}}} \leq \frac{n^{1+\lg n}}{2^{\frac{\lg^2 n}{2}}} \\
&= n^{1+\lg n} 2^{-\frac{\lg^2 n}{2}} \\
&= n^{1+\lg n} 2^{-\frac{\lg n \lg n}{2}} \\
&= n^{1+\lg n} n^{-\frac{\lg n}{2}} \\
&= n^{1+\frac{\lg n}{2}} \\
&= n^{\frac{1}{2}\lg n\left(\frac{2}{\lg n}+1\right)}.
\end{aligned}$$

[6] Este raciocínio é o mesmo que em Pittel (1982), mas aqui as contas estão mais detalhadas.

Então,

$$\begin{aligned}
\mathbb{E}[N] &\leq 2 + (n-1)n^{\frac{1}{2}\lg n\left(\frac{2}{\lg n}+1\right)} \\
&\leq 2 + n^{1+\frac{1}{2}\lg n\left(\frac{2}{\lg n}+1\right)} - n^{\frac{1}{2}\lg n\left(\frac{2}{\lg n}+1\right)} \\
&\leq n^{1+\frac{1}{2}\lg n\left(\frac{2}{\lg n}+1\right)} \\
&\leq n^{\frac{1}{2}\lg n\left(\frac{2}{\lg n}+\frac{2}{\lg n}+1\right)} \\
&= n^{\frac{1}{2}\lg n(1+o(1))}.
\end{aligned}$$

□

Corolário 23. *O número esperado de estados na execução de* nobound *para um grafo G em $\mathcal{G}_{n,1/2}$ com $n \geq 2$ é*

$$\mathbb{E}[|T_{\text{nobound}}(G)|] = n^{\frac{1}{2}\lg n(1+o(1))}.$$

Demonstração. Seja G um grafo no modelo $\mathcal{G}_{n,p}$. Pelo Teorema 11,

$$|T_{\text{nobound}}(G)| = 2|\mathcal{C}(G)| - 1,$$

e pelo Corolário 22,

$$\mathbb{E}[|\mathcal{C}(G)|] = n^{\frac{1}{2}\lg n(1+o(1))}.$$

Portanto,

$$\mathbb{E}[|T_{\text{nobound}}(G)|] = n^{\frac{1}{2}\lg n(1+o(1))}.$$

□

4.1.4 Melhor caso – o grafo sem arestas

Por completude, o Teorema 24 mostra que as instâncias que representam o melhor caso para o algoritmo nobound são os grafos sem arestas $\overline{K}_n$.

Teorema 24. *O tamanho da árvore de estados na execução de* nobound *para o grafo sem arestas com n vértices é $2n+1$, isto é,*

$$|T_{\text{nobound}}(\overline{K}_n)| = 2n + 1,$$

e esse é o melhor caso para nobound.

Demonstração. O conjunto $\mathcal{C}(\overline{K}_n)$ de cliques em $\overline{K}_n$ possui $n+1$ cliques, sendo uma vazia e outras n contendo cada um dos vértices do grafo. Assim, pelo Teorema 11,

$$|T_{\mathsf{nobound}}(\overline{K}_n)| = 2 \times (n+1) - 1 = 2n + 1.$$

Em geral, um grafo G qualquer com n vértices pode ter mais de zero arestas, e cada aresta adicionada aumenta no minimo uma clique em G, então $\overline{K}_n$ é o melhor caso para nobound e

$$|T_{\mathsf{nobound}}(G)| \geq 2n + 1.$$

□

4.2 ANÁLISE DO ALGORITMO BASIC

O algoritmo Basic é o algoritmo mais simples dentre os algoritmos MCBB que emprega um limitante superior para reduzir o tamanho da árvore de estados. O limite superior é o tamanho do conjunto de candidatos.

4.2.1 O grafo completo

O grafo completo representa o pior caso para o algoritmo nobound, que gera uma árvore com $2^{n+1} - 1$ estados para K_n. Entretanto, para o algoritmo Basic, a situação oposta ocorre. Por mais simples que seja o limitante superior de Basic, ele é suficiente para tornar o grafo completo tão simples de ser resolvido como o grafo sem arestas, como visto no Teorema 25.

Teorema 25. *O tamanho da árvore de estados na execução de* Basic *para o grafo completo com n vértices é $2n + 1$, isto é,*

$$|T_{\mathsf{Basic}}(K_n)| = 2n + 1.$$

Demonstração. A raiz de $T_{\mathsf{Basic}}(K_n)$ é o estado $(\emptyset, [n])$ e o seu descendente mais à esquerda é o estado $([n], \emptyset)$. Os estados na árvore que são filhos esquerdos de algum estado estão no caminho entre $(\emptyset, [n])$ e $([n], \emptyset)$ e são da forma $([i], [i+1..n])$. A clique máxima é guardada em C no estado $([n], \emptyset)$. Cada estado na árvore que é filho direito de outro estado tem a forma $([i], [i+2..n])$, de modo que

$$|Q| + |K| = |[i]| + |[i+2..n]| = i + (n - i - 1) = n - 1 < n = |C|.$$

Com isso, a árvore é podada imediatamente em cada um dos estados que são filhos direitos e a árvore tem um estado raiz, n estados que são filhos esquerdos e n estados que são filhos direitos, ou seja,

$$|T_{\mathsf{Basic}}(K_n)| = 1 + n + n = 2n + 1.$$

□

Para ilustrar o comportamento apresentado no Teorema 25, a Figura 5 apresenta a árvore gerada pelo algoritmo Basic para o grafo completo com quatro vértices.

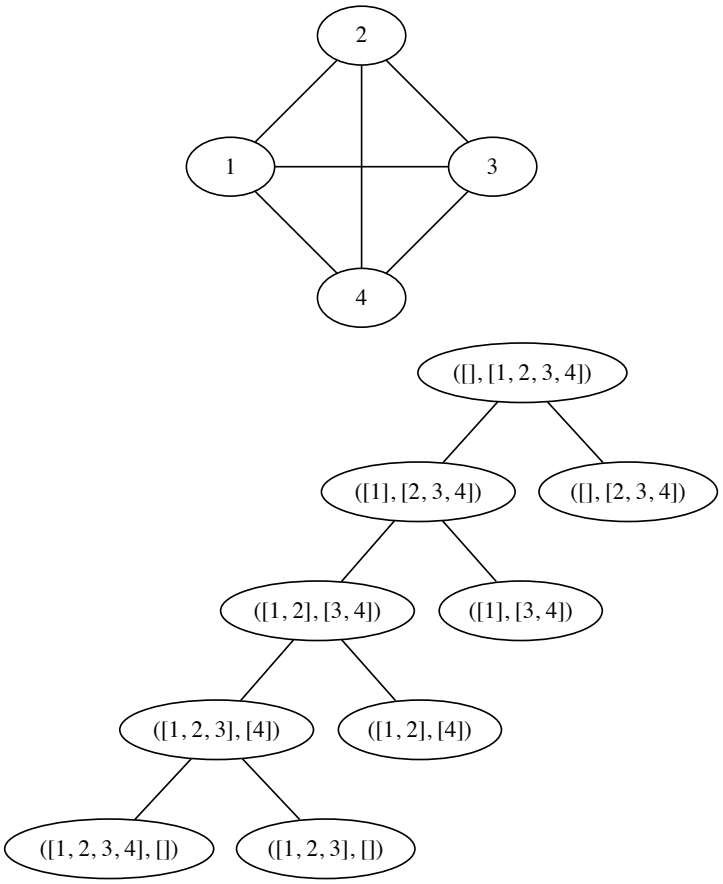


Figura 5 – O grafo K_4 e a árvore de estados $T_{\mathsf{Basic}}(K_4)$.

4.2.2 Grafos de Moon–Moser

Para o algoritmo nobound, os grafos de Moon–Moser obrigam a geração de uma árvore de estados de tamanho exponencial. No caso do algoritmo Basic, a adição de um limitante superior faz com que o número de estados seja reduzido de maneira significativa. Ao mesmo tempo, a análise se torna mais complicada. O tamanho da árvore pode ser

representado por uma relação de recorrência, mas não foi possível encontrar uma solução em forma fechada para tal relação. Os resultados apresentados nesta seção mostram que, de todo modo, a árvore de estados do algoritmo Basic tem tamanho exponencial para os grafos de Moon–Moser.

Para um algoritmo $\mathcal{A}$, denota-se por $T_{\mathcal{A}}(G, Q, K)$ a subárvore de $T_{\mathcal{A}}(G)$ que tem como raiz o estado (Q, K) e

$$T_{\mathcal{A},n}(Q, K) := T_{\mathcal{A}}(M(n), Q, K).$$

Sejam $v \in K$ o pivô do estado (Q, K) e λ a árvore vazia, então, para $\mathcal{A} = \mathsf{nobound}$,

$$T_{\mathsf{nobound},n}(Q, K) = \begin{cases} (\lambda, \lambda), & \text{se } K = \emptyset, \\ (T_{\mathsf{nobound},n}(Q \cup \{v\}, K \cap \Gamma_{\cap}(v)), T_{\mathsf{nobound},n}(Q, K - \{v\})), & \text{se } K \neq \emptyset, \end{cases}$$

e para $\mathcal{A} = \mathsf{Basic}$,

$$T_{\mathcal{A},n}(Q, K) = \begin{cases} (\lambda, \lambda), & \text{se } |Q| + |K| \leq \omega(M(n)), \\ (T_{\mathcal{A},n}(Q \cup \{v\}, K \cap \Gamma_{\cap}(v)), T_{\mathcal{A},n}(Q, K - \{v\})), & \text{se } |Q| + |K| > \omega(M(n)). \end{cases} \tag{4.4}$$

4.2.2.1 Caso n múltiplo de três

Nesta seção, são analisados o grafos quando $n \equiv 0 \pmod 3$. Os resultados são generalizados para todos os casos na próxima seção.

Sem perda de generalidade, vamos pressupor que os vértices de cada triângulo em $\overline{M(n)}$ são rotulados com números subsequentes, ou seja, os conjuntos $\{3i - 2, 3i - 1, 3i\}$ com $i \in [n/3]$ são independentes em $M(n)$. Assim, o vértice de rótulo 1 é o pivô no estado $(\emptyset, V(M(n)))$, o vértice 2 é o pivô em seu filho direito e o vértice 3 é o pivô em $(\emptyset, V(M(n)) - \{1, 2\})$. Em geral, se x é pivô em (Q, K), o vértice $x + 1$ é pivô em $(Q, K - \{x\})$. Assim, o grafo $M(n)$ é um grafo regular onde os vértices são, para todos os efeitos, indistinguíveis entre si.

Com isso, o valor de $|T_{\mathsf{Basic},n}(Q, K)|$ depende somente do valores de n, $|Q|$ e $|K|$. Além disso,

$$|K \cap \Gamma_{\cap}(v)| = 3 \left\lfloor \frac{|K| - 1}{3} \right\rfloor$$

pois um, dois ou três vértices são excluídos de K.

Seja $t(n, q, k)$ o tamanho de $T_{\mathsf{Basic},n}(Q, K)$ quando $q = |Q|$ e $k = |K|$.

O algoritmo Basic encontra uma clique máxima logo no estado folha que é o descendente mais à esquerda da raiz. Além disso, quando o conjunto K está vazio, o tamanho de Q é igual a $n/3$. Considerando estes fatos, temos de (4.4) que

$$t(n,q,k) = \begin{cases} 1, & \text{se } q+k \leq \frac{n}{3}, \\ 1 + t\left(n, q+1, 3\left\lfloor \frac{k-1}{3} \right\rfloor\right) + t(n,q,k-1), & \text{se } q+k > \frac{n}{3}. \end{cases} \tag{4.5}$$

Assim,

$$|T_{\mathsf{Basic}}(M(n))| = |T_{\mathsf{Basic},n}(\emptyset, V(M(n)))| = t(n,0,n),$$

e é de se esperar que

$$|T_{\mathsf{Basic}}(M(n))| = \Theta(b^n).$$

4.2.2.2 Caso geral

Quando n não é múltiplo de três, existem um ou dois fragmentos com apenas dois vértices que devem ser levados em consideração. O caso $n \equiv 2 \pmod 3$ é simples, pois existe apenas um fragmento de tamanho dois. Pressupondo que os vértices deste fragmento têm rótulos 1 e 2, ou seja, serão os primeiros pivôs, a árvore $T_{\mathsf{Basic}}(M(n))$ é exatamente igual à subárvore direita de $T_{\mathsf{Basic}}(M(n+1))$. Para visualizar esse fato, basta imaginar que o primeiro fragmento tem mais um vértice de rótulo 0, voltando-se, assim, ao caso em que n é múltiplo de três. Esse vértice imaginário é o primeiro pivô e é, portanto, descartado de K no filho direito. Ou seja, o valor de $t(n,q,k)$ dado por (4.5) também é válido para o caso $n \equiv 2 \pmod 3$.

No caso $n \equiv 1 \pmod 3$, com $n > 1$, existem dois fragmentos de tamanho dois. Pressupondo que os vértices destes fragmentos têm rótulos 1 e 2 para o primeiro fragmento e 3 e 4 para o segundo, o algoritmo chega em uma situação análoga ao caso $n \equiv 1 \pmod 3$ após os vértices 1 e 2 serem pivôs. O estado raiz de $T_{\mathsf{Basic}}(M(n))$ é o estado $(\emptyset, V(M(n)))$ e seu filho esquerdo é $(\{1\}, V(M(n)) - \{1,2\})$ que enraíza uma árvore de tamanho $t(n,1,n-2)$. O filho direito da raiz é o estado $X = (\emptyset, V(M(n)) - \{1\})$, e seus filhos são $E(X) = (\{2\}, V(M(n)) - \{1,2\}$ e $D(X) = (\emptyset, V(M(n)) - \{1,2\}$. A subárvore enraizada por $E(X)$ tem tamanho $t(n,1,n-2)$ e a subárvore enraizada por $D(X)$ tem tamanho $t(n,0,n-2)$. Assim, o tamanho da árvore no caso $n \equiv 1 \pmod 3$, com $n > 1$, é dado pela contagem de todos esses estados, isto é

$$|T_{\mathsf{Basic}}(M(n))| = 2 + 2t(n,1,n-2) + t(n,0,n-2).$$

Considerando todos esses casos e cuidando do caso $n = 1$ como especial, temos

$$|T_{\mathsf{Basic}}(M(n))| = \begin{cases} 3, & \mathsf{se}\ n = 1, \\ t(n, 0, n), & \mathsf{se}\ n \equiv 0 \pmod 3, \\ 2 + 2t(n, 1, n-2) + t(n, 0, n-2), & \mathsf{se}\ n \equiv 1 \pmod 3, \\ t(n, 0, n), & \mathsf{se}\ n \equiv 2 \pmod 3. \end{cases}$$

Não foi possível encontrar uma solução exata em forma fechada para (4.5). Assim, valores de $|T_{\mathsf{Basic}}(M(n))|$ foram calculados para n até 1000, visando a identificação de padrões. Os dados calculados para n até 48 estão na Tabela 4.

Diversos dados foram computados a patir da sequência dada pelos valores de $|T_{\mathsf{Basic}}(M(n))|$, e são exibidos na tabela. Apesar de os valores de $|T_{\mathsf{Basic}}(M(n))|$ terem sido calculados de forma exata, todos os outros valores foram computados utilizando números de ponto flutuante, então a presença de erros numéricos deve ser levada em consideração.

A coluna $|T_{\mathsf{Basic}}(M(n))|/|T_{\mathsf{Basic}}(M(n-1))|$ exibe a razão entre um valor e o valor anterior. Esta razão parece tender para um valor próximo a 1.4 quando $n \equiv 0 \pmod 3$ ou $n \equiv 2 \pmod 3$ e para um valor próximo a 1.8 quando $n \equiv 1 \pmod 3$.

Supondo que $|T_{\mathsf{Basic}}(M(n))|$ seja $\Theta(b^n)$, a coluna b^* apresenta uma estimativa para a base b, calculada como

$$b^* = \sqrt[3]{\frac{|T_{\mathsf{Basic}}(M(n))|}{|T_{\mathsf{Basic}}(M(n-1))|} \times \frac{|T_{\mathsf{Basic}}(M(n-1))|}{|T_{\mathsf{Basic}}(M(n-2))|} \times \frac{|T_{\mathsf{Basic}}(M(n-2))|}{|T_{\mathsf{Basic}}(M(n-3))|}}.$$

Os valores b^* convergem rapidamente para um valor próximo a 1.523. Para todo $n > 175, b^* = 1.52292041348$. Assim, hipotetiza-se que $|T_{\mathsf{Basic}}(M(n))|$ é $\Theta(1.52292041348^n)$.

A próxima coluna da Tabela 4, $|T_{\mathsf{Basic}}(M(n))|/b^n$, apresenta a razão entre os valores computados $|T_{\mathsf{Basic}}(M(n))|$ e b^n, com $b = 1.52292041348$. Esta razão será denotada por α. Se for possível decidir um valor específico para α, então teremos $|T_{\mathsf{Basic}}(M(n))| \approx \alpha b^n$. Como os valores da razão $|T_{\mathsf{Basic}}(M(n))|/|T_{\mathsf{Basic}}(M(n-1))|$ convergem para valores diferentes dependendo do resto da divisão de n por três, o valor de α será diferente para cada um dos três casos. Para $n > 216$, os valores de α encontrados foram

$$\alpha = \begin{cases} 1.91959016105, & \mathsf{se}\ n \equiv 0 \pmod 3, \\ 2.28800820259, & \mathsf{se}\ n \equiv 1 \pmod 3, \\ 2.09571897784, & \mathsf{se}\ n \equiv 2 \pmod 3. \end{cases}$$

Assim, pode-se afirmar empiricamente que

$$|T_{\mathsf{Basic}}(M(n))| \approx \begin{cases} 1.91959016105 \times 1.52292041348^n, & \text{se } n \equiv 0 \pmod 3, \\ 2.28800820259 \times 1.52292041348^n, & \text{se } n \equiv 1 \pmod 3, \\ 2.09571897784 \times 1.52292041348^n, & \text{se } n \equiv 2 \pmod 3. \end{cases} \tag{4.6}$$

A coluna αb^n da Tabela 4 apresenta valores estimados usando (4.6). A última coluna apresenta o erro relativo entre os valores estimados para $|T_{\mathsf{Basic}}(M(n))|$ e os valores calculados de forma exata.

A Tabela 5 apresenta dados com para n selecionados até 1000. O erro relativo entre o valor estimado e o valor exato de $|T_{\mathsf{Basic}}(M(n))|$ diminui até $n = 150$, mas depois se torna estável e crescente devido à falta de precisão numérica. Ainda assim, a estimativa é bastante satisfatória visto que, mesmo para $n = 1000$, o erro só aparece no décimo dígito; o valor exato de $|T_{\mathsf{Basic}}(M(1000))|$ tem 184 dígitos[7].

Para efeitos de validação dos dados, a implementação do algoritmo Basic foi executado para os grafos de Moon–Moser, com n variando de um a 48. O número de estados e o tempo de execução estão apresentados na Tabela 6. É de fundamental importância o fato de que o tamanho da árvore criada experimentalmente ficou de acordo com os calculados para $|T_{\mathsf{Basic}}(M(n))|$.

4.3 COLORAÇÃO COMO LIMITANTE SUPERIOR

Diversos algoritmos da família MCBB usam coloração como limitante superior, por vezes em conjunto com outras técnicas e outros limitantes. Nesta seção são apresentados resultados para o algoritmo MCLIQ que, como visto na Seção 2.2.3, é o primeiro algoritmo proposto da família que usa coloração. Os grafos analisados são o grafo completo e os grafos de Moon–Moser. Para estes últimos, também é apresentado o tamanho da árvore para os algoritmos MCQ, MCR, MaxCliqueDyn, MCS, χ, $\chi + \mathsf{DF}$, MaxCLQ e BBMCX.

4.3.1 O grafo completo

O algoritmo MCLIQ gera uma árvore de estados para o grafo completo com o mesmo tamanho que a gerada pelo algoritmo Basic, como visto no Teorema 26.

[7] $|T_{\mathsf{Basic}}(M(1000))| = 10880920779755021653178273212768281208893807310811157153054265557037292617659233091965708520238580175569818521799847042246228801843202291954380962919410719052925734749967041754$[illegible]

Tabela 4 – Dados do algoritmo Basic em grafos de Moon–Moser com n até 48.

n	ω	$\lvert T_{\mathsf{Basic}}(M(n))\rvert$	$\frac{\lvert T_{\mathsf{Basic}}(M(n))\rvert}{\lvert T_{\mathsf{Basic}}(M(n-1))\rvert}$	b^*	$\frac{\lvert T_{\mathsf{Basic}}(M(n))\rvert}{b^n}$	αb^n	Erro relativo
1	1	3			2.0801	3.5	1.39×10^{-1}
2	1	3	1.0000		1.4422	4.9	3.83×10^{-1}
3	1	5	1.6667		1.0000	6.8	2.63×10^{-1}
4	2	9	1.8000	1.4422	2.0801	12	2.69×10^{-1}
5	2	15	1.6667	1.7100	1.0260	17	1.26×10^{-1}
6	2	21	1.4000	1.6134	1.1905	24	1.23×10^{-1}
7	3	41	1.9524	1.6577	1.1918	43	5.68×10^{-2}
8	3	57	1.3902	1.5605	1.6210	61	6.00×10^{-2}
9	3	79	1.3860	1.5553	1.4839	85	6.60×10^{-2}
10	4	149	1.8861	1.5374	2.0192	154	2.96×10^{-2}
11	4	207	1.3893	1.5371	1.8293	214	3.35×10^{-2}
12	4	287	1.3865	1.5373	1.6476	299	3.94×10^{-2}
13	5	533	1.8571	1.5294	2.1284	542	1.72×10^{-2}
14	5	741	1.3902	1.5297	1.9284	757	2.05×10^{-2}
15	5	1029	1.3887	1.5305	1.7368	1055	2.49×10^{-2}
16	6	1895	1.8416	1.5263	2.1856	1916	1.07×10^{-2}
17	6	2637	1.3916	1.5267	1.9821	2672	1.31×10^{-2}
18	6	3667	1.3906	1.5274	1.7903	3727	1.62×10^{-2}
19	7	6719	1.8323	1.5249	2.2177	6766	6.93×10^{-3}
20	7	9357	1.3926	1.5253	2.0151	9438	8.58×10^{-3}
21	7	13 025	1.3920	1.5258	1.8260	13 165	1.07×10^{-2}
22	8	23 789	1.8264	1.5241	2.2380	23 898	4.55×10^{-3}
23	8	33 147	1.3934	1.5244	2.0374	33 336	5.66×10^{-3}
24	8	46 173	1.3930	1.5248	1.8514	46 501	7.05×10^{-3}
25	9	84 155	1.8226	1.5237	2.2519	84 409	3.01×10^{-3}
26	9	117 303	1.3939	1.5239	2.0534	117 744	3.75×10^{-3}
27	9	163 477	1.3936	1.5241	1.8700	164 245	4.68×10^{-3}
28	10	297 545	1.8201	1.5234	2.2619	298 139	1.99×10^{-3}
29	10	414 849	1.3942	1.5236	2.0651	415 884	2.49×10^{-3}
30	10	578 327	1.3941	1.5237	1.8837	580 129	3.11×10^{-3}
31	11	1 051 661	1.8185	1.5233	2.2692	1 053 055	1.32×10^{-3}
32	11	1 466 511	1.3945	1.5233	2.0737	1 468 939	1.65×10^{-3}
33	11	2 044 839	1.3944	1.5235	1.8937	2 049 068	2.06×10^{-3}
34	12	3 716 213	1.8174	1.5231	2.2745	3 719 484	8.79×10^{-4}
35	12	5 182 725	1.3946	1.5232	2.0799	5 188 422	1.10×10^{-3}
36	12	7 227 565	1.3945	1.5233	1.9011	7 237 490	1.37×10^{-3}
37	13	13 129 871	1.8166	1.5231	2.2784	13 137 547	5.84×10^{-4}
38	13	18 312 597	1.3947	1.5231	2.0844	18 325 969	7.30×10^{-4}
39	13	25 540 163	1.3947	1.5232	1.9064	25 563 460	9.11×10^{-4}
40	14	46 384 967	1.8162	1.5230	2.2811	46 402 983	3.88×10^{-4}
41	14	64 697 565	1.3948	1.5230	2.0877	64 728 952	4.85×10^{-4}
42	14	90 237 729	1.3948	1.5231	1.9102	90 292 412	6.06×10^{-4}
43	15	163 857 173	1.8158	1.5230	2.2832	163 899 461	2.58×10^{-4}
44	15	228 554 739	1.3948	1.5230	2.0900	228 628 413	3.22×10^{-4}
45	15	318 792 469	1.3948	1.5230	1.9130	318 920 825	4.02×10^{-4}
46	16	578 808 203	1.8156	1.5230	2.2846	578 907 463	1.71×10^{-4}
47	16	807 362 943	1.3949	1.5230	2.0917	807 535 876	2.14×10^{-4}
48	16	1 126 155 413	1.3949	1.5230	1.9149	1 126 456 700	2.67×10^{-4}

Tabela 5 – Valores calculados para grafos de Moon–Moser com o algoritmo Basic.

n	ω	$\lvert T_{\mathsf{Basic}}(M(n))\rvert$	αb^n	Erro relativo
50	17	2.85×10^{9}	2.85×10^{9}	1.42×10^{-4}
100	34	4.24×10^{18}	4.24×10^{18}	1.09×10^{-7}
150	50	4.84×10^{27}	4.84×10^{27}	6.00×10^{-11}
200	67	7.19×10^{36}	7.19×10^{36}	4.12×10^{-10}
250	84	1.07×10^{46}	1.07×10^{46}	5.13×10^{-10}
300	100	1.22×10^{55}	1.22×10^{55}	6.16×10^{-10}
350	117	1.81×10^{64}	1.81×10^{64}	7.20×10^{-10}
400	134	2.69×10^{73}	2.69×10^{73}	8.22×10^{-10}
450	150	3.08×10^{82}	3.08×10^{82}	9.25×10^{-10}
500	167	4.57×10^{91}	4.57×10^{91}	1.03×10^{-9}
550	184	6.79×10^{100}	6.79×10^{100}	1.13×10^{-9}
600	200	7.75×10^{109}	7.75×10^{109}	1.23×10^{-9}
650	217	1.15×10^{119}	1.15×10^{119}	1.34×10^{-9}
700	234	1.71×10^{128}	1.71×10^{128}	1.44×10^{-9}
750	250	1.95×10^{137}	1.95×10^{137}	1.54×10^{-9}
800	267	2.90×10^{146}	2.90×10^{146}	1.65×10^{-9}
850	284	4.32×10^{155}	4.32×10^{155}	1.75×10^{-9}
900	300	4.93×10^{164}	4.93×10^{164}	1.85×10^{-9}
950	317	7.32×10^{173}	7.32×10^{173}	1.95×10^{-9}
1000	334	1.09×10^{183}	1.09×10^{183}	2.06×10^{-9}

Teorema 26. *O tamanho da árvore de estados na execução de* MCLIQ *para o grafo completo com n vértices é* $2n + 1$*, isto é,*

$$|T_{\mathsf{MCLIQ}}(K_n)| = 2n + 1.$$

Demonstração. Em toda coloração de K_n, existe uma cor para cada um dos vértices. Além disto, todo subgrafo induzido de K_n com m vértices é colorido com m cores, qualquer que seja a coloração. Assim, para todo estado $(Q, K) \in T_{\mathsf{MCLIQ}}(K_n)$, o limitante superior dado pelo número de cores em uma coloração é igual a $|K|$. Como este valor é exatamente o valor do limitante superior utilizado pelo algoritmo Basic, o resultado segue do Teorema 25. □

4.3.2 Grafos de Moon–Moser

O grafo de Moon–Moser $M(n)$ para n múltiplo de três é o grafo junção de $n/3$ grafos sem arestas com três vértices, isto é, $M(n) = \frac{n}{3}\overline{K}_3$. Este grafo têm cliques máximas de tamanho $\omega(M(n)) = n/3$ e número cromático $\chi(M(n)) = n/3$. Assim, o número cromático é igual ao tamanho da maior clique. Além disso, $M(n)$ é um grafo perfeito, isto é, $\omega(M(n)[X]) = \chi(M(n)[X])$, para todo $X \subseteq V(M(n))$.

Tabela 6 – Resultados experimentais para grafos de Moon–Moser com o algoritmo Basic.

n	$\omega(M(n))$	$\lvert T_{\mathsf{Basic}}(M(n))\rvert$	Tempo (s)
1	1	3	0.0000
2	1	3	0.0000
3	1	5	0.0000
4	2	9	0.0001
5	2	15	0.0002
6	2	21	0.0010
7	3	41	0.0006
8	3	57	0.0008
9	3	79	0.0030
10	4	149	0.0022
11	4	207	0.0031
12	4	287	0.0044
13	5	533	0.0083
14	5	741	0.0122
15	5	1029	0.0161
16	6	1895	0.0307
17	6	2637	0.0526
18	6	3667	0.0756
19	7	6719	0.1110
20	7	9357	0.1670
21	7	13 025	0.2420
22	8	23 789	0.4418
23	8	33 147	0.6090
24	8	46 173	0.8315
25	9	84 155	1.6241
26	9	117 303	2.1817
27	9	163 477	3.0008
28	10	297 545	6.3194
29	10	414 849	9.2707
30	10	578 327	12.5664
31	11	1 051 661	23.6067
32	11	1 466 511	36.9261
33	11	2 044 839	46.7473
34	12	3 716 213	88.4083
35	12	5 182 725	121.9285
36	12	7 227 565	157.2450
37	13	13 129 871	328.2524
38	13	18 312 597	461.1147
39	13	25 540 163	627.9143
40	14	46 384 967	1198.3578
41	14	64 697 565	1740.5243
42	14	90 237 729	2173.8991
43	15	163 857 173	4163.4834
44	15	228 554 739	5814.8974
45	15	318 792 469	7724.3961
46	16	578 808 203	14 080.9784
47	16	807 362 943	19 305.7687
48	16	1 126 155 413	28 522.0266

Assim, se um algoritmo conseguir colorir o grafo $M(n)$ e seus subgrafos induzidos pelos conjuntos de candidatos de forma ótima, ocorre um fenômeno análogo ao ocorrido para o grafo completo, isto é, a clique máxima é encontrada rapidamente e a árvore de estados é podada em todos os filhos direitos. O Algoritmo Guloso cumpre esta condição, como mostrado no Lema 27, e, consequentemente, o algoritmo MCLIQ cria uma árvore de tamanho linear no número de vértices do grafo, como apresentado no Teorema 28.

Lema 27. *Seja $G = \sum_{i=1}^{k} G_i$ um grafo junção com k fragmentos onde cada G_i, com $i \in [k]$, é grafo sem arestas, então o Algoritmo* Guloso *colore G com k cores.*

Demonstração. O Algoritmo Guloso colore um vértice de G de cada vez e cada vértice é colorido com a menor cor possível. Como cada G_i é grafo sem arestas e há aresta em G ligando cada vértice em G_i a todos os vértices que não fazem parte de G_i, todos os vértices de G_i são coloridos com a mesma cor e esta é diferente de todas as cores dos vértices que não estão em G_i. Assim, o Algoritmo Guloso cria uma cor para cada fragmento G_i e o número total de cores é k. □

Teorema 28. *O tamanho da árvore de estados na execução de* MCLIQ *para $M(n)$ é*

$$|T_{\mathsf{MCLIQ}}(M(n))| = \begin{cases} \frac{2n+3}{3}, & \text{se } n \equiv 0 \pmod 3, \\ \frac{2n+7}{3}, & \text{se } n \equiv 1 \pmod 3, \\ \frac{2n+5}{3}, & \text{se } n \equiv 2 \pmod 3. \end{cases}$$

Demonstração. A raiz de $T_{\mathsf{MCLIQ}}(M(n))$ é o estado $(\emptyset, [n])$ e o seu descendente mais à esquerda é o estado $(\mathcal{C}, \emptyset)$, onde $\mathcal{C}$ é uma clique máxima de $M(n)$. Os estados na árvore que são filhos esquerdos de algum estado estão no caminho entre $(\emptyset, [n])$ e $(\mathcal{C}, \emptyset)$ e são da forma $(X, [3|X| + 1..n])$. A clique máxima é guardada em C no estado $(\mathcal{C}, \emptyset)$. Cada estado na árvore que é filho direito de outro estado tem a forma $(X, [3|X| + 2..n])$ e, pelo Lema 27, o Algoritmo Guloso colore $G[K]$ usando $\chi(G[K])$ cores. Assim, para todo estado que é filho direito de outro estado,

$$|Q| + \chi(M(n)[K]) = |Q| + (\omega(M(n)) - |Q|) = \omega(M(n)).$$

Com isso, a árvore é podada imediatamente em cada um dos estados que são filhos direitos e a árvore tem raiz, $\omega(M(n))$ estados que são filhos esquerdos e $\omega(M(n))$ estados que são filhos direitos, ou seja,

$$|T_{\mathsf{MCLIQ}}(K_n)| = 1 + \omega(M(n)) + \omega(M(n)) = 2\omega(M(n)) + 1$$

Tabela 7 – Tamanho da árvore de estados $T_{\mathcal{A}}(M(n))$ para vários algoritmos $\mathcal{A}$ que usam coloração.

$\mathcal{A}$	$T_{\mathcal{A}}(M(n))$ $n \equiv 0 \pmod 3$	$n \equiv 2 \pmod 3$	$n \equiv 1 \pmod 3$
MCQ, MaxCliqueDyn e BBMCX	$2n-1$	$2n-1$	$2n-1$
MCR e MCS	$\frac{2n+3}{3}$	$\frac{4n-5}{3}$	$\frac{4n-7}{3}$
MCLIQ, χ, $\chi+$DF e MaxCLQ	$\frac{2n+3}{3}$	$\frac{2n+5}{3}$	$\frac{2n+7}{3}$

e o resultado segue do valor de $\omega(M(n))$ que é

$$\omega(M(n)) = \begin{cases} \frac{n}{3}, & \text{se } n \equiv 0 \pmod 3, \\ \frac{n+2}{3}, & \text{se } n \equiv 1 \pmod 3, \\ \frac{n+1}{3}, & \text{se } n \equiv 2 \pmod 3. \end{cases}$$

□

Diferentemente do algoritmo MCLIQ, vários algoritmos não realizam a coloração no estado raiz, em uma tentativa de reduzir o tempo total de processamento à custa de uma árvore de estados um pouco maior. Com isso, existem algoritmos que são derivados de MCLIQ e geram uma árvore de estados maior que este.

Os tamanhos das árvores para os algoritmos MCQ, MCR, MCS, MaxCliqueDyn, χ e $\chi+$DF foram inferidos em Anjos (2015) a partir de resultados experimentais. No presente trabalho, também foram avaliados experimentalmente os algoritmos MaxCLQ e BBMCX, que estão apresentados no Capítulo 5. O tamanho da árvore para o algoritmo MCLIQ foi determinado analiticamente no Teorema 28 e conferido experimentalmente. O tamanho das árvores geradas por todos estes algoritmos para os grafos $M(n)$ está apresentado na Tabela 7.

5 CLIQUE MÁXIMA E PMAXSAT

O problema da Satisfatibilidade Booleana (Sat) é o problema de decidir se uma fórmula booleana em forma normal conjuntiva é satisfatível. O problema Sat foi o primeiro problema a ser provado $\mathcal{NP}$–completo, por Cook (1971) e independentemente por Levin (1973) (artigo em russo, com tradução para o inglês disponível (TRAKHTENBROT, 1984)). Existem diversas variações do problema Sat, incluindo versões de decisão e de otimização, e esses problemas estão entre os mais estudados da Ciência da Computação. O problema da Satisfatibilidade Máxima (MaxSat) consiste em encontrar uma valoração para as variáveis booleanas que maximiza o número de cláusulas satisfeitas em uma fórmula dada em forma normal conjuntiva. O problema da Satisfatibilidade Máxima Parcial (PMaxSat) é uma generalização de MaxSat onde parte das cláusulas deve obrigatoriamente ser satisfeita por qualquer valoração. Tanto MaxSat quanto PMaxSat são problemas de maximização $\mathcal{NP}$–difíceis, assim como CM. A terminologia e a notação relativas a estes problemas estão apresentadas na Seção 5.1.

O problema PMaxSat possui relevância imediata no estudo de CM pela existência de uma redução bastante simples de CM para PMaxSat, apresentada na Seção 5.2. Assim, a clique máxima de um grafo pode ser encontrada pela codificação do grafo em uma instância de PMaxSat e submissão a um resolvedor. Entretanto, segundo Li e Quan (2010b), usar resolvedores PMaxSat para resolver instâncias convertidas não resulta em tempos competitivos com algoritmos de branch–and–bound para CM, mas não estão disponíveis na literatura dados que demonstrem tal fenômeno. O esquema de branch–and–bound para PMaxSat é semelhante ao dos algoritmos MCBB então seu estudo pode levar a novas ideias para algoritmos para CM. De fato, recentemente técnicas oriundas de resolvedores PMaxSat têm sido adaptadas para uso dentro de algoritmos de branch–and–bound para CM que utilizam coloração.

Algoritmos de coloração são amplamente utilizados como limitante superior em algoritmos de branch–and–bound para clique máxima, porém tal limitante pode não ser justo. Além disso, algoritmos que usem apenas coloração para reduzir o espaço de busca possuem custo de tempo comprovadamente exponencial no pior caso, como visto na Seção 3.2. Desde 2010, técnicas provenientes de resolvedores para PMaxSat estão sendo adaptadas para algoritmos de branch–and–bound para CM como tentativas de

criar limitantes mais apertados que os dados pela coloração, com base em uma redução nova entre os dois problemas (LI; QUAN, 2010b), também apresentada na Seção 5.2. De fato, é possível que um limitante adaptado resulte em um valor que é menor que o número cromático do grafo, e por isso foi proposto chamar os limitantes superiores resultantes da aplicação destas técnicas de limitantes *infra-cromáticos* (SAN SEGUNDO; NIKOLAEV; BATSYN, 2015). Diversos trabalhos que adaptam técnicas oriundas de resolvedores PMaxSat já estão publicados na literatura (LI; QUAN, 2010b; LI; QUAN, 2010a; LI; FANG; XU, 2013; LI; JIANG; XU, 2015; MASLOV; BATSYN; PARDALOS, 2014; SAN SEGUNDO; TAPIA; LOPEZ, 2013; SAN SEGUNDO; NIKOLAEV; BATSYN, 2015; SAN SEGUNDO et al., 2016b; SAN SEGUNDO et al., 2016a; SAN SEGUNDO et al., 2017; LI; JIANG; MANYÀ, 2017).

O primeiro algoritmo proposto na literatura que aplica uma heurística proveniente de resolvedores para o problema PMaxSat como uma tentativa de gerar um limitante superior mais apertado para o CM é o algoritmo MaxCLQ (LI; QUAN, 2010b). Tal heurística depende de uma redução entre os dois problemas.

Esquematicamente a ideia pode ser descrita como segue. Dados um grafo G e uma coloração de G, uma certa instância de PMaxSat é computada. Essa instância é submetida a uma subrotina de um resolvedor PMaxSat que implementa uma heurística chamada *detecção de literais falhos* (em inglês, *failed literal detection*). Se essa heurística encontrar um limitante superior para o número de cláusulas satisfatíveis na instância, então esse limitante é convertido para um limitante superior para o tamanho da clique máxima em G, e é mais apertado que o número de cores na coloração inicial. O algoritmo resultante, chamado algoritmo MaxCLQ, faz parte da família MCBB, uma vez que todo este processamento faz parte da chamada à função upper-bound.

A descrição do algoritmo resultante usando cálculo proposicional obscurece o seu sentido em termos de grafos. Na Seção 5.3, tal heurística é reformulada sem recorrer à terminologia de PMaxSat, usando apenas termos da teoria dos grafos. Esta descrição do algoritmo MaxCLQ foi apresentada em Züge e Carmo (2014) e está publicada em Züge e Carmo (2016).

Posteriormente, foi proposto o algoritmo BBMCX (SAN SEGUNDO; NIKOLAEV; BATSYN, 2015). Este algoritmo também usa uma heurística de detecção de literais falhos que é uma simplificação da heurística adotada no algoritmo MaxCLQ. Tal heurística está explicada na Seção 5.4.

Como dito anteriormente, existem indicações na literatura de que usar resolvedores PMaxSat para resolver instâncias convertidas não resulta em tempos competitivos com algoritmos de branch–and–bound para CM, mas faltam dados experimentais para suportar tal afirmação. Considerando que este é um ramo de pesquisa recente e bastante aquecido, investigamos o potencial do uso da técnicas de resolvedores PMaxSat utilizando dois resolvedores baseados em branch–and–bound. Resultados experimentais comparando estes resolvedores com um algoritmo de branch–and–bound para CM estão apresentados na Seção 5.5.

5.1 DEFINIÇÕES

Seja $V = \{x_1, x_2, \ldots, x_n\}$ um conjunto finito de *variáveis*. Um *literal* de V é uma variável x_i ou sua negação $\overline{x_i}$, para algum $x_i \in V$. Uma *valoração* para V é um conjunto A de literais em V, de modo que ou $x_i \in A$ ou $\overline{x_i} \in A$ para todo $x_i \in V$. Uma *cláusula* em V é um conjunto de literais em V. Uma cláusula é *satisfeita* por uma valoração se (pelo menos) um de seus literais está na valoração. Uma *fórmula* é um conjunto de cláusulas. O problema Satisfatibilidade Máxima Parcial (PMaxSat) é o problema de, dada uma tripla (V, S, H) onde V é um conjunto de variáveis e S e H são fórmulas em V, encontrar uma valoração para V que satisfaz todas as cláusulas em H e o número máximo de cláusulas em S.

As definições podem ser convertidas para a notação do cálculo proposicional. Neste caso, as cláusulas são disjunções de literais e uma fórmula em *forma normal conjuntiva* é uma conjunção de cláusulas. As cláusulas referentes ao conjunto S de instância PMaxSat são chamadas de cláusulas *soft*, enquanto as pertencentes ao conjunto H são cláusulas *hard*.

5.2 REDUÇÕES DE CM PARA PMAXSAT

Duas reduções de CM para PMaxSat são apresentadas em (LI; QUAN, 2010b), uma tradicional e uma nova, que será chamada de redução LQ. Em ambas as reduções, os conjuntos V e H são iguais. Dado o grafo G que é a instância de CM, o conjunto V é formado por variáveis x_i, uma para cada vértice $v_i \in V(G)$. Uma valoração é interpretada de forma muito natural: a presença de x_i indica que o vértice v_i está na clique, enquanto que a presença de $\overline{x_i}$ indica que v_i não faz parte da clique. As cláusulas no conjunto H são

formadas de modo a impedir que vértices que não são vizinhos sejam inseridos na clique, então há uma cláusula $\{\overline{x_i}, \overline{x_j}\}$ para cada par $\{v_i, v_j\} \notin E(G)$ onde v_i e v_j são vértices de G.

As reduções diferem na construção das cláusulas em S. Na redução tradicional, cada variável é inserida isoladamente em uma cláusula, portanto $S = \{\{x_i\} \mid x_i \in V\}$. Na redução LQ, o conjunto S é criado com base na observação de que para toda coloração f de um grafo G, cada clique de G é composta por no máximo um vértice de cada cor em f. Portanto, variáveis respectivas a vértices que estão na mesma cor em f podem estar na mesma cláusula em S. Sendo assim, na redução LQ, é computada uma coloração f para G e para cada cor C em f, uma cláusula $\{x_i \mid v_i \in C\}$ é criada.

Assim, para um grafo G e uma k-coloração f de G, as reduções resultam nas seguintes instâncias para PMaxSat:

- Redução tradicional:

$$
\begin{aligned}
V &= \{x_i \mid v_i \in V(G)\},\\
S &= \{\{x_i\} \mid v_i \in V(G)\},\\
H &= \left\{\{\overline{x_i}, \overline{x_j}\} \mid \{v_i, v_j\} \in \binom{V(G)}{2} - E(G)\right\}.
\end{aligned}
$$

- Redução LQ:

$$
\begin{aligned}
V &= \{x_i \mid v_i \in V(G)\},\\
S &= \bigcup_{c \in [k]} \left\{\left\{x_i \mid v_i \in f^{-1}(c)\right\}\right\},\\
H &= \left\{\{\overline{x_i}, \overline{x_j}\} \mid \{v_i, v_j\} \in \binom{V(G)}{2} - E(G)\right\}.
\end{aligned}
$$

Por exemplo, o grafo $G = (\{1,2,3,4,5\}, \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{1,5\},\{3,5\}\})$, ilustrado na Figura 6, resulta nos seguintes conjuntos para cada uma das reduções:

- Redução tradicional:

$$
\begin{aligned}
V &= \{1,2,3,4,5\},\\
S &= \{\{1\},\{2\},\{3\},\{4\},\{5\}\},\\
H &= \{\{\overline{1},\overline{3}\},\{\overline{1},\overline{4}\},\{\overline{2},\overline{4}\},\{\overline{2},\overline{5}\}\}.
\end{aligned}
$$

- Redução LQ:

$$V = \{1, 2, 3, 4, 5\},$$
$$S = \{\{1, 4\}, \{2, 5\}, \{3\}\},$$
$$H = \{\{\overline{1}, \overline{3}\}, \{\overline{1}, \overline{4}\}, \{\overline{2}, \overline{4}\}, \{\overline{2}, \overline{5}\}\}.$$

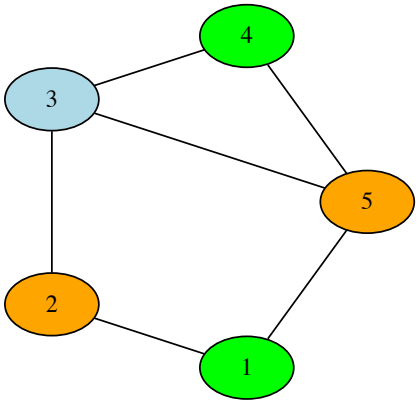


Figura 6 – Grafo colorido para exemplo das reduções PMaxSat.

5.3 O ALGORITMO MAXCLQ

A ideia de aplicação da redução LQ em Li e Quan (2010b) não é usar um resolvedor de PMaxSat para resolver o CM, mas sim permitir a adaptação de técnicas usadas em tais resolvedores para diminuir o espaço de busca em algoritmos de branch–and–bound para CM. Assim, o algoritmo MaxCLQ usa coloração em cada estado e o limitante dado por tal coloração é refinado pelo uso de uma heurística de detecção de literais falhos. A ideia pode ser sumarizada dentro do contexto do Algoritmo MaxCliqueBB da Seção 2.2 como segue.

1. Compute uma coloração f para $G[K]$.
2. Seja b o número de cores na coloração f.
3. A partir de G, K e f, compute uma certa instância (V, S, H) para PMaxSat.
4. Use a heurística de detecção de literais falhos para encontrar um conjunto não vazio $I \subseteq S$ tal que seja impossível satisfazer todas as cláusulas de I e H simultaneamente.
5. Se nenhum conjunto I com esta propriedade for encontrado, devolva b.
6. Caso contrário,

a) decremente o valor de b por 1,

b) remova o conjunto I de S,

c) volte ao passo (4).

Esta melhoria é explicada em termos de conceitos da teoria dos grafos a seguir.

Seja G um grafo e f uma coloração de G. Um conjunto X de cores de f será chamado de *folgado* com relação a f se não há clique em G com um vértice de cada cor de X. Por exemplo, qualquer conjunto de três cores em uma coloração de um circuito de cinco vértices é um conjunto folgado. Definimos uma *família folgada* como um conjunto de conjuntos folgados mutuamente disjuntos de cores de f. A ideia para a função upper-bound a ser usada no Algoritmo MaxCliqueBB é direta a partir da Proposição 4.

Proposição 4. *Seja G um grafo e f uma k-coloração de G.*

1. *Se L é um conjunto folgado de cores em f, então $\omega(G) \leq k - 1$, e*

2. *se $\mathcal{S}$ é uma família folgada, então $\omega(G) \leq k - |\mathcal{S}|$.*

upper-bound(G, K, C)

Entrada: um grafo G, um conjunto K de vértices de G e uma clique C em G.
Saída: um limitante superior de $\omega(G[K])$.

```
1 f ← uma coloração de G[K]
2 C ← o conjunto de cores em f
3 k ← |C|
4 Devolva k − tamanho-de-familia-folgada(G[K], C)
```

O Algoritmo tamanho-de-familia-folgada abaixo encontra e remove do conjunto $\mathcal{C}$ um conjunto folgado de cores de cada vez de forma gulosa. Observe que, como todo conjunto folgado encontrado é removido, todos os conjuntos encontrados são mutuamente disjuntos. Este ponto é importante porque caso contrário o limitante poderia ser decrescido erroneamente. Por exemplo, o erro aconteceria caso um conjunto folgado fosse encontrado que é um superconjunto de outro conjunto folgado encontrado previamente, visto que qualquer superconjunto de um conjunto folgado também é folgado.

O Algoritmo conjunto-folgado devolve um conjunto folgado em $\mathcal{C}$ contendo a cor X. Observe que decidir se o conjunto de todas as cores em uma k-coloração de G é folgado

tamanho-de-familia-folgada$(G, \mathcal{C})$

Entrada: um grafo G e o conjunto $\mathcal{C}$ de cores de uma coloração de G.
Saída: o tamanho de uma família folgada.

```
1 s ← 0
2 Enquanto C contém uma cor não testada faça
3 |   X ← uma cor não testada em C de tamanho mínimo
4 |   Marque X como testada
5 |   L ← conjunto-folgado(G, C, X)
6 |   Se L ≠ ∅ então
7 |   |   C ← C − L
8 |   |   s ← s + 1
9 Devolva s
```

é o mesmo que perguntar se existe uma clique de tamanho k em G, que é um problema $\mathcal{NP}$–completo.

Considere o seguinte algoritmo guloso que será usado mais adiante: a partir de um vértice u, gulosamente tente montar uma clique C com um vértice de cada cor escolhendo vértices apenas de cores onde é possível, no momento, escolher apenas um vértice. Para facilitar o entendimento do algoritmo guloso, uma correlação pode ser feita com o próprio esquema de branch–and–bound. Todo vértice que pode ser inserido em C é um candidato, isto é, os candidatos são a vizinhança comum dos vértices em C. Porém, como não há *backtracking*, só pode ser escolhido como pivô um vértice que seja único em sua cor; em outras palavras, se existem dois candidatos em uma cor, nenhum dos dois pode ser o pivô no momento. Além disso, dizemos que o algoritmo guloso *falhou* se em algum momento existir uma cor na coloração sem nenhum candidato, e neste caso, ele encerra imediatamente. Por fim, a própria clique não precisa ser guardada, basta determinar se uma clique com tamanho igual ao número de cores pode ser montada.

Agora suponha que tal algoritmo guloso falhou ao tentar produzir uma clique. Este fato, isoladamente, não é suficiente para encontrar um conjunto folgado de vértices, pois, apesar de o algoritmo não ter encontrado uma clique a partir do vértice u, pode ser que exista uma clique com um vértice de cada cor que inclua outro vértice da mesma cor de v. Então, o processo é repetido para todos os vértices v na cor X. Se o algoritmo guloso falhar para todo v em X, a cor X juntamente com as cores de todos os vértices que foram escolhidos e as cores de onde nenhum vértice for candidato formam um conjunto folgado de cores. Esse processo está descrito no Algoritmo conjunto-folgado.

conjunto-folgado$(G, \mathcal{C}, X)$

Entrada: um grafo G, um conjunto de cores $\mathcal{C}$ e uma cor X em $\mathcal{C}$.
Saída: um conjunto folgado de cores em $\mathcal{C}$ contendo X, ou $\emptyset$ caso nenhum seja encontrado.

```
1  L ← {X}
2  Para cada v ∈ X faça
3  |  T ← C
4  |  Remova X e todo u ∉ Γ_G(v) de T
5  |  Enquanto não há cor vazia e há cor unitária Y = {w} em T faça
6  |  |  Remova Y e todo u ∉ Γ_G(w) de T
7  |  Se T = ∅ então
8  |  |  Devolva ∅
9  |  Senão
10 |  |  Adicione a L as cores de C que se tornaram vazias ou unitárias em T
11 Devolva L
```

Com a definição do Algoritmo conjunto-folgado, a descrição da melhoria introduzida pelo algoritmo MaxCLQ está completa. Esta melhoria pode ser aplicada em qualquer algoritmo da família MCBB que use coloração como limitante superior. Uma simplificação do processo de busca de conjuntos folgados deu origem ao algoritmo BBMCX, estudado a seguir.

5.4 O ALGORITMO BBMCX

A busca por conjuntos folgados apresentada na seção anterior pode ser considerada custosa em um ponto de vista prático. Uma simplificação foi introduzida com o algoritmo BBMCX (SAN SEGUNDO; NIKOLAEV; BATSYN, 2015). Neste algoritmo, também é procurada uma família folgada após a coloração na tentativa de obtenção de um limitante mais apertado, então podemos considerar inicialmente que os Algoritmos upper-bound e tamanho-de-familia-folgada são análogos aos definidos na seção anterior. A diferença fundamental entre os dois algoritmos está na forma como conjuntos folgados são encontrados. O Algoritmo conjunto-folgado é capaz de encontrar conjuntos de qualquer tamanho, já no algoritmo BBMCX apenas conjuntos folgados de tamanho igual a três são procurados.

O número três foi escolhido por ser o menor tamanho possível para um conjunto folgado. Isto ocorre pois o uso de uma coloração gulosa garante que para cada par de cores, existe pelo menos um par de vértices vizinhos.

Para procurar um conjunto folgado de tamanho igual a três, o seguinte algoritmo é

usado pelo algoritmo BBMCX: a partir de um vértice v, procure duas cores c_1 e c_2 menores que a cor de v, tais que só há um vizinho w de v em c_1 e não há nenhum vértice em c_2 que é vizinho de v e w. Uma forma de interpretar este algoritmo é que ele tenta formar triângulos $\{v, w, u\}$, na esperança de mostrar que tal u não existe para alguma cor. A parte importante é que w deve ser o único vizinho de v em sua cor, lembrando que é certo que existe pelo menos um vizinho de v em cada cor com valor menor que o seu. De fato, o algoritmo tenta achar um u para todo possível w, caso exista mais de uma cor onde apenas um vértice é vizinho de v.

Considere agora o seguinte: se o vértice v que é usado como ponto de partida para o algoritmo descrito estiver em uma cor unitária $\{v\}$ e o algoritmo encontrou duas cores c_1 e c_2 como desejado, o conjunto de cores $\{\{v\}, c_1, c_2\}$ é um conjunto folgado, isto é, não há clique de tamanho três com os vértices destas cores. Isto ocorre porque o único vizinho de v em c_1 é w e não há vértice em c_2 que forme um triângulo com v e w. Esse fato possui algumas implicações bastante importantes que são fundamentais para o algoritmo BBMCX. Em primeiro lugar, se a coloração já for apertada o suficiente para que um estado seja cortado, não é necessário buscar conjuntos folgados. Com base nisto, são definidos os vértices a partir dos quais os conjuntos folgados serão procurados. Ao invés de colorir um vértice de cada vez e buscar conjuntos folgados após a coloração estar computada por completo, o algoritmo BBMCX usa uma coloração que preenche uma cor de cada vez por completo, incluindo gulosamente tantos vértices quanto possível. Além disso, o algoritmo BBMCX é derivado do algoritmo MCLIQ no sentido de que o pivô é escolhido na cor de maior valor e, assim, a coloração pode ser reaproveitada no filho direito.

Se um vértice v for colorido com uma cor de valor baixo o suficiente para que não seja pivô no estado atual nem em um estado que é descendente mais à direita do atual, não é necessário buscar conjuntos folgados a partir de v. Assim, considerando o Algoritmo MaxCliqueBB, o limitante superior é calculado como $|Q| + \mathsf{upper\text{-}bound}(G, K, C)$ e um estado (Q, K) é cortado se esse valor for menor ou igual que o tamanho da maior clique já encontrada C. Seja $k = |Q| - |C|$, os estados (Q, K') onde $K' \subseteq K$ e todos os vértices estão em uma cor com valor menor ou igual a k serão cortados. Usando uma coloração que preenche uma cor de cada vez, não é necessário que conjuntos folgados sejam procurados em cores com valor no máximo k.

O algoritmo BBMCX usa uma coloração que é uma variação do Algoritmo Guloso, e possui a mesma característica de que cada vértice em dada cor tem vizinhos em todas

as cores menores que a sua. Esta variação será chamada de GulosoPorCor. Na coloração GulosoPorCor, uma cor é preenchida de cada vez. Após a coloração das $k = |Q| - |C|$ primeiras cores, o algoritmo BBMCX cria, temporariamente, uma cor $\{v\}$ para cada vértice v ainda não colorido e os conjuntos folgados são procurados apenas a partir destes vértices. Caso um conjunto folgado $\{\{v\}, c_1, c_2\}$ seja encontrado, o limitante pode ser diminuído de um e, se a cor de v é $k+1$, a maior clique em $G[K]$ tem no máximo k vértices. Neste caso, o vértice v também não será pivô nos estados (Q, K') onde $K' \subseteq K$ e, assim, a cor de v é fixada com um valor menor ou igual a k. Caso contrário, isto é, não foi encontrado conjunto folgado a partir de v, o vértice v é colorido normalmente. Por fim, se o valor da cor temporária $\{v\}$ é maior que $k+1$, o mesmo raciocínio pode ser aplicado, pois basta incrementar um no valor de cada uma das cores maiores que k e substituir o valor de $\{v\}$ por $k+1$.

O procedimento de coloração do algoritmo BBMCX está apresentado no Algoritmo GulosoPorCor. A busca por conjuntos folgados que permite a geração de um limitante infra-cromático está apresentada no Algoritmo InfraCromático. Os algoritmos utilizam um conjunto F que guarda os valores das cores que fazem parte de um conjunto folgado, de modo a garantir que os conjuntos folgados sejam disjuntos. O laço de busca pela cor c_2 é desenrolado em dois laços pois, segundo os autores, nesta ordem as cores mais prováveis de formarem um conjunto folgado são examinadas primeiro. Um último detalhe é que no caso em que é encontrado um conjunto folgado a partir de vértice v, tal vértice é adicionado em uma cor "fictícia" de valor zero, então, rigorosamente falando, os conjuntos rotulados não representam uma coloração, e serão chamados de *pseudocoloração*.

5.5 USO DE RESOLVEDORES PMAXSAT

Segundo Li e Quan (2010b), o uso direto de resolvedores PMaxSat para solução de CM não resulta em tempos de execução competitivos com os de algoritmos de branch–and–bound para CM. Não foram encontrados, porém, dados que suportem tal afirmação. Nesta seção, o uso de resolvedores é investigado experimentalmente e dois resolvedores contemporâneos são comparados com o algoritmo MaxCliqueDyn.

O problema PMaxSat admite um esquema de branch–and–bound muito semelhante ao do CM, onde valorações são enumeradas via *backtracking*, amplamente chamado de

GulosoPorCor(G, K, k)

Entrada: um grafo G, um conjunto K de vértices de G e um inteiro k.
Saída: uma pseudocoloração de $G[K]$.

```
1  cor ← 1
2  P[0] ← ∅
3  F ← ∅
4  Enquanto há vértice em G ainda não colorido faça
5  |  P[cor] ← ∅
6  |  Para todo v ∈ V(G) ainda não colorido faça
7  |  |  Se P[cor] ∩ Γ(v) = ∅ então
8  |  |  |  Se cor ≥ k e InfraCromático(G, v, F) então
9  |  |  |  |  P[0] ← P[0] ∪ {v}
10 |  |  |  Senão
11 |  |  |  |  P[cor] ← P[cor] ∪ {v}
12 |  cor ← cor + 1
13 Devolva P
```

InfraCromático(G, v, P, F, k)

Entrada: um grafo G, um vértice v de G, um conjunto de cores P, um conjunto de inteiros F e um inteiro k.
Saída: Verdadeiro, caso encontre um conjunto folgado a partir de v; Falso, caso contrário.

```
1  Para c1 ← 1 até k faça
2  |  Se c1 ∉ F e só existe um vértice w em P[c1] que é vizinho de v então
3  |  |  Para c2 ← c1 + 1 até k faça
4  |  |  |  Se c2 ∉ F e P[c2] ∩ Γ_G(v) ∩ Γ_G(w) = ∅ então
5  |  |  |  |  F ← F ∪ {c1} ∪ {c2}
6  |  |  |  |  Devolva Verdadeiro
7  |  |  Para c2 ← c1 até c1 − 1 faça
8  |  |  |  Se c2 ∉ F e P[c2] ∩ Γ_G(v) ∩ Γ_G(w) = ∅ então
9  |  |  |  |  F ← F ∪ {c1} ∪ {c2}
10 |  |  |  |  Devolva Verdadeiro
11 Devolva Falso
```

algoritmo ou procedimento DPLL. Originalmente, o algoritmo DPLL foi desenvolvido para o problema Sat. Seu nome é uma alusão aos autores originais (DAVIS; PUTNAM, 1960; DAVIS; LOGEMANN; LOVELAND, 1962).

Em uma instância (V, S, H), para cada variável $x \in V$, uma valoração deve incluir x ou $\neg x$. Assim, um algoritmo gera uma árvore de estados em que os estados incluem uma valoração parcial Q, isto é, onde nem todas as variáveis estão presentes, e um conjunto de candidatos K com os literais referentes às variáveis cujo valor ainda não foi definido. A Figura 7 ilustra o esquema de branch–and–bound para PMaxSat. Este esquema está

bastante simplificado, mas é suficiente para mostrar o paralelo entre os dois problemas. Uma apresentação detalhada sobre o algoritmo DPLL e várias melhorias está disponível em Heras, Larrosa e Oliveras (2008).

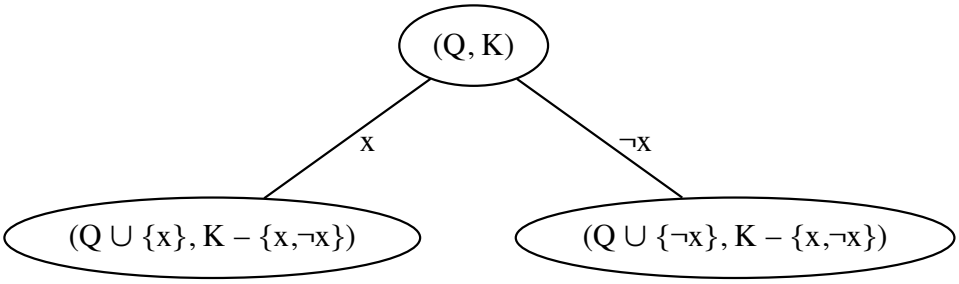


Figura 7 – Esquema de branch–and–bound para PMaxSat. O estado (Q, K) possui dois filhos. Os rótulos "x" e "$\neg x$" indicam o valor inserido na valoração em cada filho.

Para investigar o potencial do uso de técnica de resolvedores PMaxSat, foram testados dois resolvedores e comparados com um algoritmo de branch–and–bound para o CM. A intuição é que se existir um resolvedor PMaxSat que gera árvores menores que um algoritmo para CM, este algoritmo pode ser melhorado adaptando técnicas do resolvedor.

Para os testes experimentais, foram escolhidos dois resolvedores cujas implementações estavam disponíveis, MiniMaxSat (HERAS; LARROSA; OLIVERAS, 2008)[8] e ahmaxsat (ABRAME; HABET, 2015)[9]. Do lado dos algoritmos de branch–and–bound para CM, o algoritmo MaxCliqueDyn (KONC; JANEŽIČ, 2007a) foi escolhido por ser um dos melhores algoritmos que precedem o início do uso das técnicas de PMaxSat e por sua implementação estar disponível[10], também em linguagem compilada.

Foram utilizadas as instâncias do DIMACS Second Implementation Challenge (JOHNSON; TRICK, 1996) e também grafos aleatórios no modelo $\mathcal{G}_{n,p}$. As instâncias foram convertidas para fórmulas usando a redução tradicional e também a redução LQ. Na redução tradicional, a fórmula é inequívoca, porém na redução LQ, a fórmula varia dependendo do resultado da coloração. Para estes testes, a redução foi computada usando o Algoritmo Guloso.

Os testes foram realizados em uma máquina GNU/Linux com 32 núcleos operando a 2.4 GHz e 128 GB de memória. Foi estabelecido um limite de tempo de execução de três

[8] o arquivo executável estava originalmente disponível em <http://www.maxsat.udl.cat/08/index.php?disp=submitted-solvers>, porém o endereço deixou de funcionar posteriormente.
[9] disponível em <http://www.lsis.org/habetd/Djamal_Habet/MaxSAT.html>.
[10] ver <http://insilab.org/maxclique/>.

horas.

Além destes testes, os resolvedores foram testados com instâncias geradas a partir de grafos aleatórios dentro da metodologia para comparação experimental introduzida no Capítulo 6. Os resultados destes testes são encontrados na Seção 6.4.

5.5.1 Resultados e discussão

Nos experimentos foram coletados duas informações para cada algoritmo: o tamanho da árvore de estados e o tempo decorrido. Nos resolvedores PMaxSat, não é permitido acesso ao código–fonte, então eles foram tratados como caixas pretas. A informação que está sendo tratada como tamanho da árvore é o número de decisões tomadas, isto é, quantas vezes um literal foi incluído em uma valoração parcial.

A Tabela 8 apresenta os dados do algoritmo MiniMaxSat, juntamente com os dados do algoritmo MaxCliqueDyn para comparação. A Tabela 9 apresenta os dados do algoritmo ahmaxsat, novamente com os dados do algoritmo MaxCliqueDyn para comparação. As colunas $|T|$ apresentam o tamanho da árvore de estados e as colunas Tempo apresentam o tempo de execução em segundos com duas casas decimais. As instâncias exibidas são aquelas para as quais os resolvedores encerraram a execução dentro do tempo permitido. Os resolvedores foram testados usando as duas reduções de CM para PMaxSat, sendo que os resultados para a redução que adota coloração estão presentes nas colunas indicadas com "LQ" no cabeçalho.

Analisando os dados obtidos fica evidente que o tempo de execução dos resolvedores PMaxSat realmente não é competitivo, visto que o algoritmo MaxCliqueDyn é mais rápido em quase todas as execuções. As únicas exceções foram a instância hamming10-2, que foi resolvida pelo algoritmo MiniMaxSat em menor tempo que o algoritmo MaxCliqueDyn, e a instância san200_0.9_2, que foi resolvida em menor tempo pelo algoritmo ahmaxsat na redução tradicional.

As árvores criadas pelo algoritmo MiniMaxSat são maiores que as criadas pelo algoritmo MaxCliqueDyn. Já o algoritmo ahmaxsat, que é mais recente que os outros dois, gerou árvores menores que o MaxCliqueDyn, o que atesta o potencial deste ramo de pesquisa. A intuição é que enquanto houver um resolvedor PMaxSat que gera árvores menores que um algoritmo para o CM, este algoritmo pode ser melhorado adaptando técnicas do resolvedor.

Tabela 8 – Resultados experimentais de MiniMaxSat para grafos DIMACS.

G	MaxCliqueDyn $\lvert T_{\mathsf{MCDyn}}(G)\rvert$	MaxCliqueDyn Tempo (s)	MiniMaxSat $\lvert T\rvert$	MiniMaxSat Tempo (s)	MiniMaxSat (LQ) $\lvert T\rvert$	MiniMaxSat (LQ) Tempo (s)
MANN_a9	99	0.00	390	0.00	182	0.02
brock200_1	235 172	0.80	2 783 359	17.75	2 794 161	16.89
brock200_2	3622	0.01	58 747	0.41	67 982	0.44
brock200_3	12 706	0.05	196 971	1.41	169 383	1.16
brock200_4	46 755	0.13	581 646	3.78	456 447	2.86
c-fat200-1	212	0.00	402	0.71	229	0.05
c-fat200-2	238	0.00	407	0.42	325	0.05
c-fat200-5	307	0.00	391	0.04	391	0.05
c-fat500-1	513	0.00	1024	0.38	267	0.40
c-fat500-5	618	0.00	969	0.35	415	0.32
hamming6-2	62	0.00	65	0.00	33	0.03
hamming6-4	123	0.00	826	0.69	629	0.03
hamming8-2	371	0.01	613	0.03	129	0.40
hamming8-4	12 704	0.05	2 207 385	14.06	1 065 844	6.40
hamming10-2	2863	2.60	8423	0.42	513	0.06
johnson8-2-4	50	0.00	182	0.02	213	0.02
johnson8-4-4	216	0.00	2920	0.03	3902	0.02
johnson16-2-4	583 334	0.54	5 021 662	10.72	8 733 582	16.28
keller4	7711	0.02	763 074	2.90	279 042	2.76
p_hat300-1	2129	0.01	22 576	0.30	18 631	0.24
p_hat300-2	7459	0.03	77 678	0.96	55 125	0.73
p_hat300-3	629 203	3.81	5 143 557	50.28	4 106 223	39.45
p_hat500-1	10 956	0.03	167 942	3.67	154 397	3.15
p_hat500-2	193 480	1.25	1 036 800	27.25	887 731	22.60
san200_0.7_1	1138	0.01	529 637 137	1572.80	410 473 000	1174.73
san200_0.9_1	11 031	0.08	84 871	0.91	240 361	1.43
san200_0.9_2	49 351	0.44	2 400 718	22.81	671 891	5.97
san200_0.9_3	332 964	2.63	9 116 566	104.40	482 754	5.81
san400_0.9_1	628 188	19.90	601 647 032	2781.66	84 159 929	713.46
sanr200_0.7	106 744	0.32	1 269 259	12.05	1 198 260	11.71
sanr200_0.9	6 361 069	41.07	49 717 078	491.60	41 300 741	376.94
sanr400_0.5	242 343	0.68	4 100 323	77.55	3 902 503	61.51
Soma	9 483 731	74.48	1 216 050 993	5200.36	561 225 215	2465.97

Tabela 9 – Resultados experimentais de ahmaxsat para grafos DIMACS.

G	MaxCliqueDyn $\lvert T_{\mathsf{MCDyn}}(G)\rvert$	MaxCliqueDyn Tempo (s)	ahmaxsat $\lvert T\rvert$	ahmaxsat Tempo (s)	ahmaxsat (LQ) $\lvert T\rvert$	ahmaxsat (LQ) Tempo (s)
MANN_a9	99	0.00	93	0.01	92	0.00
brock200_1	235 172	0.80	145 224	58.75	122 104	46.06
brock200_2	3622	0.01	9506	2.85	7216	2.02
brock200_3	12 706	0.05	24 190	7.42	25 025	6.82
brock200_4	46 755	0.13	59 145	18.27	29 683	9.55
c-fat200-1	212	0.00	203	1.13	209	0.35
c-fat200-2	238	0.00	214	0.92	220	0.33
c-fat200-5	307	0.00	198	0.43	143	0.30
c-fat500-1	513	0.00	559	16.94	624	45.27
c-fat500-5	618	0.00	515	13.02	561	9.18
hamming6-2	62	0.00	36	0.00	36	0.03
hamming6-4	123	0.00	174	0.70	133	0.03
hamming8-2	371	0.01	203	0.49	180	0.83
hamming8-4	12 704	0.05	65 269	26.64	29 698	12.74
hamming10-2	2863	2.60	1188	61.11	1287	61.61
johnson8-2-4	50	0.00	49	0.00	39	0.00
johnson8-4-4	216	0.00	162	0.03	132	0.01
johnson16-2-4	583 334	0.54	400 506	16.70	206 492	6.59
keller4	7711	0.02	34 695	5.24	13 536	3.51
p_hat300-1	2129	0.01	5169	7.23	3185	5.91
p_hat300-2	7459	0.03	17 797	21.27	5700	6.16
p_hat300-3	629 203	3.81	343 461	467.91	508 717	580.55
p_hat500-1	10 956	0.03	36 577	58.47	23 753	39.74
p_hat500-2	193 480	1.25	496 326	896.25	153 383	224.98
san200_0.7_1	1138	0.01	2193	2.06	1224	1.15
san200_0.9_1	11 031	0.08	383	0.25	419	0.17
san200_0.9_2	49 351	0.44	307	0.41	1042	1.48
san200_0.9_3	332 964	2.63	62 251	83.24	13 231	17.35
san400_0.9_1	628 188	19.90	327 389	909.62	65 113	209.90
sanr200_0.7	106 744	0.32	77 541	47.00	59 213	34.48
sanr200_0.9	6 361 069	41.07	143 283	232.76	138 638	198.53
sanr400_0.5	242 343	0.68	373 764	274.76	346 190	256.11
Soma	9 483 731	74.48	2 628 570	3231.88	1 757 218	1781.74

6 METODOLOGIA PARA COMPARAÇÃO EXPERIMENTAL DE ALGORITMOS

De um modo geral, novos algoritmos para solução exata de CM são apresentados na literatura acompanhados de resultados experimentais. Tipicamente, são apresentados os tempos de execução para um conjunto de instâncias para o algoritmo proposto. A diferença de ambientes de programação e de execução faz com que as conclusões possíveis ao comparar tais resultados sejam limitadas.

Na maioria dos algoritmos propostos na literatura, os autores não disponibilizam a implementação de forma livre. Para reproduzir os experimentos, é necessário que os algoritmos sejam implementados novamente de forma independente. Então, diferenças nos resultados podem surgir na programação do algoritmo. Dentre os algoritmos MCBB citados na Seção 2.2, a exceção é o algoritmo MaxCliqueDyn (KONC; JANEŽIČ, 2007a), cuja implementação em linguagem C está disponível[11].

Dadas estas dificuldades em reproduzir os experimentos, a alternativa imediata para comparar diferentes algoritmos é utilizar os próprios dados publicados. Para tal, uma técnica de redimensionamento de dados tem sido utilizada, baseada na ideia de que a diferença no tempo de execução de um mesmo programa em computadores diferentes deveria ser uma constante multiplicativa. A técnica consiste em compilar e executar o programa `dfmax`[12] para um conjunto padrão de instâncias e, a partir dos tempos registrados, calcular um fator de correção para cada conjunto de dados já publicado. Em seguida, os dados são multiplicados pelo fator apropriado e reapresentados. Com isto, os dados estariam na mesma escala em que os resultados seriam obtidos caso os experimentos tivessem sido reproduzidos. Como exemplo, esta técnica foi adotada na apresentação dos algoritmos MCR (TOMITA; KAMEDA, 2007) e MaxCLQ (LI; QUAN, 2010b). Entretanto, recentemente esta técnica foi testada e analisada em um trabalho onde vários algoritmos foram implementados na linguagem Java[13] (PROSSER, 2012). Os resultados mostram que os dados obtidos pelo redimensionamento podem apresentar um grande erro em comparação com dados obtidos quando os experimentos são realizados de fato em ambientes diferentes. Assim, conclusões errôneas podem ser tiradas e o uso de tal técnica não é recomendado.

[11] ver <http://insilab.org/maxclique/>.
[12] disponível em <ftp://dimacs.rutgers.edu/pub/dsj/clique/dfmax.c>.
[13] implementações disponíveis em <http://www.dcs.gla.ac.uk/~pat/maxClique/>.

Neste capítulo é apresentada uma metodologia para comparação experimental de algoritmos de branch–and–bound para solução de CM, que é embasada nos resultados apresentados na Seção 4.1.3. A metodologia proposta não é sensível à implementação, de modo que para um novo algoritmo, basta a execução de novos experimentos e os resultados podem ser comparados diretamente com resultados anteriores.

Na Seção 4.1.3.2, foi visto que o tamanho da árvore de estados do algoritmo nobound para grafos aleatórios no $\mathcal{G}_{n,1/2}$ é $n^{\frac{1}{2}\lg n(1+o(1))}$ e, na Seção 4.1.3.1, que a razão $\frac{\log_{1/p}|T_{\mathcal{A}}(G)|}{(\log_{1/p} n)^2}$ para um grafo G em $\mathcal{G}_{n,p}$ tem um comportamento bastante específico para os algoritmos f–driven e nobound, que enumeram todas as cliques do grafo de entrada. A questão que norteia a metodologia proposta é: qual o comportamento desta mesma razão para cada um dos algoritmos de branch–and–bound estudados?

O valor de p será fixado em $1/2$, o que significa que todos os grafos com um determinado n têm a mesma probabilidade de ser escolhidos. Assim, os grafos a serem escolhidos formam o espaço de probabilidades uniforme $\mathcal{G}_{n,1/2}$.

A *pontuação* para um grafo G em $\mathcal{G}_{n,1/2}$ do algoritmo $\mathcal{A}$ é dada por

$$R_{\mathcal{A}}(G) := \frac{\lg|T_{\mathcal{A}}(G)|}{(\lg n)^2}.$$

Um experimento consiste em escolher um grafo G em $\mathcal{G}_{n,1/2}$, executar cada algoritmo $\mathcal{A}$ de interesse em G e calcular a pontuação $R_{\mathcal{A}}(G)$ para cada $\mathcal{A}$. A ideia é computar a média de $R_{\mathcal{A}}$ para uma amostra de tamanho apropriado de grafos com n vértices gerados aleatoriamente de maneira uniforme, ou seja, grafos em $\mathcal{G}_{n,1/2}$, e usar este valor como uma medida para o desempenho do algoritmo $\mathcal{A}$. Assim, a metodologia consiste em assumir como hipótese de trabalho que $\mathbb{E}[|T_{\mathcal{A}}|] = O(n^{c_{\mathcal{A}}\lg n})$ em $\mathcal{G}_{n,1/2}$ e usar o valor de $R_{\mathcal{A}}$ como uma estimativa para o valor de $c_{\mathcal{A}}$.

Em termos de desempenho de um determinado algoritmo, estamos diretamente testando a efetividade das técnicas adotadas para redução do espaço de busca. Um valor menor de $R_{\mathcal{A}}$ não implica em um tempo necessariamente menor, devido ao tempo gasto no cálculo do limitante superior.

6.1 ALGORITMOS TESTADOS

A metodologia proposta foi adotada para comparação dos algoritmos nobound, order–driven (CHVÁTAL, 1977), MaxCLQ (LI; QUAN, 2010b; ZÜGE; CARMO, 2016), BBMCX (SAN SEGUNDO; NIKOLAEV; BATSYN, 2015), e cada um dos algoritmos

detalhados em Carmo e Züge (2012): Basic; CP (CARRAGHAN; PARDALOS, 1990); DF, χ e $\chi + \mathsf{DF}$ (FAHLE, 2002); MCQ (TOMITA; SEKI, 2003); MaxCliqueDyn (KONC; JANEŽIČ, 2007a); MCR (TOMITA; KAMEDA, 2007); MCS (TOMITA et al., 2010).

Todos os algoritmos escolhidos satisfazem a hipótese do Teorema 21, isto é, para todo algoritmo $\mathcal{A}$ as funções $g_{\mathcal{A}}(G)$ e $f^*_{\mathcal{A}}(G)$ são polinomiais em $|V(G)|$. Mais precisamente, temos $f^*_{\mathcal{A}}(G) = O(|V(G)|^2)$ para todos os algoritmos exceto MCS, para o qual $f^*_{\mathsf{MCS}}(G) = O(|V(G)|^3)$.

Os algoritmos nobound, Basic, CP, DF, χ, $\chi + \mathsf{DF}$, MCQ, MCR, MaxCliqueDyn e MCS haviam sido implementados em Züge (2011) utilizando a linguagem Python (LUTZ, 2010) e o pacote NetworkX[14] para manipulação de redes e grafos e disponibilizados em um repositório na internet[15].

Esta implementação foi atualizada para utilizar as bibliotecas disponibilizados pelo ambiente SageMath e ser compilado com Cython (BEHNEL et al., 2011)[16]. O Cython é um compilador para linguagem Python e uma linguagem própria também chamada de Cython, que permite chamadas diretas a funções escritas na linguagem C. A implementação atualizada está disponível em um novo repositório[17].

Como o algoritmo order–driven não se encaixa no esquema dos algoritmos MCBB, uma nova implementação seria necessária. Foi desenvolvida uma adaptação do algoritmo MaxCliqueBB que gera todos os subproblemas como no algoritmo order–driven. As etapas de memoização e construção da solução final não foram implementadas, por serem desnecessárias para o propósito de contagem do número de estados.

Os algoritmos MaxCLQ e BBMCX foram implementados com base nas descrições apresentadas no Capítulo 5.

Os testes foram realizados em um computador com processador Intel Xeon E5-1650, com seis núcleos a 3.5 GHz, e 32 GB de memória, rodando a distribuição Ubuntu 16.04 do sistema operacional GNU/Linux. A execução foi automatizada utilizando o programa GNU Parallel (TANGE, 2011).

[14] ver <http://networkx.lanl.gov/>.
[15] a saber, <http://subversion.assembla.com/svn/alglabmc/>.
[16] ver <http://cython.org>
[17] a saber, <https://gitlab.c3sl.ufpr.br/apzuge/maxcliquebb>.

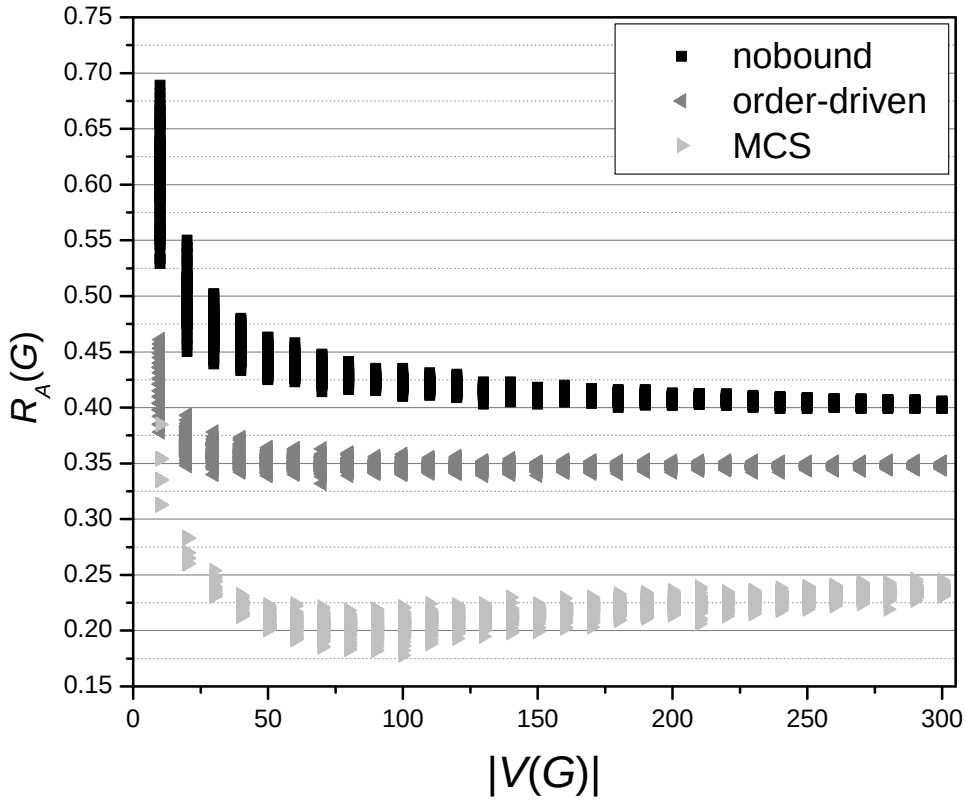


Figura 8 – Valores de $R_{\mathcal{A}}(G)$ para os algoritmos nobound, order–driven, Basic, CP e DF e $G \in \bigcup_{n\in\{10,20,\ldots,300\}} \mathcal{P}_n$.

6.2 DEFINIÇÃO DE PARÂMETROS

Para decidir o número n de vértices nas instâncias a serem consideradas, foram executados testes preliminares usando grafos aleatórios para diversos valores de n. Para cada $n \in \{10, 20, \ldots, 300\}$, foi criado um conjunto $\mathcal{P}_n$ contendo 100 grafos aleatórios em $\mathcal{G}_{n,1/2}$. Para cada grafo $G \in \bigcup_{n\in\{100,\ldots,300\}} \mathcal{P}_n$, o tamanho da árvore $T_A(G)$ e o valor de $R_{\mathcal{A}}(G)$ foram computados.

Como poderia ser esperado, a variância no valor de $R_{\mathcal{A}}(G)$ decresce conforme o valor de $|V(G)|$ aumenta. Assim, com base nos valores coletados nestes experimentos preliminares, decidiu-se fixar o valor mínimo de n em 100. O valor máximo de n foi determinado com base em limitações do nosso ambiente de execução, e pode variar dependendo de quais algoritmos estão sendo comparados. Para exemplificar os resultados desta etapa, os valores de $R_{\mathcal{A}}(G)$ para $\mathcal{A} \in \{\text{nobound}, \text{order–driven}, \text{MCS}\}$ são apresentados na Figura 8.

Tabela 10 – Valores de $R_{\mathcal{A}}(\mathcal{I})$.

Posição	$\mathcal{A}$	$R_{\mathcal{A}}(\mathcal{I})$	$\sigma_{R_{\mathcal{A}}(\mathcal{I})}$
1	MaxCLQ	0.218 852	0.011 797
2	BBMCX	0.221 413	0.009 996
3	MCS	0.222 873	0.010 820
4	MaxCliqueDyn	0.233 902	0.008 589
5	MCR	0.236 429	0.010 647
6	MCQ	0.237 847	0.009 628
7	χ + DF	0.240 133	0.010 251
8	χ	0.240 332	0.009 990
9	DF	0.279 007	0.007 480
10	CP	0.331 465	0.003 368
11	Basic	0.341 907	0.003 361
12	order–driven	0.347 358	0.002 231
13	nobound	0.409 352	0.006 330

6.3 RESULTADOS EXPERIMENTAIS E DISCUSSÃO

Para cada $n \in \{100, 110, 120, \ldots, 580\}$, um conjunto $\mathcal{I}_n$ com 100 grafos aleatórios em $\mathcal{G}_{n,1/2}$ foi criado. Para cada grafo $G \in \mathcal{I} := \bigcup_{n\in\{100,\ldots,300\}} \mathcal{I}_n$ e cada algoritmo $\mathcal{A}$, o tamanho da árvore de estados $T_{\mathcal{A}}(G)$ e o valor de $R_{\mathcal{A}}(G)$ foram computados. Por fim, para cada algoritmo $\mathcal{A}$, foi computado o valor de

$$R_{\mathcal{A}}(\mathcal{I}) := \frac{\sum_{G\in\mathcal{I}} R_{\mathcal{A}}(G)}{|\mathcal{I}|}.$$

Os valores de $R_{\mathcal{A}}(\mathcal{I})$ estão apresentados na Tabela 10, juntamente com o desvio padrão de $R_{\mathcal{A}}(G)$ para $G \in \mathcal{I}$. Os algoritmos estão ordenados pela pontuação, lembrando que valores menores indicam pontuações melhores.

As Figuras 9 e 10 mostram, para cada algoritmo $\mathcal{A}$, os valores de

$$R_{\mathcal{A}}(n) := \frac{\sum_{G\in\mathcal{I}_n} R_{\mathcal{A}}(G)}{|\mathcal{I}_n|}.$$

Estes dados podem ser interpretados como segue. A pontuação $R_{\mathcal{A}}(\mathcal{I})$ indica que o tamanho médio da árvore de estados do algoritmo $\mathcal{A}$ foi $n^{R_{\mathcal{A}}(\mathcal{I})\lg n}$, com a média sendo calculada sobre todo o conjunto $\mathcal{I}$ de instâncias. De acordo com esta métrica, o melhor algoritmo, isto é, o algoritmo mais eficiente na redução da árvore de estados, é o algoritmo MaxCLQ, que criou aproximadamente $n^{0.219\lg n}$ estados em média.

É bastante importante observar que as pontuações calculadas ordena os algoritmos na ordem cronológica de publicação destes, com a exceção dos algoritmos BBMCX e MaxCLQ.

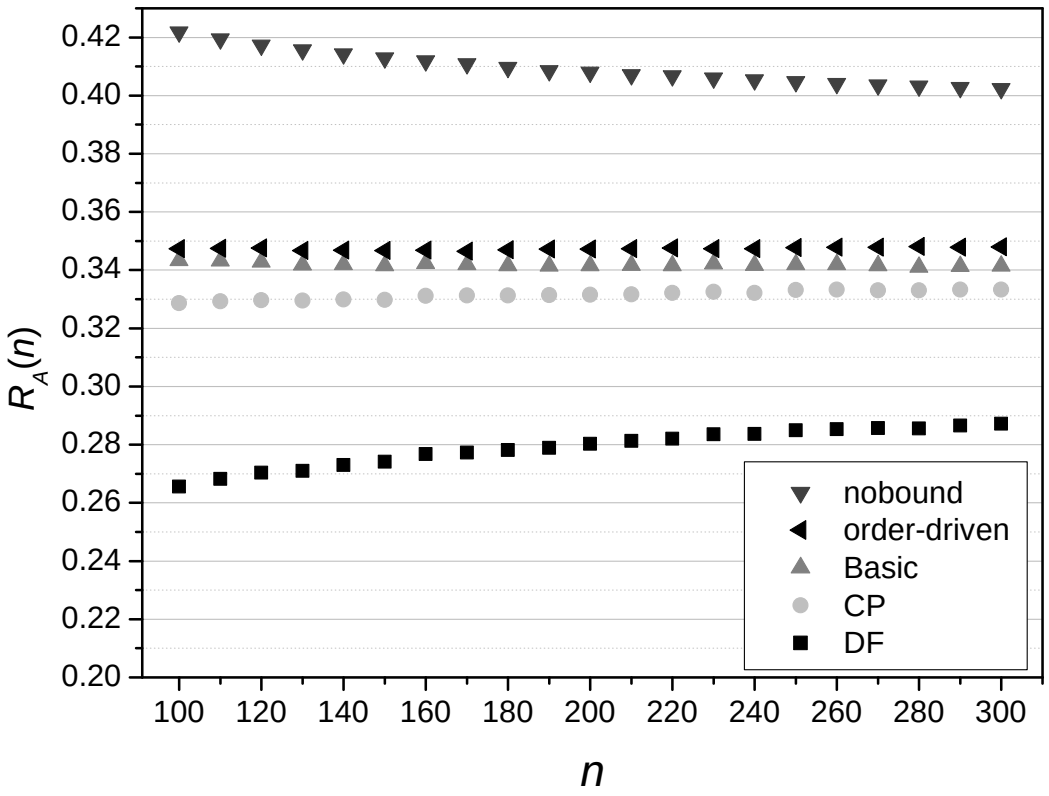


Figura 9 – Valores de $R_{\mathcal{A}}(n)$ para os algoritmos nobound, order–driven, Basic, CP e DF.

Esta inversão era esperada, visto que BBMCX é derivado de MaxCLQ, porém mais simples. Esta ordem também está em concordância com outros resultados experimentais disponíveis na literatura.

A Tabela 11 mostra o tempo de execução médio para alguns algoritmos, no conjunto $\mathcal{I}_{300}$ de instâncias. O algoritmo order–driven foi omitido porque apenas sua primeira fase, que é suficiente para calcular as pontuações $R_{\text{order–driven}}(G)$, foi implementada. Este algoritmo ilustra a conveniência da metodologia proposta: não foi necessário implementar o algoritmo em sua totalidade para avaliá-lo.

Os algoritmos χ e $\chi + \text{DF}$ apresentam tempos de execução bastante altos; isto ocorre devido ao fato de que eles computam quatro colorações diferentes na função upper-bound. O caso do algoritmo MaxCLQ é análogo. Este algoritmo aplica uma heurística custosa, adaptada de um resolvedor Sat, como visto na Seção 5.3. É interessante observar que este fato serve de motivação para o desenvolvimento do algoritmo BBMCX, onde uma versão mais simples da mesma heurística é utilizada, vista na Seção 5.4, resultando em

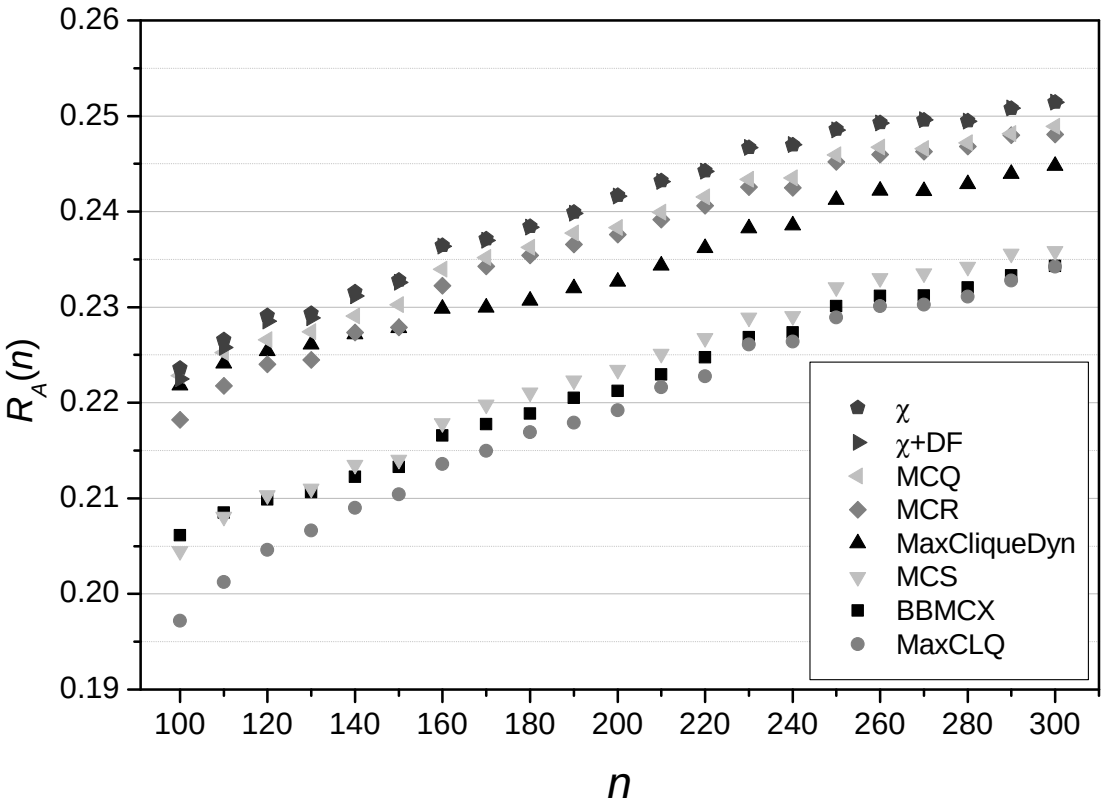


Figura 10 – Valores de $R_{\mathcal{A}}(n)$ para os algoritmos χ, $\chi + \mathsf{DF}$, MCQ, MCR, MaxCliqueDyn, MCS, BBMCX e MaxCLQ.

Tabela 11 – Tempo médio de execução e desvio padrão para $n = 300$ de alguns algoritmos, ordenados pelo posicionamento da Tabela 10.

$\mathcal{A}$	**Tempo** (s)	σ_{Tempo}
MaxCLQ	42.95	3.20
BBMCX	10.41	0.96
MCS	5.58	0.26
MaxCliqueDyn	9.21	0.67
MCR	11.96	0.77
MCQ	9.16	0.78
$\chi + \mathsf{DF}$	4072.16	462.57
χ	4777.62	548.08
DF	118.95	7.68
CP	35.36	2.21
Basic	51.56	3.45
nobound	230.58	14.80

Tabela 12 – Avaliação de dez algoritmos de acordo com uma variação da metodologia proposta com instâncias em $\mathcal{J}$.

Posição	$\mathcal{A}$	$R_{\mathcal{A}}(\mathcal{J})$	$\sigma_{R_{\mathcal{A}}(\mathcal{J})}$
1	MaxCLQ	0.225 584	0.013 929
2	BBMCX	0.227 341	0.012 119
3	MCS	0.228 944	0.012 739
4	MaxCliqueDyn	0.239 054	0.010 544
5	MCR	0.241 918	0.012 078
6	MCQ	0.242 977	0.011 114
7	DF	0.282 798	0.008 415
8	CP	0.332 573	0.003 398
9	Basic	0.342 000	0.002 999
10	nobound	0.406 519	0.006 690

um tempo mais baixo na média. De fato, os tempos de execução do algoritmo BBMCX podem ser ainda melhores em uma implementação especializada do que os apresentados aqui. Isto porque a implementação utilizada para estes testes não adota mapas de bits para representação de conjuntos, como proposto originalmente em (SAN SEGUNDO; NIKOLAEV; BATSYN, 2015). Assim como ocorreu com o algoritmo order–driven, não foi necessário implementar o algoritmo BBMCX em sua totalidade para computar os valores de $R_{\mathsf{BBMCX}}(G)$.

Outras duas avaliações foram realizadas para conjuntos maiores de instâncias, porém com menos algoritmos. A segunda avaliação foi realizada usando o conjunto de instâncias $\mathcal{J} := \bigcup_{n\in\{100,\ldots,400\}} \mathcal{I}_n$. Os valores de $R_{\mathcal{A}}(\mathcal{J})$ estão apresentados na Tabela 12 e ilustrados nas Figuras 11 e 12. Não foram computados os valores de R_{χ}, $R_{\chi+\mathsf{DF}}$ e $R_{\mathsf{order–driven}}$, pois a execução destes algoritmos se tornou proibitivamente demorada no ambiente de execução adotado. Apesar de os valores resultantes serem diferentes dos calculados utilizando o conjunto $\mathcal{I}$, é de grande importância observar que os algoritmos permanecem nas mesmas posições relativas, quando ordenados pelas pontuações.

Uma terceira avaliação foi realizada com todas as instâncias geradas, presentes no conjunto $\mathcal{K} := \bigcup_{n\in\{100,\ldots,580\}} \mathcal{I}_n$. Os valores de $R_{\mathcal{A}}(\mathcal{K})$ estão apresentados na Tabela 13 e ilustrados nas Figuras 13 e 14. Além de R_{χ}, $R_{\chi+\mathsf{DF}}$ e $R_{\mathsf{order–driven}}$, também estão omitidos os algoritmos R_{nobound} e R_{DF}, cujos tempos de execução tornam suas avaliações impraticáveis. Novamente, os algoritmos permanecem na mesma ordem que nas avaliações anteriores.

Tabela 13 – Avaliação de oito algoritmos de acordo com uma variação da metodologia proposta com instâncias em $\mathcal{K}$.

Posição	$\mathcal{A}$	$R_{\mathcal{A}}(\mathcal{K})$	$\sigma_{R_{\mathcal{A}}(\mathcal{K})}$
1	MaxCLQ	0.233 735	0.015 607
2	BBMCX	0.235 014	0.014 187
3	MCS	0.236 724	0.014 616
4	MaxCliqueDyn	0.245 354	0.011 944
5	MCR	0.249 047	0.013 645
6	MCQ	0.249 791	0.012 827
7	CP	0.334 178	0.003 588
8	Basic	0.342 387	0.002 637

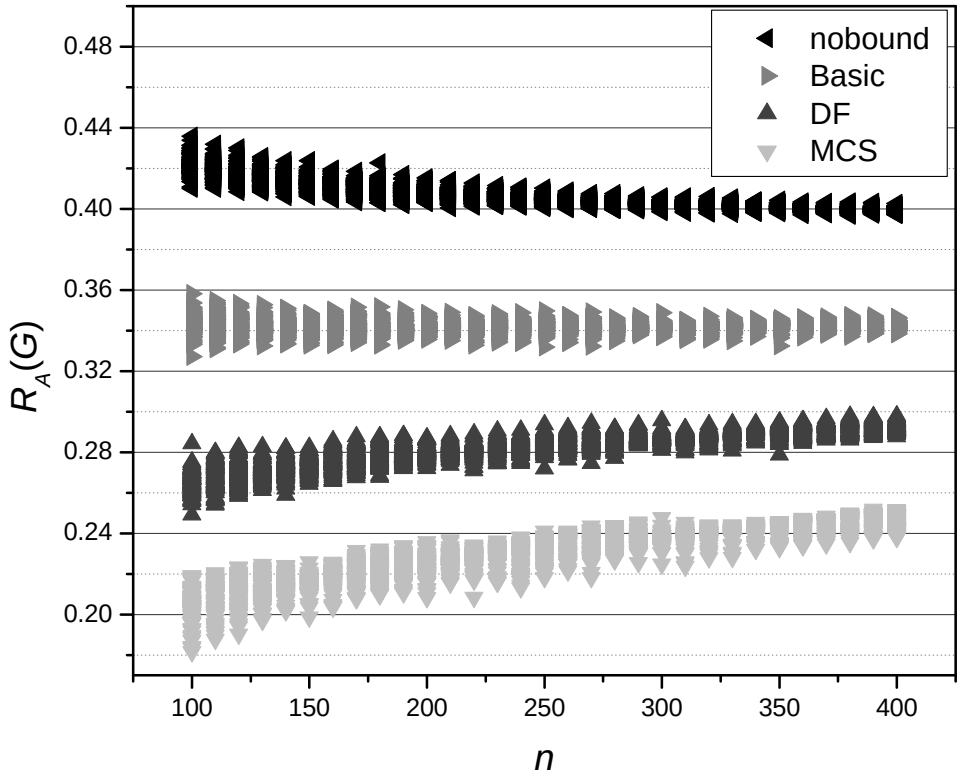


Figura 11 – Valores de $R_{\mathcal{A}}(G)$ para os algoritmos nobound, Basic, DF e MCS e $G \in \mathcal{J}$.

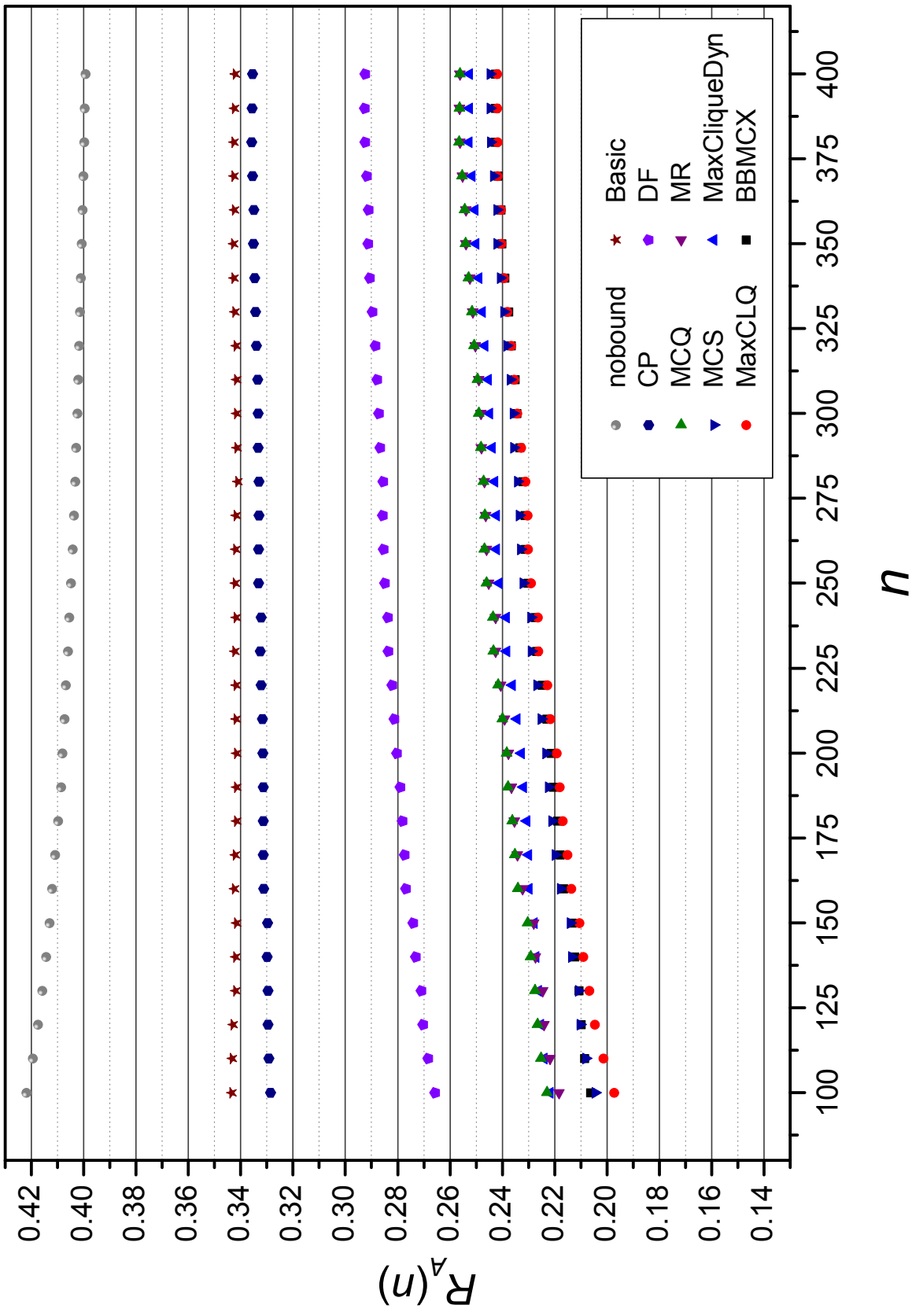


Figura 12 – Valores de $R_{\mathcal{A}}(n)$ para instâncias em $\mathcal{J}$.

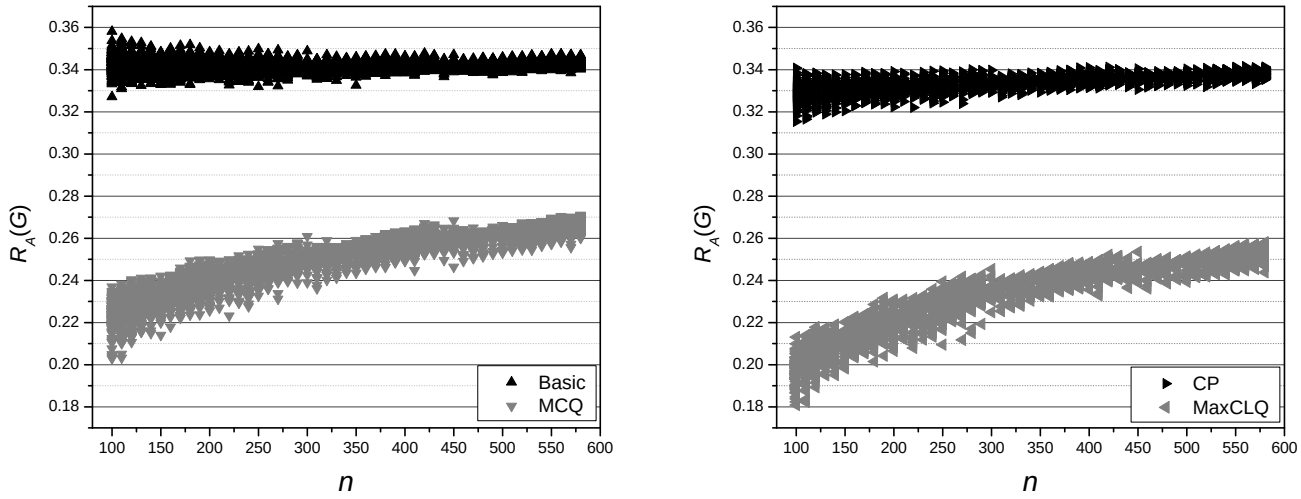


Figura 13 – Valores de $R_{\mathcal{A}}(G)$ para os algoritmos Basic, MCQ, CP e MaxCLQ e $G \in \mathcal{K}$.

6.4 APLICAÇÃO EM OUTROS ALGORITMOS

A metodologia proposta foi adotada para testar algoritmos da família MCBB e o algoritmo order–driven, por serem os algoritmos do Capítulo 4, que embasa a metodologia. Algoritmos em outros contextos também podem ser avaliados, sendo que é necessário computar os valores $R_{\mathcal{A}}(G)$. Por exemplo, para resolvedores PMaxSat, o valor é o número de decisões tomadas, como apresentado na Seção 5.5.1.

Em uma rodada adicional de experimentos, outros algoritmos apresentados neste trabalho foram testados nos grafos do conjunto $\mathcal{I}$. Os resolvedores de PMaxSat citados na Seção 5.5, MiniMaxSat e ahmaxsat, foram testados usando as duas reduções da Seção 5.2. O resultados para a redução baseada em coloração estão nas linhas indicadas com "redução LQ". Também foram testados o algoritmo MCJ, apresentado na Seção 3.3.3, e o algoritmo MCLIQ, visto na Seção 2.2.3.

Os resultados estão na Tabela 14. Dois dos algoritmos adicionados, MCJ e ahmaxsat, apresentaram bons resultados, ficando atrás apenas dos três melhores algoritmos para CM avaliados, isto é, MaxCLQ, BBMCX e MCS. O resolvedor MiniMaxSat, mais antigo que o outro, não apresentou resultados competitivos. As instâncias PMaxSat geradas com a redução LQ levaram a pontuações melhores que as da redução tradicional. O algoritmo MCLIQ ficou logo abaixo do algoritmo MCQ, como podia ser esperado.

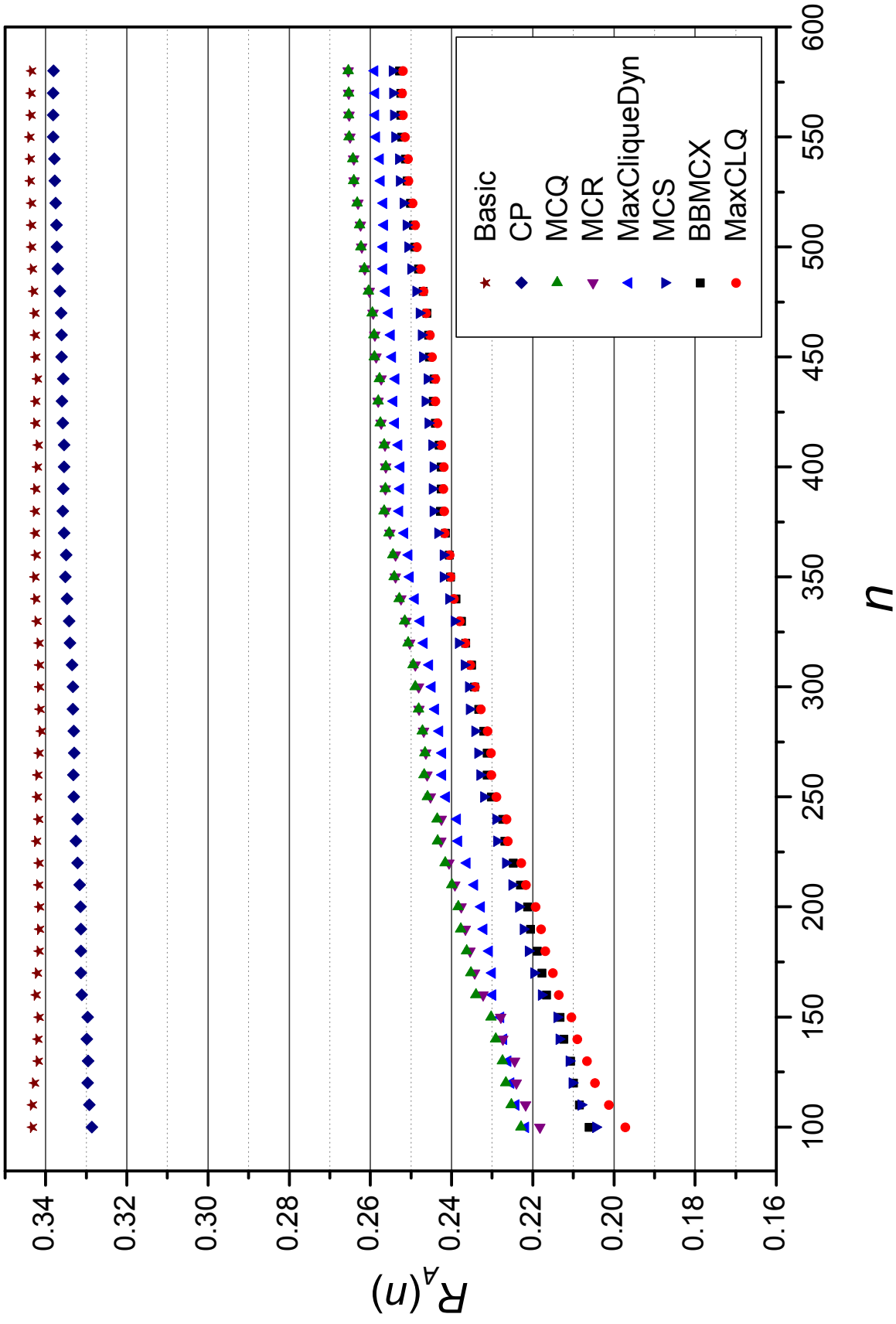


Figura 14 – Valores de $R_{\mathcal{A}}(n)$ para instâncias em $\mathcal{K}$.

Tabela 14 – Valores de $R_{\mathcal{A}}(\mathcal{I})$ para algoritmos da família MCBB, resolvedores PMaxSat, e algoritmos order–driven e MCJ.

Posição	$\mathcal{A}$	$R_{\mathcal{A}}(\mathcal{I})$	$\sigma_{R_{\mathcal{A}}(\mathcal{I})}$
1	MaxCLQ	0.218 852	0.011 797
2	BBMCX	0.221 413	0.009 996
3	MCS	0.222 873	0.010 820
4	ahmaxsat (redução LQ)	0.225 725	0.011 993
5	ahmaxsat	0.227 481	0.011 766
6	MCJ	0.228 027	0.009 712
7	MaxCliqueDyn	0.233 902	0.008 589
8	MCR	0.236 429	0.010 647
9	MCQ	0.237 847	0.009 628
10	MCLIQ	0.238 286	0.010 519
11	χ + DF	0.240 133	0.010 251
12	χ	0.240 332	0.009 990
13	MiniMaxSat (redução LQ)	0.278 206	0.008 224
14	MiniMaxSat	0.278 803	0.007 215
15	DF	0.279 007	0.007 480
16	CP	0.331 465	0.003 368
17	Basic	0.341 907	0.003 361
18	order–driven	0.347 358	0.002 231
19	nobound	0.409 352	0.006 330

7 CONCLUSÃO

O problema da Clique Máxima (CM) é um problema fundamental e existe uma grande motivação pela busca de algoritmos tão eficientes quanto possível para sua solução exata. Os melhores algoritmos para o CM com desempenho de pior caso conhecido tem custo de tempo exponencial no número de vértices do grafo.

Resultados experimentais apresentados na literatura para algoritmos de branch–and–bound indicam que instâncias de tamanho considerável podem ser resolvidas em pouco tempo. Com isso, observa-se uma distância entre os melhores resultados em trabalhos analíticos e os melhores resultados em trabalhos experimentais.

No presente trabalho, foram realizados estudos cujos resultados ajudam a entender o desempenho dos algoritmos para o CM para várias famílias de grafos. Foi mostrado que uma grande família de algoritmos de branch–and–bound (que apresentam os melhores resultados experimentais) possui custo de tempo médio subexponencial de ordem $O(n^{c \lg n})$. Este fato oferece uma explicação para a discrepância aparente entre resultados obtidos analiticamente e experimentalmente.

Com base no estudo analítico de algoritmos e instâncias, foi desenvolvida uma metodologia para análise experimental de algoritmos. Esta metodologia tem a vantagem inédita de que algoritmos podem ser comparados diretamente, independentemente de detalhes de implementação. Isto é relevante pois boa parte dos resultados experimentais obtidos usando grafos aleatórios publicados não permite comparação direta entre algoritmos, devido a diferenças de implementação e ambientes de execução. De fato, usando a metodologia proposta, não é necessário que algoritmos sejam implementados com foco em tempo de execução.

Diversos algoritmos de branch–and–bound usam coloração na tentativa de redução de seu espaço de busca. Boa parte dos algoritmos estudados no presente trabalho se enquadra neste contexto. Foi estudado e apresentado um limitante inferior para o pior caso dos algoritmos que usam apenas coloração como limitante superior. Este pior caso consiste em instâncias geradas como grafos junção que fazem com que tais algoritmos executem em tempo exponencial. Foi desenvolvido um método para detecção e decomposição de grafos junção e, a partir dele, uma família de algoritmos que é capaz de resolver as instâncias citadas em tempo polinomial.

Uma outra abordagem recente para melhorar os algoritmos que utilizam coloração é adaptar técnicas utilizadas em um outro problema $\mathcal{NP}$–difícil de maximização, o problema da Satisfatibilidade Máxima Parcial (PMaxSat). Como a ideia original é converter o grafo para uma fórmula PMaxSat e aplicar diretamente uma heurística de um resolvedor para gerar um limitante superior, não é imediato qual foi o processamento realizado em termos do próprio grafo no qual a clique máxima está sendo procurada. Neste trabalho, algumas destas técnicas foram estudadas e convertidas para uma descrição que permite o uso direto em algoritmos de branch–and–bound para CM e que possibilita a compreensão do como o algoritmo está trabalhando em relação ao grafo.

Por fim, destaca-se que diversos algoritmos foram implementados e analisados experimentalmente. A implementação dos algoritmos nobound, Basic, CP, DF, χ, $\chi + \mathsf{DF}$, MCLIQ, MCQ, MCR, MaxCliqueDyn, MCS e MCJ e das versões descritas no Capítulo 5 dos algoritmos MaxCLQ e BBMCX foi disponibilizada em um repositório de acesso público.

REFERÊNCIAS

ABRAME, A.; HABET, D. Ahmaxsat: Description and evaluation of a branch and bound max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, v. 9, p. 89–128, 2015. Citado na página 88.

ANJOS, C. S. dos. *Análise experimental de algoritmos*. Dissertação (Mestrado) — Universidade Federal do Paraná, ago. 2015. Disponível em: <http://acervodigital.ufpr.br/handle/1884/40875>. Citado na página 76.

BEHNEL, S.; BRADSHAW, R.; CITRO, C.; DALCIN, L.; SELJEBOTN, D.; SMITH, K. Cython: The best of both worlds. *Computing in Science Engineering*, v. 13, n. 2, p. 31–39, mar.-abr. 2011. ISSN 1521-9615. Citado na página 94.

BOLLOBÁS, B. *Random Graphs*. Cambridge: Cambridge University Press, 2001. (Cambridge Studies in Advanced Mathematics). ISBN 9780521797221. Citado na página 18.

BOMZE, I. M.; BUDINICH, M.; PARDALOS, P. M.; PELILLO, M. The maximum clique problem. In: DU, D.-Z.; PARDALOS, P. M. (Ed.). *Handbook of Combinatorial Optimization: Supplement Volume A*. Boston, MA: Springer, 1999. v. 4, p. 1–74. ISBN 978-1-4757-3023-4. Disponível em: <http://dx.doi.org/10.1007/978-1-4757-3023-4_1>. Citado 2 vezes nas páginas 13 e 20.

BRON, C.; KERBOSCH, J. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, ACM Press, New York, NY, USA, v. 16, n. 9, p. 575–577, set. 1973. ISSN 0001-0782. Disponível em: <http://dx.doi.org/10.1145/362342.362367>. Citado na página 19.

CARMO, R.; ZÜGE, A. Branch and bound algorithms for the maximum clique problem under a unified framework. *Journal of the Brazilian Computer Society*, Springer-Verlag, v. 18, n. 2, p. 137–151, dez. 2012. ISSN 0104-6500. Disponível em: <https://link.springer.com/article/10.1007%2Fs13173-011-0050-6?LI=true>. Citado 2 vezes nas páginas 21 e 94.

CARRAGHAN, R.; PARDALOS, P. M. An exact algorithm for the maximum clique problem. *Operations Research Letters*, v. 9, n. 6, p. 375–382, 1990. ISSN 0167-6377. Disponível em: <http://dx.doi.org/10.1016/0167-6377(90)90057-C>. Citado 2 vezes nas páginas 25 e 94.

CHVÁTAL, V. Determining the stability number of a graph. *SIAM Journal on Computing*, v. 6, n. 4, p. 643–662, dez. 1977. Disponível em: <http://dx.doi.org/10.1137/0206046>. Citado 3 vezes nas páginas 60, 62 e 93.

COOK, S. A. The complexity of theorem-proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. New York: ACM, 1971. (STOC '71), p. 151–158. Disponível em: <http://doi.acm.org/10.1145/800157.805047>. Citado na página 77.

DAVIS, M.; LOGEMANN, G.; LOVELAND, D. A machine program for theorem-proving. *Commun. ACM*, ACM, New York, v. 5, n. 7, p. 394–397, jul. 1962. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/368273.368557>. Citado na página 87.

DAVIS, M.; PUTNAM, H. A computing procedure for quantification theory. *J. ACM*, ACM, New York, v. 7, n. 3, p. 201–215, jul. 1960. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/321033.321034>. Citado na página 87.

DOWNEY, R.; FELLOWS, M. *Parameterized Complexity*. New York: Springer, 2012. (Monographs in Computer Science). ISBN 978-1-4612-6798-0. Citado na página 19.

DUARTE JR., E. P.; GARRETT, T.; BONA, L. C. E.; CARMO, R.; ZÜGE, A. P. Finding stable cliques of planetlab nodes. In: *2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2010. p. 317–322. ISBN 978-1-4244-7500-1. Disponível em: <http://dx.doi.org/10.1109/DSN.2010.5544300>. Citado na página 20.

FAHLE, T. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: MÖHRING, R.; RAMAN, R. (Ed.). *Proceedings of the 10th Annual European Symposium on Algorithms (ESA 2002)*. Berlin, Heidelberg: Springer, 2002. (Lecture Notes in Computer Science, v. 2461), p. 485–498. ISBN 978-3-540-45749-7. Disponível em: <http://dx.doi.org/10.1007/3-540-45749-6_44>. Citado 3 vezes nas páginas 25, 26 e 94.

FOMIN, F. V.; GRANDONI, F.; KRATSCH, D. Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: SIAM, 2006. (SODA), p. 18–25. ISBN 0-89871-605-5. Disponível em: <http://dx.doi.org/10.1145/1109557.1109560>. Citado 2 vezes nas páginas 13 e 20.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1979. Citado 2 vezes nas páginas 19 e 26.

GYÁRFÁS, A.; SEBŐ, A.; TROTIGNON, N. The chromatic gap and its extremes. *Journal of Combinatorial Theory, Series B*, v. 102, n. 5, p. 1155–1178, 2012. ISSN 0095-8956. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0095895612000470>. Citado na página 29.

HARARY, F. *Graph theory*. Addison-Wesley Pub. Co., 1969. (Addison-Wesley series in mathematics). Disponível em: <http://books.google.com.br/books?id=QNxgQZQH868C>. Citado na página 30.

HERAS, F.; LARROSA, J.; OLIVERAS, A. Minimaxsat: An efficient weighted max-sat solver. *J. Artif. Intell. Res.*, v. 31, p. 1–32, 2008. Citado na página 88.

HOU, B.; WANG, Z.; CHEN, Q.; SUO, B.; FANG, C.; LI, Z.; IVES, Z. G. Efficient maximal clique enumeration over graph data. *Data Science and Engineering*, v. 1, n. 4, p. 219–230, 2016. ISSN 2364-1541. Disponível em: <http://dx.doi.org/10.1007/s41019-017-0033-5>. Citado na página 19.

JIAN, T. An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. *IEEE Transactions on Computers*, IEEE Computer Society, Washington, DC, v. 35, n. 9, p. 847–851, 1986. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.1986.1676847>. Citado na página 19.

JOHNSON, D. S.; TRICK, M. A. (Ed.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*. Boston, MA, USA: American Mathematical Society, 1996. v. 26. ISBN 0821866095. Citado 2 vezes nas páginas 46 e 88.

KARP, R. M. Reducibility among combinatorial problems. In: MILLER, R. E.; THATCHER, J. W.; BOHLINGER, J. D. (Ed.). *Complexity of Computer Computations*. Springer US, 1972, (The IBM Research Symposia Series). p. 85–103. ISBN 978-1-4684-2003-6. Disponível em: <http://dx.doi.org/10.1007/978-1-4684-2001-2_9>. Citado na página 19.

KATAYAMA, K.; HAMAMOTO, A.; NARIHISA, H. An effective local search for the maximum clique problem. *Information Processing Letters*, Elsevier, v. 95, n. 5, p. 503–511, 2005. ISSN 0020-0190. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020019005001444>. Citado na página 29.

KONC, J.; JANEŽIČ, D. An improved branch and bound algorithm for the maximum clique problem. *MATCH Communications in Mathematical and in Computer Chemistry*, jun. 2007. Disponível em: <http://www.sicmm.org/konc/articles/match2007.pdf>. Citado 5 vezes nas páginas 27, 46, 88, 92 e 94.

KONC, J.; JANEŽIČ, D. Protein-protein binding-sites prediction by protein surface structure conservation. *Journal of chemical information and modeling*, American Chemical Society, Ljubljana, Slovenia., v. 47, n. 3, p. 940–944, mar. 2007. ISSN 1549-9596. Disponível em: <http://dx.doi.org/10.1021/ci6005257>. Citado na página 20.

KREHER, D. L.; STINSON, D. R. *Combinatorial algorithms: generation, enumeration, and search*. Boca Raton, Florida: CRC Press, 1999. (Discrete Mathematics and Its Applications). ISBN 978-0849339882. Citado 2 vezes nas páginas 19 e 21.

LAVNIKEVICH, N. *On the Complexity of Maximum Clique Algorithms: usage of coloring heuristics leads to the* $\Omega(2^{n/5})$ *algorithm running time lower bound*. 2013. Disponível em: <http://arxiv.org/abs/1303.2546>. Citado 2 vezes nas páginas 30 e 33.

LEVIN, L. A. Universal sequential search problems. *Problems of Information Transmission*, v. 9, p. 115–116, 1973. Disponível em: <http://mi.mathnet.ru/ppi914>. Citado na página 77.

LI, C.-M.; FANG, Z.; XU, K. Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In: *25th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2013. p. 939–946. Disponível em: <http://dx.doi.org/10.1109/ICTAI.2013.143>. Citado 2 vezes nas páginas 29 e 78.

LI, C.-M.; JIANG, H.; MANYÀ, F. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, v. 84, p. 1–15, 2017. ISSN 0305-0548. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054817300576>. Citado 2 vezes nas páginas 29 e 78.

LI, C.-M.; JIANG, H.; XU, R.-C. Incremental maxsat reasoning to reduce branches in a branch-and-bound algorithm for maxclique. In: SPRINGER. *International Conference on Learning and Intelligent Optimization*. 2015. p. 268–274. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6734837>. Citado 2 vezes nas páginas 29 e 78.

LI, C.-M.; QUAN, Z. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 2010. v. 1, p. 344–351. ISSN 1082-3409. Disponível em: <http://dx.doi.org/10.1109/ICTAI.2010.57>. Citado 2 vezes nas páginas 29 e 78.

LI, C.-M.; QUAN, Z. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In: *Twenty-Fourth Conference on Artificial Intelligence (AAAI)*. AAAI Publications, 2010. p. 128–133. Disponível em: <http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1611>. Citado 8 vezes nas páginas 29, 77, 78, 79, 81, 86, 92 e 93.

LUTZ, M. *Programming Python: Powerful Object-Oriented Programming*. 4. ed. Sebastopol, California: O'Reilly Media, Inc., 2010. ISBN 978-0-596-15810-1. Citado na página 94.

MASLOV, E.; BATSYN, M.; PARDALOS, P. M. Speeding up branch and bound algorithms for solving the maximum clique problem. *Journal of Global Optimization*, Springer US, v. 59, n. 1, p. 1–21, 2014. ISSN 1573-2916. Disponível em: <http://dx.doi.org/10.1007/s10898-013-0075-9>. Citado 2 vezes nas páginas 29 e 78.

MICHIE, D. "Memo" functions and machine learning. *Nature*, v. 218, n. 5136, p. 19–22, 04 1968. Disponível em: <http://dx.doi.org/10.1038/218019a0>. Citado na página 42.

MOON, J.; MOSER, L. On cliques in graphs. *Israel Journal of Mathematics*, v. 3, n. 1, p. 23–28, mar. 1965. Disponível em: <http://dx.doi.org/10.1007/BF02760024>. Citado 3 vezes nas páginas 19, 30 e 56.

NAUDÉ, K. A. Refined pivot selection for maximal clique enumeration in graphs. *Theoretical Computer Science*, v. 613, p. 28–37, 2015. ISSN 0304-3975. Disponível em: <http://dx.doi.org/10.1016/j.tcs.2015.11.016>. Citado na página 19.

PITTEL, B. On the probable behaviour of some algorithms for finding the stability number of a graph. *Mathematical Proceedings of the Cambridge Philosophical Society*, v. 92, p. 511–526, nov. 1982. ISSN 1469-8064. Disponível em: <http://dx.doi.org/10.1017/S0305004100060205>. Citado 2 vezes nas páginas 63 e 64.

PROSSER, P. Exact algorithms for maximum clique: A computational study. *Algorithms*, v. 5, n. 4, p. 545–587, 2012. ISSN 1999-4893. Disponível em: <http://dx.doi.org/10.3390/a5040545>. Citado na página 92.

ROBSON, J. M. Algorithms for maximum independent sets. *Journal of Algorithms*, v. 7, n. 3, p. 425–440, set. 1986. ISSN 01966774. Disponível em: <http://dx.doi.org/10.1016/0196-6774(86)90032-5>. Citado na página 20.

ROBSON, J. M. *Finding a maximum independent set in time* $O(2^{n/4})$. Université de Bordeaux, 2001. Disponível em: <http://www.labri.fr/perso/robson/mis/techrep.html>. Citado 2 vezes nas páginas 13 e 20.

SAN SEGUNDO, P.; ARTIEDA, J.; BATSYN, M.; PARDALOS, P. M. An enhanced bitstring encoding for exact maximum clique search in sparse graphs. *Optimization Methods and Software*, v. 32, n. 2, p. 312–335, 2017. Disponível em: <http://dx.doi.org/10.1080/10556788.2017.1281924>. Citado 2 vezes nas páginas 29 e 78.

SAN SEGUNDO, P.; ARTIEDA, J.; LEON, R.; TAPIA, C. An enhanced infra-chromatic bound for the maximum clique problem. In: PARDALOS, P. M.; CONCA, P.; GIUFFRIDA, G.; NICOSIA, G. (Ed.). *Machine Learning, Optimization, and Big Data: Second International Workshop (MOD 2016)*. Cham: Springer International Publishing, 2016. (Lecture Notes in Computer Science, v. 10122), p. 306–316. ISBN 978-3-319-51469-7. Disponível em: <http://dx.doi.org/10.1007/978-3-319-51469-7_26>. Citado 2 vezes nas páginas 29 e 78.

SAN SEGUNDO, P.; LOPEZ, A.; BATSYN, M.; NIKOLAEV, A.; PARDALOS, P. M. Improved initial vertex ordering for exact maximum clique search. *Applied Intelligence*, v. 45, n. 3, p. 868–880, 2016. ISSN 1573-7497. Disponível em: <http://dx.doi.org/10.1007/s10489-016-0796-9>. Citado 2 vezes nas páginas 29 e 78.

SAN SEGUNDO, P.; NIKOLAEV, A.; BATSYN, M. Infra-chromatic bound for exact maximum clique search. *Computers & Operations Research*, v. 64, p. 293–303, 2015. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/j.cor.2015.06.009>. Citado 5 vezes nas páginas 29, 78, 84, 93 e 99.

SAN SEGUNDO, P.; TAPIA, C.; LOPEZ, A. Watching subgraphs to improve efficiency in maximum clique search. In: _____. *Contemporary Challenges and Solutions in Applied Artificial Intelligence*. Heidelberg: Springer International Publishing, 2013. p. 115–122. ISBN 978-3-319-00651-2. Disponível em: <https://doi.org/10.1007/978-3-319-00651-2_16>. Citado 2 vezes nas páginas 29 e 78.

TANGE, O. GNU Parallel – the command-line power tool. *;login: The USENIX Magazine*, Frederiksberg, Denmark, v. 36, n. 1, p. 42–47, fev. 2011. Disponível em: <http://www.gnu.org/s/parallel>. Citado 2 vezes nas páginas 46 e 94.

TARJAN, R. E.; TROJANOWSKI, A. E. *Finding a maximum independent set*. Stanford, CA, USA, 1976. Citado na página 19.

THE SAGE DEVELOPERS. *SageMath, the Sage Mathematics Software System (Version 7.1)*. [S.l.], 2016. http://www.sagemath.org. Citado na página 33.

TOMITA, E. Efficient algorithms for finding maximum and maximal cliques and their applications. In: POON, S.-H.; RAHMAN, M. S.; YEN, H.-C. (Ed.). *WALCOM: Algorithms and Computation: 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29–31, 2017, Proceedings*. Cham: Springer International Publishing, 2017. p. 3–15. ISBN 978-3-319-53925-6. Disponível em: <http://dx.doi.org/10.1007/978-3-319-53925-6_1>. Citado na página 27.

TOMITA, E.; KAMEDA, T. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, Springer, v. 37, n. 1, p. 95–111, jan. 2007. ISSN 0925-5001. Disponível em: <http://dx.doi.org/10.1007/s10898-006-9039-7>. Citado 4 vezes nas páginas 20, 26, 92 e 94.

TOMITA, E.; KOHATA, Y.; TAKAHASHI, H. A simple algorithm for finding a maximum clique. *Technical Report of the University of Electro-Communications*, Tokyo, n. 1, p. 1–13, 1988. Citado na página 26.

TOMITA, E.; MATSUZAKI, S.; NAGAO, A.; ITO, H.; WAKATSUKI, M. A much faster algorithm for finding a maximum clique with computational experiments. *Journal of Information Processing*, v. 25, p. 667–677, 2017. Citado na página 27.

TOMITA, E.; SEKI, T. An efficient branch-and-bound algorithm for finding a maximum clique. In: CALUDE, C. S.; DINNEEN, M. J.; VAJNOVSZKI, V. (Ed.). *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science (DMTCS)*. Berlin, Heidelberg: Springer, 2003, (Lecture Notes in Computer Science, v. 2731). p. 278–289. ISBN 978-3-540-40505-4. Disponível em: <http://dx.doi.org/10.1007/3-540-45066-1_22>. Citado 2 vezes nas páginas 26 e 94.

TOMITA, E.; SUTANI, Y.; HIGASHI, T.; TAKAHASHI, S.; WAKATSUKI, M. A simple and faster branch-and-bound algorithm for finding a maximum clique. In: RAHMAN, M.; FUJITA, S. (Ed.). *Proceedings of the 4th International Workshop Algorithms and Computation (WALCOM)*. Berlin, Heidelberg: Springer, 2010, (Lecture Notes in Computer Science, v. 5942). p. 191–203. ISBN 978-3-642-11439-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-11440-3_18>. Citado 2 vezes nas páginas 26 e 94.

TOMITA, E.; TANAKA, A.; TAKAHASHI, H. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, v. 363, n. 1, p. 28–42, out. 2006. ISSN 03043975. Disponível em: <http://dx.doi.org/10.1016/j.tcs.2006.06.015>. Citado na página 19.

TOMITA, E.; YOSHIDA, K.; HATTA, T.; NAGAO, A.; ITO, H.; WAKATSUKI, M. A much faster branch-and-bound algorithm for finding a maximum clique. In: ZHU, D.; BEREG, S. (Ed.). *Proceedings of the 10th International Workshop Frontiers in Algorithmics (FAW)*. Cham: Springer International Publishing, 2016. (Lecture Notes in Computer Science, v. 9711), p. 215–226. ISBN 978-3-319-39817-4. Disponível em: <http://dx.doi.org/10.1007/978-3-319-39817-4_21>. Citado na página 26.

TRAKHTENBROT, B. A. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, IEEE, v. 6, n. 4, p. 384–400, 1984. Citado na página 77.

TSUKIYAMA, S.; IDE, M.; ARIYOSHI, H.; SHIRAKAWA, I. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, SIAM, v. 6, n. 3, p. 505–517, 1977. Disponível em: <http://dx.doi.org/10.1137/0206036>. Citado na página 19.

ZUCKERMAN, D. Linear degree extractors and the inapproximability of max clique and chromatic number. In: *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 2006. (STOC), p. 681–690. ISBN 1-59593-134-1. Disponível em: <http://doi.acm.org/10.1145/1132516.1132612>. Citado na página 19.

ZÜGE, A. P. *Solução Exata do Problema da Clique Máxima*. Dissertação (Mestrado) — Universidade Federal do Paraná, nov. 2011. Disponível em: <http://dspace.c3sl.ufpr.br/dspace/handle/1884/27588>. Citado 3 vezes nas páginas 13, 21 e 94.

ZÜGE, A. P.; CARMO, R. Maximum clique via maxsat and back again. In: *Latin American Workshop on Cliques in Graphs*. Pirenópolis: Instituto de Informática da Universidade Federal de Goiás, 2014. p. 29. Citado na página 78.

ZÜGE, A. P.; CARMO, R. Maximum clique via maxsat and back again. *Matemática Contemporânea*, Sociedade Brasileira de Matemática, v. 44, p. 1–10, 2016. Disponível em: <http://mc.sbm.org.br/wp-content/uploads/sites/15/2016/02/44-18.pdf>. Citado 2 vezes nas páginas 78 e 93.