UNIVERSIDADE FEDERAL DO PARANÁ

CLEVERSON SEBASTIÃO DOS ANJOS

ANÁLISE EXPERIMENTAL DE ALGORITMOS

CURITIBA
2015

CLEVERSON SEBASTIÃO DOS ANJOS

ANÁLISE EXPERIMENTAL DE ALGORITMOS

Dissertação de mestrado apresentada como requisito parcial para a obtenção do título de Mestre 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
2015

A599a Anjos, Cleverson Sebastião dos
Análise experimental de algoritmos/ Cleverson Sebastião dos Anjos. – Curitiba, 2015.
158 f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Ciência da Computação, 2015.

Orientador: Renato Carmo .
Bibliografia: p. 157-158.

1. Algoritmos - Análise. 2. Algoritmos - Testes. 3. Desempenho - Medição. I. Universidade Federal do Paraná. II.Carmo, Renato. III. Título.

CDD: 518.10285

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Cleverson Sebastião dos Anjos, avaliamos o trabalho intitulado, "Análise Experimental de Algoritmos", cuja defesa foi realizada no dia 21 de agosto de 2015, às 10:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X)**aprovação** do candidato. ( )**reprovação** do candidato.

Curitiba, 21 de agosto de 2015.

Prof. Dr. Renato Carmo
PPGInf - Orientador

Prof. Dr.Jaime Cohen
UEPG – Membro Externo

Prof. Dr. Fabiano Silva
PPGInf – Membro Interno

AGRADECIMENTOS

- Aos meus pais, João Caos dos Anjos e Ione Aparecida Bueno dos Anjos, e minha irmã Tânia Mara Luz que me ensinaram por exemplo o poder do trabalho duro, esforço e que tudo pode ser alcançado com a dose de comprometimento certa.
- Ao meu orientador, professor Dr. Renato Carmo, por me ajudar em todos os passos do mestrado, sempre com muito bom humor, compreensão e paciência, tornando-se assim um exemplo de professor e de pessoa que com certeza seguirei em minha carreira.
- Ao membros da banca, pelas sugestões e críticas que contribuíram substancialmente ao trabalho.
- Ao professor Dr. Lourival Aparecido de Gois, que me incentivou a ingressar na carreira acadêmica e que também tenho como um exemplo profissional.
- Ao professor Me. Alexandre Züge, pelo esclarecimento de inúmeras dúvidas e auxílio vital em várias fases deste trabalho.
- Aos meus amigos Giselle Rocha Cogo, Nicolle Sotto, Cristina Kasprzak, Sandro Ferreira, Gisele Cordeiro e José Luis Schamne pelo seu apoio, companheirismo e bom humor que tornaram esse processo mais agradável.
- À UTFPR, minha empregadora durante esse período, por ter me apoiado através da redução parcial de carga horária.

"Comece pelo começo, siga até chegar ao fim e então, pare." Alice no País das Maravilhas - Lewis Carroll.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

RESUMO

Neste trabalho aplicamos os conceitos de análise experimental de algoritmos, de acordo com o livro "A Guide to Experimental Algorithmics", da autora Catherine C. McGeogh de forma a analisar experimentalmente o comportamento de dez algoritmos exatos para o problema da clique máxima. O fazemos através da definição de um processo experimental, estabelecimento de um ambiente de testes estruturado, aplicação de um modelo de *design* e apresentação dos dados sob diferentes perspectivas. Enquanto realizamos uma comparação entre tais algoritmos buscamos também reportar nossa experiência e algumas das ideias e metodologia de análise experimental.

Palavras-chave: Análise Experimental de Algoritmos, Problema da Clique Máxima.

ABSTRACT

In this work we apply the concepts of experimental algorithm analysis, as exposed in the book "A Guide to Experimental Algoritmics", by the author Catherine C. McGeogh in order to perform experimentally the behaviour of 10 exact algorithms for the maximum clique problem. We do so by, defining an experimental process, establishing a structured testing environment, applying an experimental *design* model and displaying the data gathered under different perspectives. While performing a comparison between the algorithms we also aim at reporting our "hands on" experience with some of the ideas and methodology of experimental analysis.

Key-words: Experimental Analysis of Algorithms, Maximum Clique Problem

1 INTRODUÇÃO

A análise de algoritmos é em sua maior parte conduzida de forma teórica. Em sua grande maioria, esse estudo se dá no molde de abstração do problema, criação de uma proposição e, para encerrar, a prova do teorema, chamada Análise Assintótica. Essa abordagem possui um ponto forte, a universalidade de seus resultados, todavia, existe um problema com essa abordagem, a falta de especificidade. A implementação do algoritmo levará minutos ou anos para executar?

Além disso, existe um grande caminho a ser percorrido do pseudocódigo ao processo em execução. Ao longo desse caminho várias presunções podem ser necessárias a respeito da conversão do pseudocódigo em uma a implementação executável por um computador, que podem afetar o desempenho do algoritmo previsto pela Análise Assintótica.

Já a Análise Empírica de algoritmos busca justamente analisá-los através da implementação e reporte de seus tempos de execução. Enquanto essa abordagem ganha no quesito especificidade, pode pecar no conceito de generalidade. É notoriamente complexo traduzir tempos de execução de um algoritmo sobre uma entrada em uma plataforma em propriedades aplicáveis a outras plataformas e entradas.

Almejando prover uma diferente abordagem, existe a Análise Experimental, que busca justamente preencher a lacuna entre a falta de especificidade da Análise Assintótica e a falta de universalidade da Análise Empírica. Ao trabalhar como uma ponte entre a teoria e a prática, a Análise Experimental procura fornecer um meio para que analistas teóricos e práticos possam compartilhar conclusões, descobertas e informações iniciais sobre os algoritmos e seu desempenho.

Estudando essa modalidade de analisar algoritmos, bem como as características encontradas da literatura, buscamos aqui demonstrá-la, exemplificar como ela pode ser aplicada e como isso resulta em ganho de conhecimento inicial, replicabilidade e reprodutibilidade de testes e resultados, entre outras coisas.

Fazemos isso analisando experimentalmente 10 algoritmos que aplicam a técnica de Branch and Bound para resolver o problema da `clique máxima`. Estudamos o comportamento de tais algoritmos sobre três classes de grafos: aleatórios, DIMACS e Moon-Moser.

Os conceitos e modelos aplicados neste trabalho seguem o exposto no livro

A Guide to Experimental Algorithmics, da autora americana Catherine Cole McGeoch (MCGEOCH, 2012), sendo de nossa autoria apenas a interpretação e aplicação dos mesmos. Quando tratamos de conceitos não presentes em (MCGEOCH, 2012), referenciamo-los de forma pontual. Tal livro trata dos principais conceitos de análise experimental, reunindo boas práticas, ferramentas e principais erros cometidos pelos experimentadores. Todos os principais conceitos são amplamente exemplificados pela autora utilizando exemplos reais e didáticos. Além do que se aplica neste trabalho, o livro trás outros temas como redução de variância, *tunning* de código e de algoritmo, outras ferramentas para análise experimental, *profiling* e técnicas de análise de dados.

O texto restante é dividido da seguinte forma: a seguir são apresentadas definições e notação utilizadas no trabalho. No Capítulo 2 apresentamos os conceitos, definições e o processo experimental da Análise Experimental. No Capítulo 3 apresentamos as análises experimentais realizadas neste trabalho através da definição do ambiente de testes, classes de entrada, algoritmos analisados e modelos de *design*. No Capítulo 4 apresentamos os resultados experimentais separados por classe de entrada e, por fim, temos no Capítulo 5 a conclusão do trabalho.

1.1 DEFINIÇÕES E NOTAÇÃO

Discutimos agora o problema da clique máxima. Esse problema é utilizado para aplicarmos os conceitos de Análise Experimental ao decorrer do trabalho. Através dessa aplicação, esperamos que os conceitos apresentados sejam melhor compreendidos e que as vantagens da Análise Experimental fiquem mais evidentes.

Dado um conjunto S e um inteiro k, o conjunto de subconjuntos de S de tamanho k é denotado $\binom{S}{k}$. 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 de vértice de G. Cada elemento de $E(G)$ é chamado de aresta de G.

Um vértice v é vizinho de um outro vértice u em G se $\{u, v\}$ é uma aresta de G. A vizinhança de um vértice v em G é o conjunto dos vizinhos de v em G, denotado por $\Gamma_G(v)$, isto é $\Gamma_G(v) := \{u \in V(G) : \{u, v\} \in E(G)\}$.

O grau do vértice v em G é o número de vizinhos de v em G, denotado por $d_G(v) := |\Gamma_G(v)|$. O maior grau de um vértice em G é denotado por $\Delta(G)$.

Dado um conjunto $Q \subseteq V(G)$, a vizinhança comum dos vértices de Q é definida

por $\Gamma^{\cap}_{G}(Q) := \cap_{v \in Q}\, \Gamma_{G(v)}$.

Se $Q = \emptyset$ (conjunto vazio), convencionamos $\Gamma^{\cap}_{G} := V(G)$.

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

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.

$C(G)$ denota o conjunto de todos os cliques de G. O número de clique de G é o tamanho de uma clique máxima de G, denotado $\omega(G)$, isto é, $\omega(G) := max\{|c| : c \in C(G)\}$.

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 $\gamma : V(G) \to \{1, ..., k\}$ tal que $\gamma^{-1}(c) = \{v \in V(G) | \gamma(v) = c\}$ é independente em G, para todo $c \in \{1, ..., k\}$.

Dizemos que $\gamma(v)$ é a *cor* de v, segundo γ. Uma coloração de G é uma k-coloração de G para algum k. O número de cores k em uma coloração γ de G é denotado $\gamma(G)$. Observamos que $\omega(G) \leq \gamma(G)$ para toda coloração γ de G.

Define-se também o problema computacional da clique máxima como: dado um grafo G deve-se retornar uma clique máxima em G.

2 A ANÁLISE EXPERIMENTAL

Apresenta-se nesse capítulo os conceitos que utilizamos da Análise Experimental dividindo-o da seguinte forma: inicialmente as principais definições e conceitos e em seguida, o processo experimental aplicado.

2.1 DEFINIÇÕES E CONCEITOS

A Análise Experimental busca combinar elementos de ambos os modelos de análise, assintótica e empírica, tratando os algoritmos de forma semelhante a experimentos de laboratório, estudando sua execução através do controle de parâmetros, isolamento de componentes chave, construção de modelos e análise dos dados coletados.

Enquanto a análise empírica é realizada em um "ambiente natural", a Análise Experimental é realizada em ambiente controlado, para que se conheçam os elementos envolvidos no comportamento do algoritmo que influenciam nos resultados apresentados com maior embasamento.

Realizar experimentos práticos permite-nos refinar a base teórica à medida que uma hipótese pode ser implementada, submetida a experimentos e então, de posse dos resultados dos experimentos, podemos encontrar oportunidades de melhoramentos e aplicá-los em nossa hipótese teórica.

Assim, percebemos que a função de realizar experimentos não é substituir a Análise Assintótica, mas sim criar um ciclo onde a teoria dá início a um processo experimental que por sua vez melhora a nossa base teórica original, o que pode acarretar novos experimentos e assim por diante.

Na Análise Experimental, não é o algoritmo abstrato que é alvo de análise, mas sim as implementações que resolvem problemas com instâncias pré-definidas em uma plataforma em particular aplicando um determinado algoritmo.

Além disso, todos os experimentos realizados buscam garantir a reprodutibilidade, isto é, é possível conduzir experimentos novos e obter dados que ofereçam suporte à conclusão do experimento original? E comparabilidade, isto é, é possível comparar resultados de terceiros com os resultados originais de forma justa e fundamentada?

Neste trabalho tais elementos são para fazer uma comparação justa e bem funda-

mentada entre dez algoritmos que resolvem um mesmo problema. A seguir os principais conceitos aplicados são apresentados, sempre buscando associá-los com suas aplicações práticas para maior clareza.

Aplicamos nesse trabalho o conceito de "Conte a História Toda" que encoraja o registro e reporte de todos os passos que de alguma forma levaram ao resultado final. Normalmente demonstram-se apenas os passos que levaram ao resultado apresentado, sendo deixados de lado aqueles que representaram uma tentativa e erro, ou aqueles que, embora tenham contribuído de alguma forma para o resultado final, não fazem parte dele. Em contrapartida, buscamos não sobrecarregar o leitor com excesso de informação, para não correrm os o risco de assim ofuscar os resultados finais.

2.2 O PROCESSO EXPERIMENTAL

Nesta seção os principais passos para a definição de um experimento são apresentados. Os itens seguir não precisam sempre serem seguidos de forma sequencial. Pode ocorrer que a análise dos dados não resulte em nenhuma informação útil, fazendo assim com que o experimentador retorne para as fases iniciais ou ainda durante a definição do experimento, note-se que existem ferramentas mais apropriadas do que as escolhidas no passo anterior. Assim, sempre tem-se a opção de retornar para passos anteriores para refinar e melhorar o experimento.

Segue abaixo o molde do processo experimental também esquematizado na Figura 1:

1. Planejamento do experimento

   1.1. Formular a pergunta a ser respondida pelo experimento.

   1.2. Construir o ambiente de testes.

   O ambiente de testes compreende as ferramentas necessárias (*hardware* e *software*), bem como as instâncias e algoritmos.

   1.3. Definir o experimento, explicitando quais propriedades serão medidas, quantos testes serão aplicados, e assim por diante. Tais propriedades são definidas adiante nesse capítulo.

2. Execução do experimento;

2.1. Executar os testes e coletar os dados.

2.2. Analisar os dados em busca de respostas para a pergunta inicial. Se a pergunta não for respondida, retorna-se ao passo 1.

2.3. Publicar as informações descobertas (para o meio acadêmico).

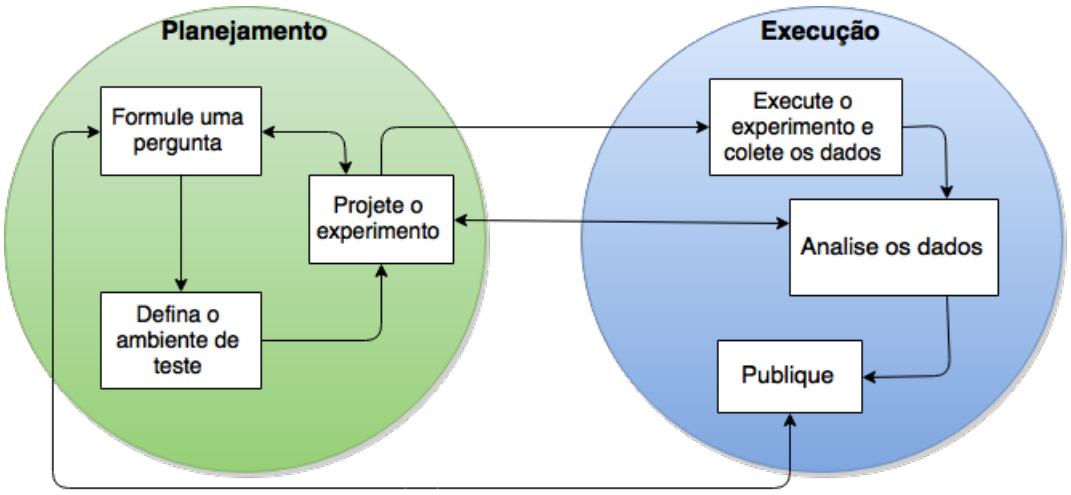

Figura 1: O Processo Experimental

Antes de prosseguir, deve-se distinguir dois conceitos: *experimento* e *teste*. O experimento refere-se ao processo como um todo, desde a definição da pergunta até a análise dos resultados. Um teste é uma execução do algoritmo para uma determinada instância.

A seguir dedicamos uma seção para cada um dos itens do processo experimental.

2.2.1 Formulação da Pergunta

Todo experimento é realizado com um propósito. Esse propósito deve ser definido na forma de uma ou mais perguntas. Na Seção 3.4 encontramos as perguntas definidas para cada classe de entrada.

Além de comparar a eficiência de algoritmos, pode-se usar a Análise Experimental para responder outros tipos de pergunta. Temos aqui algumas perguntas definidas em (JOHNSON, 1996) que aplicam-se à maior parte dos experimentos de análise de algoritmos:

- Como detalhes de implementação, configuração de parâmetros, heurísticas e escolha de estrutura de dados afetam o tempo de execução do algoritmo?

- Como o tempo de execução do algoritmo varia de acordo com o tamanho da instância e como isso depende da estrutura da instância?
- Qual a contagem de qual operação do algoritmo que melhor ajuda a descrever seu tempo de execução?
- Quais, na prática, são os gargalos computacionais do algoritmo e como eles dependem do tamanho e estrutura da instância? Como isso difere da previsão feita pela análise de pior caso?
- Como o tempo de execução (e a taxa de crescimento do tempo de execução) são afetados pela arquitetura da máquina? Um teste de *profiling* detalhado (análise de recursos gastos em cada instrução do algoritmo) pode ajudar a explicar isso?
- Dado que se está executando as mesmas (ou similares) instâncias em uma única máquina, quão previsíveis são os tempos de execução/contagem de operações para instâncias maiores do que as que foram testadas?
- Qual é a resposta das perguntas acima quando trocamos "tempo de execução" por algum outro tipo de recurso computacional?
- Dada uma classe substancialmente nova de instâncias, existe uma mudança significativa no comportamento do algoritmo para algoritmos previamente conhecidos?

2.2.2 Definição do Ambiente de Testes

A definição do ambiente de testes consiste em reunir todas as ferramentas necessárias para a condução dos experimentos, bem como a documentação dessas ferramentas em detalhes, para que o ambiente possa, caso necessário, ser reproduzido da forma mais fiel possível.

No que diz respeito ao *hardware*, pode-se citar: dados de processador como fabricante, arquitetura, modelo, frequência de *clock*, número de núcleos e capacidade de armazenamento das memórias *cache* (apenas o modelo não é suficiente, pois nem sempre através do modelo podemos conseguir todas as informações sobre ele, ou, com o passar do tempo, podemos não mais encontrar referências completas), informações sobre a memória primária e secundária. Se houver algum item incomum de *hardware*, deve ser mencionado.

Quando os testes forem executados, deve-se registrar a quantidade de memória primária utilizada pelos demais processos.

Quanto ao ambiente de *software*, deve-se definir essencialmente qual o sistema operacional e sua versão, qual o compilador, sua versão e se está sendo empregado algum tipo de otimização e a linguagem de programação utilizada para a implementação dos algoritmos. Caso deseje-se aplicar testes de *profiling* (análise de recursos gastos em cada instrução do algoritmo) no código fonte, especificar a ferramenta utilizada.

Para conduzir os testes, deve-se utilizar mais de uma plataforma de *hardware* e conduzir os mesmos testes em ambas as plataformas. Os resultados deverão seguir o mesmo padrão. Caso isso não ocorra, deve-se procurar por processos, defeitos no código, configurações de ferramentas de teste ou outros elementos que possam interferir nos resultados obtidos.

Outra forma de validar os resultados é solicitar que outra pessoa conduza isoladamente o mesmo experimento. Assim, coloca-se em teste sua reprodutibilidade.

Neste trabalho usamos as servidoras *Achel* e *Latrappe* do laboratório de Informática da UFPR, a linguagem de programação *Python* para a implementação dos algoritmos e o ambiente *Linux* para a execução dos experimentos. O Capítulo 3 contém descrições detalhadas dessas ferramentas.

2.2.2.1 Classes de Entrada e Algoritmos

Deve-se registrar quais são as classes de entrada usadas na execução dos testes e quais os algoritmos utilizados, bem como suas características principais.

As entradas, de acordo com (MCGEOCH, 2012), podem ser classificadas de acordo com seu propósito dentro dos experimentos:

- Entradas de teste de estresse: são classes usadas para procurar falhas no código e testar o algoritmo sob condições extremas. Um algoritmo que recebe um grafo como entrada pode utilizar grafos completos ou sem nenhuma aresta, por exemplo;

- Entradas de pior caso: são as entradas que representam os piores casos na execução do algoritmo, que demandam o máximo de um determinado recurso, como memória ou tempo de processamento, por exemplo, representado pelo maior número possível de execuções de uma dada operação. Esse tipo de entrada pode ser de difícil

determinação, pois nem sempre é conhecido para o tipo de problema computacional tratado.

- Entradas aleatórias: são entradas, como o próprio nome diz, geradas de forma aleatória, a partir de parâmetros manipulados de forma direta ou indireta.
- Entradas estruturalmente aleatorizadas que se subdividem em duas categorias:
  - Geradores centrados no algoritmo: oferecem controle sobre aspectos chave do algoritmo que buscam evidenciar algum tipo de comportamento. Se um algoritmo executado sobre grafos depende de alguma forma no número cromático do grafo, pode-se usar classes que embora sejam geradas aleatoriamente, permitem definir tal número cromático, para que possamos analisar, por exemplo, seu efeito no resultado final da execução do mesmo.
  - Geradores centrados na realidade: geram entradas aleatorizadas que se assemelham a entradas obtidas quando o algoritmo é executado em situações do dia-a-dia.
- Entradas reais: entradas que são usadas de fato quando o algoritmo é executado no mundo real. Podem ser difíceis de encontrar em grande quantidade.
- Entradas híbridas: combinam elementos das entradas reais com componentes gerados artificialmente. Como entradas reais podem ser difíceis de reunir, pode-se utilizar as entradas reais obtidas e aleatorizar alguns elementos delas gerando mais opções de entrada.
- *TestBed* Público: entradas fornecidas publicamente juntamente com suas características principais. Esse tipo de classe é bastante útil dado que sua ampla disponibilidade faz com que seja um terreno comum para a comparação de algoritmos que resolvem o mesmo problema computacional.

As entradas utilizadas nos experimentos deste trabalho são: grafos aleatórios, grafos de Moon-Moser e os grafos DIMACS para o problema da clique máxima. Tais classes são descritas em maiores detalhes no Capítulo 3.

2.2.3 Definição do Experimento

Neste passo o experimento é moldado, construindo um roteiro a ser seguido.

A seguinte terminologia, oriunda de (MCGEOCH, 2012), é empregada para referenciar os elementos do experimento:

- Métrica de desempenho: uma dimensão do algoritmo que pode ser medida, como por exemplo, tempo de execução ou consumo de memória. Neste trabalho utiliza-se a métrica de tempo de execução do algoritmo implementado.

- Indicador de desempenho: uma instância da métrica de desempenho, como o consumo de memória RAM ou Tempo de CPU que pode ser medida em um experimento. Aqui, combinamos dois indicadores: o Tempo de CPU e o número de Passos de *Branching* executados por cada algoritmo. A motivação por trás dessa escolha encontra-se detalhada na Seção 2.2.3.2.

- Parâmetro: qualquer propriedade que afete de alguma forma o resultado do indicador de desempenho. Existem três tipos de parâmetros:

  – Parâmetro de algoritmo: propriedade relacionada diretamente com o funcionamento do algoritmo, como o número de iterações em um laço.

  – Parâmetro de instância: são as propriedades associadas diretamente com os dados de entrada usados na execução do algoritmo.

  – Parâmetros de ambiente: são as propriedades do ambiente em que se executa os testes, como versão do compilador, linguagem de programação, *hardware*, entre outros.

  Utilizamos parâmetros de instância, número de arestas e vértices dos grafos de entrada, para alguns tipos de grafo, por exemplo, e parâmetros de ambiente, uma vez que executamos os testes em duas máquinas com configurações distintas.

- Fator: uma propriedade que é manipulada de forma explícita durante os experimentos. Como exemplo, para experimentos com grafos aleatórios, os fatores podem ser o número de vértices n e a probabilidade de cada aresta estar presente p.

- Nível: o valor atribuído para cada fator durante um teste. Pode ser executado um teste onde utiliza-se um grafo completo com 100 vértices (n) e, consequentemente, 4.950 arestas (m). O nível de n então é 100 e o nível de m é 4.950.

- Ponto de *Design* (*Design Point*): uma combinação de níveis para os fatores a serem usados em um teste. A combinação dos níveis acima mencionados representa um

ponto de *design*. Utilizamos a técnica de *Design* Fatorial Completo, explicada em maiores detalhes na Seção 2.2.3.3, para a definição dos níveis de fatores.

- Teste: uma execução do algoritmo com um dado ponto de *design* que gera um resultado mensurável do indicador de desempenho, ou seja, executa-se o algoritmo usando a entrada configurada de acordo com o ponto de *design* escolhido.

- Parâmetro Fixo: um parâmetro cujo nível é mantido durante a execução dos testes. Pode-se, por exemplo, fixar o número de vértices do grafo em 100 e variar o número de arestas. Assim, o número de vértices é o parâmetro fixo.

- Parâmetro de Ruído: um parâmetro sobre qual não se possui controle absoluto, podendo variar de forma semi-controlada ou descontrolada. No caso dos grafos aleatórios usados, tem-se controle apenas sobre o número de vértices e a probabilidade com que as arestas estão presentes. Assim, tem-se uma estimativa do número de arestas, mas seu valor não pode ser controlado com exatidão.

2.2.3.1 Tipos de Experimento

Experimentos são conduzidos basicamente por um dos seguintes motivos:

- Descoberta: o experimento é realizado em busca de informações gerais sobre o algoritmo. Por exemplo, quanto tempo esse algoritmo leva para executar em meu computador dada essa faixa de tamanhos de entrada? Existem partes do algoritmo que afetam expressivamente o desempenho como um todo?

- A corrida de cavalos: o experimento busca provar que um determinado algoritmo é melhor, sob algum aspecto, que outros algoritmos que resolvem o mesmo tipo de problema computacional, através de comparações entre eles.

Experimentos de descoberta são bastante oportunos no início do processo experimental, pois os resultados podem ajudar a definir quais questões devemos perguntar. Se um determinado algoritmo apresenta um Tempo de CPU destoante quando é usada uma determinada classe de entrada, podemos usar os experimentos para descobrir qual elemento é responsável por tal comportamento, por exemplo, ou descobrir quais níveis de fatores são praticáveis dado o *hardware* disponível.

2.2.3.2 Escolha de Indicador de Desempenho

Frequentemente a métrica de desempenho mais interessante é o tempo de execução, e em segundo lugar, no caso de algoritmos de aproximação ou que fazem o uso de heurísticas, a qualidade da solução, isto é, quão próxima da solução correta está a solução fornecida pelo algoritmo?

Neste trabalho empregamos dois indicadores que trabalham em conjunto: Tempo de CPU e número de Passos de *Branching* executados pelo algoritmo de busca da clique máxima.

É importante destacarmos a diferença entre *tempo de execução* e *tempo de CPU*. O primeiro é uma métrica de desempenho, sendo o segundo o indicador de desempenho.

Encontra-se um problema maior, no que diz respeito à generalidade, quando se deseja analisar o Tempo de CPU. É bastante trivial apenas reportar tal tempo, mas como saber o quão precisa é a medição realizada? Encontra-se em (BARTZ-BEIELSTEIN et al., 2010) a crítica de, no caso de experimentos com algoritmos, sua execução produz resultados menos confiáveis que os resultados teóricos, pois são dependentes de contexto e de arquitetura, tornando-os menos gerais e reprodutíveis. Essa observação não desencoraja a prática experimental, antes a fortalece através da atenção e busca de formas de resolver tal problema. Por isso a medição de Tempo de CPU deve ser feita de forma embasada e estruturalmente bem definida, para que possamos reduzir a qualidade errática a ela atribuída.

Objetivando sanar esse problema busca-se então formas de tornar o Tempo de CPU informado mais confiável e comparável. Com o controle de ambiente de *hardware* e *software* e definição clara de fatores, níveis e pontos de *design*, ditado pela Análise Experimental, conta-se com um maior embasamento por trás do tempo reportado, auxiliando no entendimento dos resultados.

Por fim, recomenda-se que seja associado um fator independente de plataforma com o Tempo de CPU, como a contagem de uma operação dominante. Assim, quem busca efetuar comparações entre experimentos possui informações independentes de plataforma para embasar suas conclusões ou entender a relação entre o funcionamento do algoritmo e seu Tempo de CPU reportado.

Um cuidado que devem-se tomar é escolher indicadores que permitam comparações entre outros algoritmos. Por esses motivos, associamos o número de Passos de *Branching*

dos algoritmos com o Tempo de CPU. Assim, associamos um fator independente de plataforma com um outro fator bastante importante, porém dependente de plataforma.

Alguns dos algoritmos analisados possuem características próprias que são passíveis de análise, mas como não estão presentes em todos os algoritmos, não permitem uma comparação completa, por isso não são utilizados em nossos experimentos.

Outros indicadores de desempenho podem ser utilizados, como consumo de memória primária e/ou secundária e consumo de banda da rede. Entretanto, um cuidado que se deve tomar é o de não escolher vários indicadores, o que resulta em uma grande quantidade de dados que podem ser difíceis de "navegar" e obter conclusões.

2.2.3.3 *Design* Fatorial Completo

Buscando a estruturação dos testes e categorização de acordo com as escolhas de fatores e pontos de *design*, outra questão aflora: como comparar resultados dos testes realizados com cada ponto de *design* entre os diferentes algoritmos?

Oriundo do Design de Experimentos (*Design of Experiments*), o *Design* Fatorial Completo, para k fatores dados, consiste em definir 2^k pontos de *design*, onde k representa o número de fatores. Para cada um dos fatores, são estabelecidos dois níveis, um alto e um baixo. Esses níveis representam valores para os quais se espera que o respectivo indicador de desempenho resultante da execução do algoritmo com fatores em tais níveis tenha um valor "alto ou baixo", e não que necessariamente o valor do fator seja "alto ou baixo". Em seguida, aplica-se todas as combinações possíveis entre os níveis desses fatores. Segundo a convenção do *Design* de Experimentos, simboliza-se um fator de nível alto com "+" e baixo com "-".

A Tabela 1 mostra um exemplo de Modelo de *Design* fatorial completo com dois fatores. As instâncias utilizadas para os algoritmos utilizados neste trabalho são grafos. Os dois fatores no exemplo são o número de vértices n e o número de arestas m.

Como, no âmbito deste trabalho, o nível do fator é crescente com o tamanho do indicador de desempenho (Tempo de CPU), os símbolos de "+" e "-" resultarão em valores do indicador de performance relativamente altos e baixos, respectivamente.

Uma vez que estes grafos fazem parte de um Tesbed Público, ou seja, são instâncias pré-definidas, onde não podemos escolher fatores e manipular seus níveis a técnica de *Design* Fatorial Completo não será aplicável apenas no caso dos grafos DIMACS.

Fatores	Experimentos			
	1	2	3	4
n	-	-	+	+
m	-	+	-	+

Tabela 1: Cada coluna representa uma combinação de níveis dos fatores m e n

A Tabela 1 mostra que devem ser conduzidos 4 testes diferentes ($k = 2$), com as dadas combinações de níveis de fatores (ou pontos de *design*). Na Tabela 2, temos uma exemplificação.

Fatores	Pontos de Design			
	1	2	3	4
n	100	100	1.000	1.000
m	400	400.000	400	400.000

Tabela 2: Cada coluna representa um ponto de *design*.

Finalmente, definimos um número de repetições de execuções de testes para cada ponto de *design* e reportamos a média dos resultados obtidos, buscando assim, reduzir efeito de ruídos causados por fatores externos.

2.2.3.4 Condição de Parada

Uma das propriedades que devem ser definidas para executar o experimento e que mais influencia na reprodutibilidade e comparabilidade é a condição de parada, ou seja, quando se deve interromper a execução do algoritmo demorado. Uma estratégia mal escolhida deixa o algoritmo executar por mais tempo que o necessário, desperdiçando tempo, ou encerra sua execução prematuramente, ocultando dados importantes.

Pode-se encontrar na literatura exemplos como (FAHLE, 2002) e (KONC; JANEŽIČ, 2007) que define seis horas de tempo máximo. Existem dois problemas com essa abordagem: em primeiro lugar, tempos de execução são notoriamente difíceis de traduzir entre plataformas. Além de prejudicar a replicabilidade, uma das principais características que um experimento deve conter, a comparabilidade, é afetada, ou seja, outros pesquisa-

dores poderão encontrar dificuldades em comparar seus resultados com os apresentados inicialmente.

Caso apenas o Tempo de CPU tenha sido utilizado como indicador de performance, quem quiser efetuar comparações deve recorrer a abordagens que podem afetar a confiabilidade de seus resultados. Como exemplo, em (FAHLE, 2002) encontramos a seguinte passagem

> Presumindo que o computador deles é cerca de 4 vezes mais lento que o nosso, ganhamos do algoritmo deles em 13 casos.

e apenas essa sentença é usada para embasar a conversão dos resultados de terceiros com os resultados do autor. Deixa-se assim uma grande lacuna a ser preenchida. A conversão de valores levou em conta dados mais criteriosos? Como chegou-se a conclusão de que uma máquina era de fato quatro vezes mais rápida que a outra?

Quando comparamos algoritmos, uma alternativa que temos, para tornar as comparações mais justas, é escolher algum critério de alguma propriedade do algoritmo como critério de parada, como a contagem de alguma variável.

Infelizmente, mesmo cientes desse problema, não encontramos para este trabalho outro critério de parada que não seja a definição de um limite.

Como uma forma de amenizar o problema adotamos a prática da associação entre Tempo de CPU e número de Passos de *Branching* sempre em conjunto, assim ainda usamos o tempo como critério de parada, desde que apresentemos o número de passos correspondente com o tempo total de processamento.

Em nossos experimentos a condição de parada só se faz necessária para a família de Grafos DIMACS, descrita em detalhes na Seção 3.2.2. Utilizamos um experimento de descoberta, descrito na Seção 3.4.2, para definir o tempo limite utilizado como critério de parada.

2.2.3.5 Modelo de *Design* Experimental

Todos os elementos citados anteriormente são unificados no Modelo de *Design* Experimental, que é basicamente um guia e concentrador de informações sobre o experimento a ser conduzido. Temos a seguir o Modelo de *Design* Experimental:

- **Em qual equipamento o experimento foi executado?**
  O propósito desse item é organizar e distinguir entre testes diferentes.
- **Qual era a carga do sistema no momento da execução?**
  Registramos a quantidade de memória RAM disponível no início da execução dos testes.
- **Pergunta:**
  Que tipo de informação pretende-se obter com esse experimento?
- **Indicadores de desempenho:**
  Quais indicadores serão utilizados?
- **Fatores:**
  Quais os fatores escolhidos?
- **Níveis:**
  Que níveis serão atribuídos para os fatores escolhidos?
- **Testes:**
  Quantos testes para cada ponto de *design* serão feitos?
- **Pontos de *Design*:**
  Quais pontos de *design* serão testados?
- **Saídas:**
  quais são os valores (dados) obtidos a partir da execução dos testes que serão registrados?

De posse dessas informações, podemos dar início à execução dos testes. Os modelos utilizados para cada tipo de grafo encontram-se no Capítulo 3.

2.2.4 Execução dos Testes

Neste passo executa-se o programa de testes, ou seja, a implementação do algoritmo que é analisado sobre a entrada escolhida seguindo o Modelo de *Design* previamente definido.

Duas questões merecem atenção aqui. A primeira é definir se serão executados testes do tipo piloto (*pilot test*) ou de trabalho (*workhorse test*).

O teste piloto é aquele usado em experimentos de descoberta, para ganhar conhecimento inicial do funcionamento algoritmo. Isso pode se dar na forma de contadores, que reportam quantas vezes um laço foi executado, por exemplo, ou qual era o estado de uma variável em certo ponto da execução. Embora essas modificações possam parecer inofensivas para o desempenho do algoritmo, testes piloto podem conter cálculos que alteram significativamente sua eficiência. Nesse tipo de teste não há preocupação com a eficiência na execução, apenas devemos garantir que ele resolva o problema computacional na forma proposta pelo algoritmo.

Já o teste de trabalho preocupa-se primariamente com a eficiência na execução do algoritmo. Todos os contadores e demais rotinas que não foram definidos como as saídas a serem registradas no Modelo de *Design* devem ser suprimidos.

A segunda questão é a recomendação de não ceder à tentação de analisar resultados parciais. Muitas vezes, e principalmente quando cada execução de teste é dispendiosa, o experimentador pode sentir vontade de analisar os dados parciais em busca de respostas. Essa análise prematura pode levar a conclusões precipitadas que não serão aplicáveis aos demais testes. Mais prejudicial ainda é quando o experimentador, aproveitando-se da característica cíclica do processo experimental, decide interromper seus testes prematuramente e redefinir passos anteriores com base nas conclusões tiradas de uma faixa restrita (e insuficiente) de resultados.

2.2.5 Análise dos Dados

Após a execução da implementação do algoritmo e a coleta dos resultados, dá-se prosseguimento à análise para extração de informações.

Muitas vezes, os dados produzidos pelos testes fornecem poucas informações, ou informações sobre as quais não é possível fazer afirmações bem embasadas, fazendo com que o preenchimento das lacunas geradas entre a análise e os resultados sejam deixadas para preenchimento pela imaginação do leitor. Para evitar tal situação, recomenda-se:

- Executar mais testes: talvez a forma mais simples de produzir conclusões mais bem embasadas é ter um número de resultados relativamente grande.

- Aumentar o número dos níveis do fator: se a resposta do custo relacionado ao nível

do fator é pequena quando comparado à variação, aumentar a faixa dos valores testados para ele pode melhorar a resposta. Infelizmente, por restrições de tempo e de *hardware*, não podemos conduzir experimentos com os níveis de fator desejados.

- Não resumir os dados prematuramente: os programas de teste devem sempre gerar valores "crus", mostrando os resultados de cada teste, para que a distribuição desses dados possa ser analisada. Posteriormente, podemos definir estatísticas como variância ou média amostrais, por exemplo.

- Reduzir (ou aumentar) os dados amostrais para o tamanho certo: evitar produzir muitos (ou muito poucos) dados nos testes. Uma quantidade pequena de dados pode ocultar propriedades importantes enquanto muitos dados podem tornar difícil enxergar o que realmente importa (reportar o valor de uma variável em cada iteração de um laço de repetição, ao invés de reportar apenas o valor na saída do laço, por exemplo).

- Não usar indicadores de desempenho que podem resumir ou ocultar propriedades: em nosso caso, poderíamos definir nossos algoritmos para reportar apenas os valores médios dos testes executados, mas isso poderia ocultar discrepâncias ou outras informações importantes. Assim, cada execução tem seus dados individuais reportados e o cálculo médio é realizado posteriormente.

- Usar indicadores de desempenho específicos: indicadores de desempenho específicos concentram-se em uma parte do algoritmo, fazendo assim com que seja mais fácil analisar ou identificar certas propriedades ligadas ao identificador. Variações de resultado identificadas através de indicadores de desempenho amplos podem ser resultado de um agregado de propriedades, cujo relacionamento nem sempre é claro.

2.2.5.1 Experimento Dobrado

O conceito de Experimento Dobrado (*Doubling Experiment*) foi inicialmente proposto em (SEDGEWICK; SCHIDLOWSKY, 1998). Esse experimento pode ser empregado no início do processo experimental, para colhermos informações sobre o desempenho do algoritmo. Basicamente o que o autor propõe é que podemos estimar a taxa de crescimento da função de custo de muitos algoritmos comparando um indicador de desempenho a medida que o tamanho da entrada dobra. Além disso, podemos verificar se as presunções

de desempenho feitas na teoria são traduzidas para a prática. Podemos interpretar os resultados da seguinte forma:

- Se os valores resultantes do indicador de desempenho não mudam com o aumento do nível de n, a função de custo é constante;
- Se os valores do indicador são incrementados por uma constante, a função de custo é $\theta(log\ n)$
- Se ao dividirmos cada resultado do indicador de desempenho por n e os valores aumentarem de forma constante, a função de custo é $\theta(n\ log\ n)$
- Se o custo dobra a medida que n dobra, a função de custo é linear.
- Se o custo quadruplica a medida que n dobra, a função de custo é $\theta(n^2)$

Utilizamos uma versão simples desta ideia durante a análise dos resultados dos grafos de Moon-Moser, na Seção 4.3.

2.2.5.2 Técnicas de Redução de Fatores

Abordamos no início deste capítulo *Design* Fatorial Completo, que fornece um meio de definirmos quais os níveis e pontos de *design* que usaremos nos testes. Um problema com essa abordagem é que a quantidade de testes demandados cresce de forma exponencial em função do número de fatores. O exemplo citado mostra que caso sejam identificados três fatores, teremos oito pontos de *design*. Se adicionarmos mais 2 fatores, passamos a ter $2^5 = 32$ fatores. Digamos então que escolhemos 20 pontos de *design* para serem testados e que vamos executar o programa de testes 3 vezes para cada ponto de *design*. Temos um total de 1.920 execuções do programa de testes a serem feitas. Caso o programa de teste seja de execução rápida, isso não representa um grande problema, mas caso tenhamos um programa de teste que demanda uma grande quantidade de tempo para executar, devemos buscar alternativas para não gastarmos uma quantidade exagerada (da qual muitas vezes não se dispõe) de tempo. Apresentamos algumas opções para reduzir o tamanho do conjunto de fatores:

- Unir fatores similares: caso dois fatores possuam um efeito similar sobre o desempenho, podemos tratá-los como um único fator, realizando testes apenas em casos onde o ponto de *design* é ++ e - -, excluindo assim os casos -+ e +-;

- Usar dados de rastreamento para inferir os efeitos de fatores omitidos: podemos modificar o código para que ele reporte estados de variáveis em determinados pontos de forma que possamos tirar conclusões sobre o efeito do “ex-fator"sobre o funcionamento do algoritmo;

- Converter fatores para parâmetros de ruído: ao invés de definir explicitamente os níveis para um fator, podemos deixar que ele varie de acordo com uma distribuição de probabilidade simples e reportar seu valor em testes aleatórios. Temos assim um conjunto de níveis e medidas de desempenho que podem ser usadas posteriormente.

- Limitar o escopo do experimento: através da fixação de fatores e redução do número de níveis. Se os resultados não variam muito de nível para nível, podemos aplicar um espaçamento maior entre os valores de nível, por exemplo. Utilizamos essa técnica durante a análise dos resultados dos grafos de Moon-Moser, na Seção 4.3.

3 ANÁLISE EXPERIMENTAL

Neste capítulo, aplicamos os conceitos apresentados no Capítulo 2 para comparar diversos algoritmos para o problema da clique máxima. Apresentamos três experimentos diferentes, cada um envolvendo uma família de instâncias diferente, descritas na Seção 3.2.

Inicialmente apresentamos na Seção 3.1 o ambiente de *hardware* e *software* utilizado nos experimentos. Apresentamos nas Seções 3.2 e 3.3 as classes de grafos e algoritmos usados, respectivamente. Finalmente, apresentamos na Seção 3.4 os modelos de *design* também divididos por tipos de instância. Elementos comuns são descritos apenas uma única vez, como o ambiente de testes e os algoritmos analisados.

3.1 AMBIENTE DE TESTES

Utilizamos duas máquinas para efetuar nossos testes. Definimos a seguir as configurações de *hardware* e *software* utilizadas. Todos os itens a seguir constituem parâmetros de ambiente.

Temos na Tabela 3 a descrição dos ambientes de *hardware* das servidoras *Achel* e *Latrappe*. Cabe ressaltar que não detemos acesso exclusivo às servidoras. Caberia em trabalho futuro um experimento para analisar o impacto de realizar os mesmos testes em uma máquina isolada, aproveitando melhor assim o uso da memória *cache*. Itens que diferem entre as servidoras são anotados em **negrito**.

Servidoras	
Achel	**Latrappe**
Arquitetura: x86_64;	Arquitetura: x86_64;
CPU(s): 24	**CPU(s): 40**
Thread(s) por núcleo: 2	Thread(s) por núcleo: 2
Core(s) por soquete: 6	Core(s) por soquete: 10
Fabricante: GenuineIntel	Fabricante: GenuineIntel
Família: 6	Família: 6
Modelo: 44	Modelo: 62
CPU MHz: 2.801.000	**CPU MHz: 2.147.468**
Cache L1 (dados): 32K	*Cache* L1 (dados): 32K
Cache L1 (instruções): 32K	*Cache* L1 (instruções): 32K
Cache L2: 256K	*Cache* L2: 256K
Cache L3: 12.288K	*Cache* L3: 25.600K
Memória RAM: 49.453.544 kB	**Memória RAM: 65.941.476 kB**

Tabela 3: Ambientes de hardware utilizados

Temos na Tabela 4 a descrição do ambiente de *software* das servidoras *Achel* e *Latrappe*.

Servidoras
Achel e *Latrappe*
Sistema Operacional: *Linux* Mint 3.14.14-kernel
Compilador: *Python* 2.7.5+ [GCC 4.8.1]
Otimização de Compilador: não

Tabela 4: Ambiente de *software* utilizado

Para medir o Tempo de CPU de cada teste usamos a função *time.time()*, da linguagem *Python*. Vale ressaltar que esta função mede o tempo em que o processo esteve ativo no processador, e não o tempo de relógio decorrido entre o início do processo e sua execução.

Os algoritmos implementados utilizam o sistema SAGE (STEIN et al.,), um sistema de código aberto que oferece suporte a pesquisa e ensino nas áreas de álgebra, geometria, teoria dos números, criptografia, computação numérica e afins.

O número de processos concorrentes ao algoritmo não é registrado por não ser possível deduzir informações precisas a partir deles. A quantidade de processos pode variar durante a execução dos testes, ou seja, podemos nos deparar com a situação em que nenhum dos processos que estavam em execução ao início do teste estejam ainda em execução no segundo seguinte, não sendo esse registro uma visão realista do ambiente em que o teste se encontra.

3.2 CLASSES DE ENTRADA

Definimos a seguir as classes de entrada utilizadas nos experimentos.

3.2.1 Grafos Aleatórios

Grafos $G_{(n,p)}$, onde n representa o número de vértices e p a probabilidade de que cada aresta possível esteja presente (BOLLOBáS, 2001).

3.2.2 DIMACS

O Desafio de Implementação DIMACS (*DIMACS Implementation Challenge*) (JOHNSON; TRICK, 1996b) foi desenvolvido para determinar noções realistas de desempenho para algoritmos que possuem pior caso excessivamente pessimista e modelos probabilísticos pouco realistas.

Em nosso trabalho, usamos as instâncias da segunda edição do desafio.

Foram escolhidos artigos que propõem instâncias com características bem definidas que servem de entrada para o problema computacional estudado e membros de toda a comunidade acadêmica puderam submeter suas soluções para as instâncias fornecidas.

Abaixo temos uma breve descrição de cada família e uma lista das instâncias. Não foram encontradas na literatura referências detalhadas sobre cada família. As informações abaixo limitam-se ao encontrado nos cabeçalhos dos arquivos geradores das instâncias.

- Família c-fat: Grafos aleatórios $C_{n,p}$, onde n é o número de vértices e p é a probabilidade de cada aresta estar presente. As instâncias foram geradas por Michael Trick

usando o programa ggen desenvolvido por Craig Morgenstern (BERMAN; PELC, 1990). Instâncias na Tabela 5.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
c-fat200-1	200	1534	12
c-fat200-2	200	3235	24
c-fat200-5	200	8473	58
c-fat500-1	500	4459	14
c-fat500-2	500	9139	26
c-fat500-5	500	23191	64
c-fat500-10	500	46627	126

Tabela 5: Lista de instâncias da família c-fat.

- Família johnson: Grafos aleatórios gerados por David Johnson (JOHNSON et al., 1991). Instâncias na Tabela 6.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
johnson8-2-4	28	210	4
johnson8-4-4	70	1855	14
johnson16-2-4	120	5460	8
johnson32-2-4	496	107880	16

Tabela 6: Lista de instâncias da família johnson.

- Família MANN: Utiliza a formulação de clique do Problema Steiner triplo, traduzido a partir da formulação da cobertura de conjunto. Instâncias geradas por Carlo Mannino. Instâncias na Tabela 7.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
MANN_a9	45	918	16
MANN_a27	378	70551	126
MANN_a45	1035	533115	345
MANN_a81	3321	5506380	≥ 1100

Tabela 7: Lista de instâncias da família MANN.

- Família brock: Grafos aleatórios com cliques escondidos entre nós com graus relativamente baixos. Instâncias geradas por Mark Brockington e Joe Culberson (BROCKINGTON; CULBERSON, 1996). Instâncias na Tabela 8.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$	*Densidade*
brock200_1	200	14834	21	0,75
brock200_2	200	9876	12	0,50
brock200_3	200	12048	15	0,60
brock200_4	200	13089	17	0,65
brock400_1	400	59723	27	0,75
brock400_2	400	59786	29	0,75
brock400_3	400	59681	31	0,75
brock400_4	400	59765	33	0,75
brock800_1	800	207585	23	0,65
brock800_2	800	208166	24	0,65
brock800_3	800	207333	25	0,65
brock800_4	800	207643	26	0,65

Tabela 8: Lista de instâncias da família brock.

- Família gen: Grafos gerados com cliques grandes de tamanho conhecidos. Instâncias geradas por Laura Sanchis (SANCHIS, 1992). Instâncias na Tabela 9 e 10.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
san200_0.7_1	200	13930	30
san200_0.7_2	200	13930	18
san200_0.9_1	200	17910	70
san200_0.9_2	200	17910	60
san200_0.9_3	200	17910	44
san400_0.5_1	400	39900	13
san400_0.7_1	400	55860	40
san400_0.7_2	400	55860	30
san400_0.7_3	400	55860	22
san400_0.9_1	400	71820	100
san1000	1000	250500	15

Tabela 9: Lista de instâncias san da família gen.

Os grafos sanr pertencem à família gen, são grafos aleatórios de tamanho similar aos san.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
sanr200_0.7	200	13868	18
sanr200_0.9	200	17863	$\geq$42
sanr400_0.5	400	39984	13
sanr200_0.7	400	55869	$\geq$21

Tabela 10: Lista de instâncias sanr da família gen.

- Família hamming: um grafo hamming com parâmetros n e d possui um nó diferente para cada vetor binário de comprimento n. Dois nós são adjacentes se e somente se o vetor correspondente de *bits* possuem distância de hamming de pelo menos d (HASSELBERG; PARDALOS; VAIRAKTARAKIS, 1993). Instâncias na Tabela 11.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
hamming6-2	64	1824	32
hamming6-4	64	704	4
hamming8-2	256	31616	128
hamming8-4	256	20864	16
hamming10-2	1024	518656	512
hamming10-4	1024	434176	40

Tabela 11: Lista de instâncias da família hamming.

- Família keller: Instancias baseadas na conjectura de keller sobre colocação de azulejos usando hipercubos. Essas instâncias foram geradas por Peter Shor (LAGARIAS; SHOR, 1992) e (CORRÁDI; SZABÓ, 1990). Instâncias na Tabela 12.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$
keller4	171	9435	11
keller5	776	225990	27
keller6	3361	4619898	$\geq$59

Tabela 12: Lista de instâncias da família keller.

- Família *p_hat*: Grafos aleatórios que são gerados com o gerador *p_hat* que é uma generalização do clássico gerador uniforme de grafos aleatórios. Grafos gerados com o gerador *p_hat* possuem uma grande faixa de grau de nós e cliques maiores que grafos uniformes gerados por Patrick Soriano e Michel Gendreau (GENDREAU; SORIANO; SALVAIL, 1993).

  Instâncias na Tabela 13.

G	$\|V(G)\|$	$\|E(G)\|$	$\omega(G)$	*Densidade*
p_hat700-3	700	183.010	≥ 62	0,75
p_hat700-2	700	121.728	44	0,50
p_hat700-1	700	60.999	11	0,25
p_hat500-3	500	93.800	≥ 49	0,75
p_hat500-2	500	62.946	36	0,50
p_hat500-1	500	31.569	9	0,25
p_hat300-3	300	33.390	36	0,75
p_hat300-2	300	21.928	25	0,50
p_hat300-1	300	10.933	8	0,25
p_hat1500-3	1.500	847.244	$\geq$94	0,75
p_hat1500-2	1.500	568.960	$\geq$63	0,50
p_hat1500-1	1.000	284.923	12	0,25
p_hat1000-3	1.000	371.743	$\geq$65	0,75
p_hat1000-2	1.000	244.799	$\geq$46	0,50
p_hat1000-1	1.000	122.253	10	0,25

Tabela 13: Lista de instâncias da família p_hat.

3.2.3 Grafos de Moon-Moser

O grafo de Turán é um grafo completo r-partido formado pela divisão de um conjunto de n vértices em r subconjuntos ligando dois vértices por uma aresta sempre que fizerem parte de subconjuntos diferentes. O grafo então possui (n mod r) conjuntos independentes de tamanho $\lceil n/r \rceil$, e $r - (n \bmod r)$ conjuntos de tamanho $\lfloor n/r \rfloor$. Cada vértice tem grau $n - \lceil n/r \rceil$ ou $n - \lfloor n/r \rfloor$. O número de arestas é dado pelo Teorema de Turán $m(n, r) = \lfloor (r-1)n^2/2r \rfloor$. Temos na Figura 2 um grafo Turán M(13,4).

O grafo de Moon-Moser é o caso especial do grafo de Turán em que $r = \lceil n/3 \rceil$. Os grafos de Moon-Moser, $M(n) = T(n, \lceil n/3 \rceil)$, possuem $3^a \times 2^b$ cliques máximas, onde $3a + b = n$ e $b \leq 2$, o maior número possível de cliques entre todos os grafos de n vértices.

Utilizamos os grafos de Moon-Moser por ser o tipo de grafo que possui o maior número de cliques máximas (MOON; MOSER, 1965).

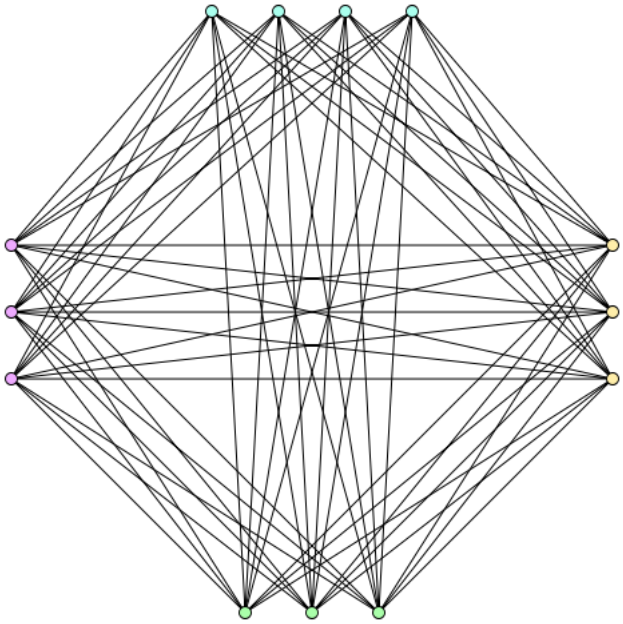

Figura 2: Exemplo de grafo de Turán M(13,4)

3.3 ALGORITMOS ANALISADOS

Apresentamos inicialmente, na Seção 3.3.1 o pseudocódigo do algoritmo BB que demonstra como a técnica de *Branch and Bound* é aplicada para resolver o problema da clique máxima. Em seguida, apresentamos os algoritmos analisados. Mais informações sobre os algoritmos listados abaixo bem como sua definição em um **framework** unificado são encontradas em (ZüGE, 2011) além dos artigos referenciados no final de cada uma das descrições abaixo.

3.3.1 Branch and Bound

Os algoritmos usados nesse trabalho utilizam a técnica de *Branch and Bound*, para encontrar a clique máxima, sendo a mesma subdividida da seguinte forma:

- *Branching*: subdividir o problema principal em problemas menores;
- *Bounding*: eliminar problemas que não levaram à solução ótima através do uso de limitantes que permitem definir quando uma solução não poderá ser a melhor.

A diferença entre os algoritmos apresentados encontra-se na técnica de *bounding*. O passo de branching é o mesmo para todos. Demonstramos a seguir o funcionamento de um algoritmo que encontra a clique máxima sem utilizar nenhuma técnica de *bounding*. Em

seguida, apresentamos qual técnica cada algoritmo apresenta para esse passo. Temos na Figura 3 o pseudocódigo do algoritmo BB:

O algoritmo BB devolve uma clique de tamanho máximo em G que contém Q.

```
BB(G, Q, K)
C ← Q
if K ≠ ∅ then
    v ← um vértice de K
    C1 ←BB(G, Q ∪ {v}, K ∩ Γ_G(v))
    if |C1| >|C| then
        C ← C1
    end if
    C2 ← BB(G, Q, K − {v})
    if |C2| >|C| then
        C ← C2
    end if
end if
return C
```

Figura 3: O Algoritmo BB

O algoritmo BB contém cinco variáveis:

- G: um grafo;
- C: a maior clique encontrada em um determinado momento da execução do algoritmo;
- Q: uma clique no grafo;
- K: o conjunto de vértices candidatos a fazer parte de uma clique;
- v: o vértice pivô, que é adicionado a Q e cujos vizinhos participam na formação de K.

No início da execução, as variáveis apresentam os seguintes estados:

- G: um grafo (dentro do qual buscamos uma clique);
- C: um conjunto vazio, pois nenhuma clique foi encontrado ainda;
- Q: um conjunto vazio, pois nenhuma clique foi encontrado ainda;

- v: vazio, nenhum vértice foi escolhido como pivô;
- K: um conjunto com todos os vértices de G, pois nesse momento todos os vértices são candidatos a uma clique de G.

Na segunda linha, C recebe o conjunto Q. Em seguida, caso ainda tenhamos vértices que possam ser adicionados a clique representada por Q, damos início a um passo de *branching*. Escolhemos um vértice v qualquer de K, na linha 4. Em seguida, fazemos uma chamada recursiva ao BB, usando como argumentos o grafo G inalterado, Q, passando a contar com o vértice v, e K sendo definido como a interseção dos elementos de K e os vizinhos do vértice v.

O que estamos fazendo nesse passo é iniciar a resolução de um subproblema de encontrar uma clique em G que contenha os vértices de Q. Caso essa clique seja maior do que a maior clique que possuímos até agora, armazenado em C, definimos que esta clique passa a ser a maior, nas linhas 6 e 7.

Já na linha 8, criamos um outro subproblema. Uma vez que, na linha 4, ao escolhermos v, estamos dando início à busca de todas as cliques que contenham v, iremos agora dar início à busca de cliques que exclusivamente não contenham v.

Fazemos a mesma análise das linhas 5 e 6 nas linhas 8 e 9 para definirmos se a clique encontrada é a maior até o momento.

Finalmente, quando não houver mais elementos em K, ou seja, não houver vértices que possam fazer parte da maior clique, encerramos a execução do algoritmo retornando a maior clique encontrada, a Clique Máxima C.

O algoritmo resolve o problema computacional de encontrar a maior clique de um grafo do seguinte modo: sempre iniciamos a execução do algoritmo com os conjuntos $Q = \emptyset$ e $K = V(G)$.

Escolhemos então um vértice v do conjunto K. Não há critério para a escolha desse vértice. De forma que sua escolha dar-se-á pela ordenação dos vértices do conjunto K.

Executamos a primeira chamada recursiva a BB onde Q passa a conter v e K passa a ser a vizinhança de v. O que estamos fazendo nesse passo é dar início a busca de todos os cliques de G que possuem v.

A segunda chamada recursiva dá início a busca de novas cliques formadas pelos vértices de G, mas K agora perde o vértice v. Dessa forma, estamos dando início a busca de

todas as cliques que não contém v. Como estamos removendo v de K, quando iniciarmos a execução do BB, escolheremos outro v que não o escolhido anteriormente e iniciaremos a busca das cliques que contém o v atual.

As decisões nas linhas 6 e 10 garantem que C sempre armazenará a maior clique encontrada até um dado momento da execução. A execução será encerrada quando $K = \emptyset$, ou seja, quando não houver mais vértices para os quais não foram enumeradas todas as cliques com a primeira chamada recursiva. Nesse ponto, C conterá a maior clique de G. A Figura 4 apresenta o grafo utilizado como entrada para o algoritmo e a Figura 5 mostra a árvore de busca com todos os Passos de *Branching*.

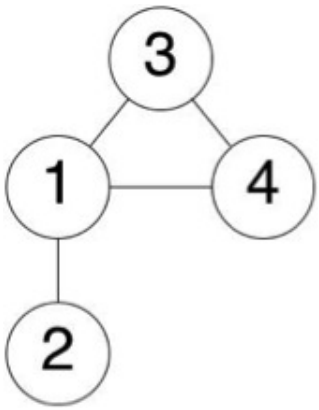

Figura 4: Grafo do algoritmo BB

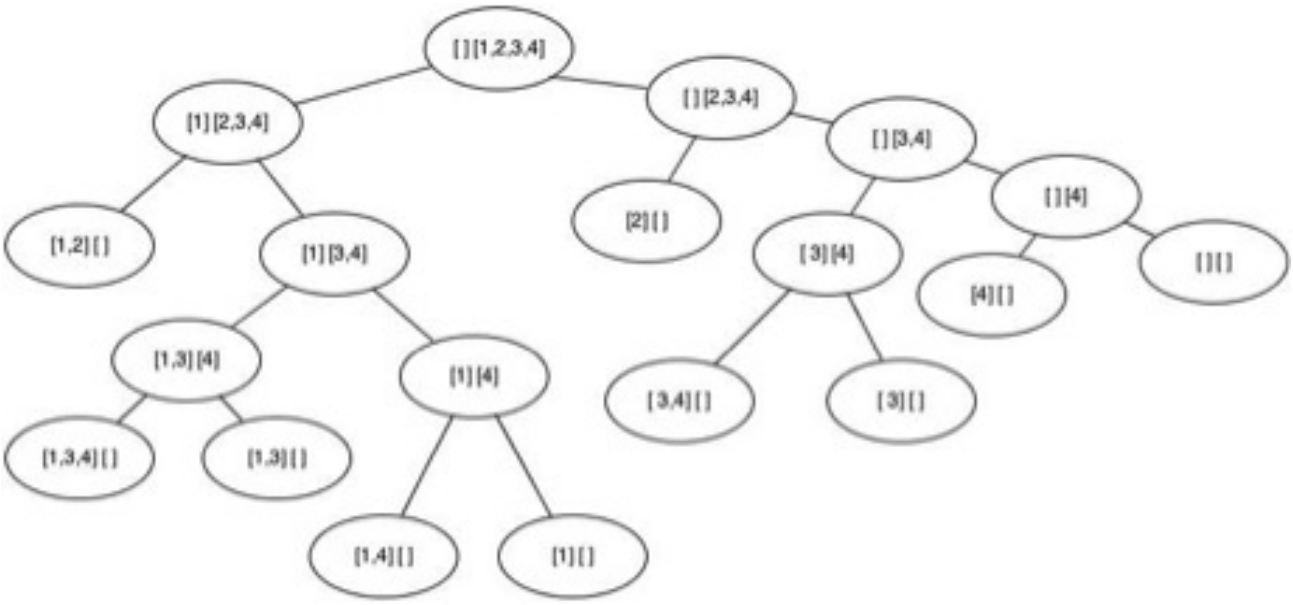

Figura 5: Árvore de Passos de *Branching* gerada pelo algoritmo BB

Percebemos que a execução do algoritmo BB gera uma árvore binária onde o filho direito representa a primeira chamada recursiva (linha 5 do algoritmo) e o filho esquerdo representa a segunda chamada (linha 9). Em cada filho, as variáveis Q e K são representadas pelos valores em colchetes à direita e à esquerda, respectivamente. Cada filho representa um passo de *branching*.

Podemos perceber que as folhas da árvore de busca gerada representam todas as cliques encontradas pelo algoritmo BB e que nas folhas o conjunto K de candidatos encontra-se vazio. Esses filhos são os valores retornados pelas chamadas nas linhas 5 e 9 do algoritmo. As decisões, nas linhas 6 e 10 garantem que a variável C contenha o maior das cliques encontradas, que em nosso exemplo, é o conjunto de vértices 1,3,4.

3.3.2 nobound

O algoritmo nobound é uma instância de BB. Temos assim um algoritmo que gera todos as cliques possíveis, e consequentemente todos os Passos de *Branching* possíveis, para um determinado grafo. Assim, podemos usá-lo como base justa de comparação para com os outros algoritmos, considerando-o o pior caso quando aos indicadores de desempenho escolhidos.

3.3.3 basic

Utiliza como limitante superior o tamanho de K. O algoritmo basic é definido em (ZüGE, 2011).

3.3.4 cp

Ordena os vértices do grafo de forma decrescente em relação ao grau antes do início da execução do algoritmo, ou seja, inicialmente o conjunto K é ordenado seguindo a ordem decrescente de grau dos vértices. O algoritmo cp é definido em (CARRAGHAN; PARDALOS, 1990).

3.3.5 df

Processa Passos de *Branching* que podem ser removidos através da técnica denominada filtro de domínio, onde são removidos de K os vértices que estão obrigatoriamente

na clique e os vértices que não podem fazer parte dela, sendo esses detectados com base em seus graus.

Inicialmente todos os vértices com grau inferior ao mínimo necessário para estar em uma clique de tamanho maior ou igual à clique máxima atual são removidos. Em seguida, os que possuem grau igual a $|K| - 1$ em K são movidos para Q. O algoritmo df é definido em (FAHLE, 2002).

3.3.6 χ

Usa coloração como limitante superior. Duas heurísticas de coloração são utilizadas:

- COLOR: cria-se uma cor de cada vez e vértices são adicionados a esta cor até que não exista nenhum vértice sem vizinho com esta cor.
- DSATUR: a cada vértice é atribuída uma cor por vez, escolhendo o vértice que possuir o maior número de vizinhos já coloridos.

Ambas as heurísticas são aplicadas sobre o conjunto K ordenado de forma não decrescente dos graus dos vértices e também na ordem inversa. Assim, são realizadas um total de 4 colorações para então a melhor ser escolhida. O algoritmo χ é definido em (FAHLE, 2002).

3.3.7 $\chi + \mathsf{df}$

Processa estados em busca de vértices que podem ser removidos de K de acordo com o definido no algoritmo df e usa coloração como limitante superior de acordo com o definido no algoritmo χ. O algoritmo χ + df é definido em (FAHLE, 2002).

3.3.8 mcq

Utiliza uma coloração Gulosa (*Greedy*) onde colore-se cada vértice na ordem em que se apresenta em K, utilizando o menor número de cores possível.

A coloração é usada em dois momentos distintos: para decidir a ordem em que cada vértice será o pivô e como limitante superior. Tal coloração, quando possível, é reaproveitada em diferentes Passos de *Branching* sem necessidade de ser recalculada. O algoritmo mcq é definido em (TOMITA; SEKI, 2003).

3.3.9 dyn

É uma variação do algoritmo mcq anteriormente apresentado, diferenciando em dois pontos:

- No mcq, os vértices das primeiras $|C| - |Q|$ cores nunca são escolhidos como pivô. No algoritmo dyn a ordenação dos vértices após a coloração em cada estado é alterada de modo que todos os vértices de cor menor ou igual a $|C| - |Q|$ mantenham a ordem relativa do estado anterior.

- Ao contrário do algoritmo mcq em que os vértices são ordenados apenas no início da execução, o algoritmo dyn reordena os vértices também em alguns estados antes da coloração. Reordenar os vértices a cada novo Passo de *Branching* pode melhorar o valor do limitante, mas aumenta o tempo de processamento.

O algoritmo dyn é definido em (KONC; JANEŽIČ, 2007).

3.3.10 mcr

Difere de mcq no que diz respeito à ordenação inicial. Durante essa ordenação, é comum chegarmos em um ponto em que todos os vértices ainda não ordenados possuem o mesmo grau, então a ordenação por graus é substituída pela coloração heurística de forma similar ao utilizado no algoritmo mcq. O algoritmo mcr é definido em (TOMITA; KAMEDA, 2007).

3.3.11 mcs

O algoritmo mcs é uma modificação de mcr apresentando diferenças na ordenação inicial dos vértices de forma que os vértices do conjunto K são coloridos sempre na mesma ordem. Além disso, é gerada uma matriz de adjacência com vértices permutados no início da execução do algoritmo, buscando assim um melhor uso da memória *cache*. Também constatam-se mudanças no processo de coloração. Seja $K = |C| - |Q|$, onde vértices com $|K|$ ou menos cores em K são cortados. O algoritmo busca aumentar o número de vértices nas primeiras $|K|$ cores. Além disso, também busca otimizar o uso da memória *cache* através da criação de uma matriz de adjacência do grafo, com os vértices permutados de acordo com a ordenação inicial. O algoritmo mcs é definido em (TOMITA et al., 2010).

Encontramos na Figura 6 (ZüGE, 2011) a ordem de publicação de vários algoritmos *Branch and Bound* encontrados na literatura, onde podemos encontrar os algoritmos acima mencionados. As setas indicam que um algoritmo é uma especialização de outro.

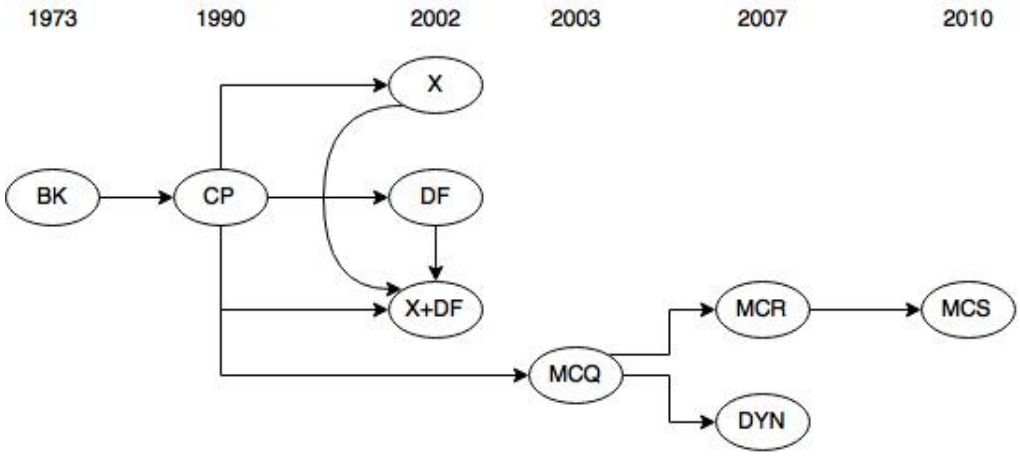

Figura 6: Relação entre os algoritmos e ordem de publicação

3.4 MODELOS DE DESIGN

Apresentamos agora os modelos de *design* definidos para os experimentos separados por classe de grafos. Cada Modelo de *Design* corresponde a um experimento.

As classes de grafos escolhidas procuram mostrar experimentos com características diferentes.

Em todos os casos realizamos os experimentos na servidora *Latrappe* e os validamos através da servidora *Achel*. Todas as vezes em que nos referimos ao indicador de desempenho Tempo de CPU t, usamos a unidade de medida em segundos. Convencionamos T como o número de Passos de *Branching*.

3.4.1 Grafos Aleatórios

Antes de definir o modelo experimental para grafos aleatórios realizou-se um experimento de descoberta na servidora *Latrappe* com o algoritmo nobound, uma vez que por não aplicar nenhum tipo de limitante, espera-se apresentar o maior valor do indicador de desempenho entre todos os algoritmos.

Na Tabela 15, podemos ver quais foram os níveis de n e p definidos para os testes com grafos aleatórios. Para definirmos os valores dos parâmetros de instância (n e p), utilizamos a técnica de *Design* Fatorial Completo descrita na Seção 2.2.3.3.

A definição dos valores na Tabela 15 deu-se da seguinte forma. Fixamos os valores das densidades dos grafos em alta = 0,8 e baixa = 0,1. A densidade baixa de 0,1 é utilizada por ser o extremo de valor baixo para esse parâmetro. Para o valor alto, escolhemos o valor 0,8 por dois motivos. O valor mais apropriado, inicialmente pensando, seria de 1. Entretanto, um grafo aleatório com densidade 1 nada mais é que um grafo completo, removendo assim o aspecto de aleatoriedade quanto às arestas.

Passamos então para a densidade 0,9. Nesta densidade, cada incremento, por menor que seja, no número de vértices, implica em um salto consideravelmente grande no Tempo de CPU, como podemos ver na Tabela 14. Optamos pela densidade de 0,8. Embora ainda não consigamos trabalhar com grafos do tamanho que desejamos, não podemos reduzir ainda mais o nível de p pois buscamos fazer testes com o valor mais alto de n praticável para esse parâmetro.

n	T	t
30	4.465.647	42,7962
60	6.280.541.851	76.698,3716

Tabela 14: Indicadores de desempenho do algoritmo nobound para grafos aleatórios com p=0,9

Com esses níveis de p fixados, definimos os níveis de n da seguinte forma: para p=0,1, qual é o menor nível de n para o qual a execução do experimento de descoberta reportou valor superior a 1 segundo? Tal valor é embasado pelo constante em (BRYANT; O'HALLARON, 2003) que afirma que uma vez que o Tempo de CPU de um processo excede 0,1 segundo o valor reportado possui acurácia de aproximadamente 90%, já que esse valor é 10 vezes mais longo que o intervalo de troca de processos padrão do escalonador de tarefas do processador de 10 milissegundos. Ou seja, se o valor total de execução for menor que 0,1 segundo, o tempo despendido na troca de processos pode interferir de forma significativa na medição do indicador de desempenho. Com base nessa informação, optamos pelo tempo mínimo de execução de 1 segundo, uma vez que estamos definindo níveis de p e n que serão usados com todos os algoritmos através do algoritmo nobound, e prevemos que os demais algoritmos serão consideravelmente mais rápidos que ele. Para a combinação "+-" (n=+ e p=-), buscamos o menor nível de n em que o Tempo de CPU, fixado p=0,1, excedeu 600 segundos, para o mesmo algoritmo.

Para a combinação "-+" a regra é a mesma usada anteriormente, mas agora com o nível de p fixado em 0,8. O mesmo se aplica à combinação "++".

Níveis dos Fatores	n	p
- -	500	0,1
+ -	1750	0,1
- +	30	0,8
+ +	60	0,8

Tabela 15: Valores dos parâmetros de instância dos grafos aleatórios.

Uma vez que a única propriedade que muda entre os testes é a carga do sistema, visando a concisão das informações, optamos por um Modelo de *Design* único para os grafos aleatórios e por registrar individualmente apenas a carga do sistema em cada teste.

Recomenda-se sempre que haja uma motivação por trás da escolha dos valores, mas o importante aqui é descrever o processo em detalhes para facilitar sua reprodução. Por último, precisamos definir quantas vezes executamos cada teste. Os grafos aleatórios possuem um diferencial quanto aos grafos DIMACS e os grafos de Moon-Moser. Cada vez que uma instância é gerada, fornecemos os níveis de n e p, mas não conseguimos precisar com exatidão a quantidade de arestas.

Executamos os testes 5 vezes, depois 10 (5 novos testes somados aos anteriores) e por fim, 20 (10 novos testes somados aos anteriores). Os resultado médio do indicador de desempenho Passos de *Branching*, para todos os pontos de *design*, encontram-se nas Tabelas 16, 17, 18 e 19, todas ordenadas de forma decrescente quanto a número de Passos de *Branching*. Os resultados para o indicador Tempo de CPU, estão nas Tabelas 20, 21, 22 e 23, ordenados por sua vez em ordem decrescente através do indicador.

Percebemos que não existe uma grande variação nos valores dos desvios padrão obtidos para cada algoritmo. Em alguns casos, aumentar o número de testes aumenta o desvio padrão, em outros casos diminui. Buscando trabalhar com uma visão mais realista e abrangente dos grafos aleatórios optamos por um Modelo de *Design* com 20 testes por ponto de *design*.

Ressaltamos que não utilizado

5 Testes			**10 Testes**			**20 Testes**		
(n,p) = (30; 0,8)								
Algoritmo	T	σ	**Algoritmo**	T	σ	**Algoritmo**	T	σ
nobound	126271,4	27,76%	nobound	127064,8	21,06%	nobound	136256,4	37,16%
basic	3317,4	46,57%	basic	3424,2	50,81%	basic	2687,9	54,24%
cp	938,6	24,02%	cp	967	27,34%	cp	931,2	26,36%
df	367,4	48,91%	df	396,2	47,41%	df	322,6	48,92%
dyn	84,6	22,26%	mcq	89,2	19,93%	mcq	90,8	23,12%
mcq	84,6	22,26%	dyn	88	19,17%	dyn	89,9	22,60%
χ	70,6	22,52%	mcs	72,2	51,93%	χ	70,7	19,48%
mcs	62,2	11,94%	χ	69,4	20,32%	mcs	66,4	43,77%
mcr	52,2	19,27%	mcr	57,8	25,28%	mcr	54,7	24,64%
χ + df	48,2	23,94%	χ + df	46,8	19,89%	χ + df	43,7	28,11%

Tabela 16: Valores de T para o ponto de *design* (n, p)= (30, 0,8), com o desvio padrão σ.

5 Testes			10 Testes			20 Testes		
(n,p) = (60; 0,8)								
Algoritmo	T	σ	**Algoritmo**	T	σ	**Algoritmo**	T	σ
nobound	37.313.521,40	32,05%	nobound	37.206.267,20	33,41%	nobound	37.206.267,20	31,70%
basic	234.440,60	43,32%	basic	217.268,60	35,93%	basic	216.822,80	34,00%
cp	54.720,20	34,06%	cp	50.354,20	29,49%	cp	49.238,10	34,72%
df	22.277,40	46,92%	df	19.225,40	42,39%	df	18.561,40	39,49%
χ	1.003,00	27,74%	χ	865,60	36,39%	χ	832,80	42,18%
χ + df	943,80	31,48%	χ + df	802,60	39,96%	χ + df	763,30	45,64%
mcs	724,20	59,82%	mcs	630,40	54,00%	mcs	684,30	61,81%
mcq	638,60	35,49%	mcq	608,80	42,10%	mcq	648,70	40,54%
dyn	571,80	36,46%	dyn	522,00	37,36%	dyn	572,10	41,20%
mcr	561,40	68,44%	mcr	473,60	62,90%	mcr	532,10	62,21%

Tabela 17: Valores de T para o ponto de *design* (n, p) = (60, 0,8), com o desvio padrão σ.

5 Testes			**10 Testes**			**20 Testes**		
(n,p) = (500; 0,1)								
Algoritmo	T	σ	**Algoritmo**	T	σ	**Algoritmo**	T	σ
nobound	71.517,40	1,19%	nobound	71.946,00	2,01%	nobound	71.903,30	2,32%
basic	29.143,00	0,86%	basic	29.324,80	1,96%	basic	29.280,90	2,15%
cp	26.150,60	1,26%	cp	26.388,80	1,36%	cp	26.318,80	2,05%
dyn	1.223,00	2,40%	mcq	1.261,40	5,68%	mcq	1.255,80	5,45%
mcq	1.216,00	2,50%	dyn	1.261,00	5,60%	dyn	1.254,60	5,34%
mcr	1.202,60	3,38%	mcr	1.245,40	6,04%	mcr	1.243,00	5,70%
df	1.172,60	13,20%	df	1.155,20	9,24%	mcs	1.154,80	3,23%
mcs	1.162,20	5,75%	mcs	1.154,20	4,35%	df	1.148,70	7,74%
χ + df	955,00	11,65%	χ	971,40	9,15%	χ	972,70	7,07%
χ	939,50	5,31%	χ + df	928,40	8,65%	χ + df	929,70	6,71%

Tabela 18: Valores de T para o ponto de *design* (n, p) = (500, 0,1), com o desvio padrão σ.

5 Testes			**10 Testes**			**20 Testes**		
(n,p) = (1.750; 0,1)								
Algoritmo	T	σ	**Algoritmo**	T	σ	**Algoritmo**	T	σ
nobound	2.896.274,20	0,46%	nobound	2.901.524,60	0,46%	nobound	2.901.369,50	0,80%
basic	1.178.691,40	0,67%	basic	1.178.808,40	0,51%	basic	1.177.123,80	0,89%
χ	85.518,60	0,91%	χ	85.578,00	0,65%	χ	85.577,70	1,25%
χ + df	83.781,80	1,03%	χ + df	83.895,40	0,83%	χ + df	83.750,80	1,24%
cp	1.094.189,40	0,43%	cp	1.097.726,40	0,59%	cp	1.098.497,50	1,03%
df	114.734,60	0,31%	df	114.872,60	0,35%	df	114.745,60	0,77%
dyn	50.377,00	1,07%	dyn	50.834,20	1,38%	dyn	50.996,80	2,00%
mcq	50.392,20	1,13%	mcq	50.856,60	1,42%	mcq	51.026,00	2,11%
mcr	48.891,00	0,71%	mcr	49.209,40	1,43%	mcr	49.314,50	1,80%
mcs	41.683,80	1,53%	mcs	41.624,60	1,17%	mcs	41.488,40	1,58%

Tabela 19: Valores de T para o ponto de *design* (n, p) = (1.750, 0,1), com o desvio padrão σ.

5 Testes			**10 Testes**			**20 Testes**		
(n,p) = (30; 0,8)								
Algoritmo	**tempo**	σ	**Algoritmo**	**tempo**	σ	**Algoritmo**	**tempo**	σ
nobound	1,2673	26,65%	nobound	1,2796	20,32%	nobound	1,3608	36,40%
χ	0,1640	21,82%	χ	0,1607	20,43%	χ	0,1633	23,36%
χ + df	0,1383	26,78%	χ + df	0,1331	25,90%	χ + df	0,1292	33,46%
df	0,0642	32,30%	df	0,0690	34,81%	df	0,0600	33,20%
basic	0,0383	41,27%	basic	0,0397	43,49%	basic	0,0322	45,46%
mcs	0,0202	4,75%	mcs	0,0208	15,68%	mcs	0,0204	12,54%
mcr	0,0175	1,78%	mcr	0,0179	5,62%	mcr	0,0178	4,65%
cp	0,0126	21,91%	cp	0,0129	22,95%	cp	0,0126	21,57%
dyn	0,0054	12,81%	dyn	0,0058	16,65%	dyn	0,0060	17,50%
mcq	0,0053	12,08%	mcq	0,0058	18,08%	mcq	0,0059	18,34%

Tabela 20: Valores de t para o ponto de *design* (n, p) = (30, 0,8), com o desvio padrão σ.

5 Testes			**10 Testes**			**20 Testes**		
(n,p) = (60; 0,8)								
Algoritmo	**tempo**	σ	**Algoritmo**	**tempo**	σ	**Algoritmo**	**tempo**	σ
nobound	452,7087	31,78%	nobound	474,0639	31,78%	nobound	453,8888	31,21%
χ + df	8,8699	33,08%	χ + df	8,5485	54,74%	χ + df	7,7465	51,64%
χ	8,4649	31,23%	χ	8,3216	55,08%	χ	7,6019	51,17%
df	7,1725	43,11%	df	6,3126	38,21%	df	6,1998	32,63%
basic	3,4703	39,99%	basic	3,2642	33,39%	basic	3,2681	31,02%
cp	0,8647	30,50%	cp	0,8070	25,60%	cp	0,7869	29,40%
mcs	0,3053	36,05%	mcs	0,2801	29,67%	mcs	0,2907	32,36%
mcr	0,2052	26,58%	mcr	0,1925	21,57%	mcr	0,1995	22,71%
mcq	0,1084	31,79%	mcq	0,1021	33,75%	mcq	0,1077	34,46%
dyn	0,1067	38,16%	dyn	0,0982	36,15%	dyn	0,1041	35,44%

Tabela 21: Valores de t para o ponto de *design* (n, p) = (60, 0,8), com o desvio padrão σ.

5 Testes			**10 Testes**			**20 Testes**		
(n,p) = (500; 0,1)								
Algoritmo	**tempo**	σ	**Algoritmo**	**tempo**	σ	**Algoritmo**	**tempo**	σ
χ	111,4859	3,67%	χ	110,7691	2,74%	χ	115,8604	20,33%
χ + df	102,9952	3,39%	χ + df	102,0534	2,66%	χ + df	106,4829	19,67%
mcs	7,5556	2,61%	mcs	7,5874	1,91%	mcs	7,5728	1,85%
mcr	7,4954	2,49%	mcr	7,5260	1,88%	mcr	7,5170	1,84%
nobound	3,9801	3,80%	nobound	3,9919	2,88%	nobound	4,0021	4,01%
df	1,8853	3,07%	df	1,8925	2,24%	df	1,8921	2,47%
cp	0,8756	3,01%	cp	0,8754	2,13%	cp	0,8752	2,35%
basic	0,7115	3,34%	basic	0,7132	2,42%	basic	0,7123	2,57%
dyn	0,4206	3,47%	dyn	0,4232	2,61%	dyn	0,4217	2,25%
mcq	0,4203	2,61%	mcq	0,4221	2,04%	mcq	0,4201	2,04%

Tabela 22: Valores de t para o ponto de *design* (n, p) = (500, 0,1), com o desvio padrão σ.

5 Testes			10 Testes			20 Testes		
(n,p) = (1750; 0,1)								
Algoritmo	tempo	σ	Algoritmo	tempo	σ	Algoritmo	tempo	σ
χ	26.095,601	2,42%	χ	26.079,355	1,78%	χ	26.071,15	1,49%
χ + df	24.234,354	2,15%	χ + df	24.158,118	1,64%	χ + df	24.028,22	1,60%
nobound	494,031	2,33%	nobound	495,973	1,75%	nobound	498,19	1,56%
mcs	337,772	1,81%	mcs	339,556	1,72%	mcs	340,31	1,34%
mcr	333,734	1,74%	mcr	335,403	1,71%	mcr	336,26	1,36%
df	86,731	2,23%	df	87,138	1,84%	df	87,29	1,51%
cp	29,484	1,93%	cp	29,644	1,66%	cp	29,70	1,43%
basic	29,183	2,24%	basic	29,301	1,75%	basic	29,35	1,52%
dyn	17,256	3,30%	dyn	17,917	8,55%	dyn	18,34	13,67%
mcq	16,456	1,81%	mcq	16,606	1,92%	mcq	16,66	1,82%

Tabela 23: Valores de t para o ponto de *design* (n, p) = (500, 0,1), com o desvio padrão σ.

A Tabela 24 mostra o Modelo de *Design* Experimental aplicado como definido no Capítulo 2:

Design Experimental	
Servidora:	*Latrappe* e *Achel* (validação)
Carga:	registrada isoladamente em cada execução
Pergunta:	Qual o comportamento dos algoritmos nobound, basic, cp, df, χ, $\chi + \mathsf{df}$, mcq, dyn, mcr e mcs quando utilizados para encontrar a clique máxima em grafos aleatórios?
Indicador de desempenho:	Tempo de CPU em conjunto com o número de Passos de *Branching*.
Fatores:	n e p
Níveis:	Níveis: $n \in \{30, 60, 500, 1750\}$ e $p \in \{0,1; 0,8\}$
Testes:	20 testes por ponto de *design*
Pontos de Design:	*Design* Fatorial Completo sobre $(n; p) \in \{(500; 0,1), (1750; 0,1), (30; 0,8), (60; 0,8)\}$.
Saídas:	tamanho da Clique Máxima ω, número de Passos de *Branching* T e Tempo de CPU t.

Tabela 24: Modelo de *Design* experimental.

Os resultados do experimento realizado utilizando este Modelo de *Design* encontram-se na Seção 4.1.

3.4.2 Família DIMACS

Os grafos DIMACS foram submetidos a experimentos com um Modelo de *Design* bastante semelhante ao dos grafos aleatórios. O que difere neste caso em particular é a ausência de fatores, por tratar-se de um *TestBed* público. O tempo máximo de CPU foi definido da seguinte forma: executamos um experimento de descoberta como ilustrado no fluxograma da Figura 7. O resultado de tal experimento encontra-se na Tabela 25, onde é anotado com *timeout* o teste que foi interrompido pelo tempo limite e "concluiu" o teste que foi encerrado antes do tempo limite. Aumentos no tamanho da maior clique encontrada em relação à iteração anterior são anotados em **negrito.**

O experimento é executado sobre todas as instâncias usando o algoritmo nobound, cujos resultados utilizamos como "limitante superior". Após executar testes para todas as instâncias utilizamos o maior tempo registrado como tempo limite de execução.

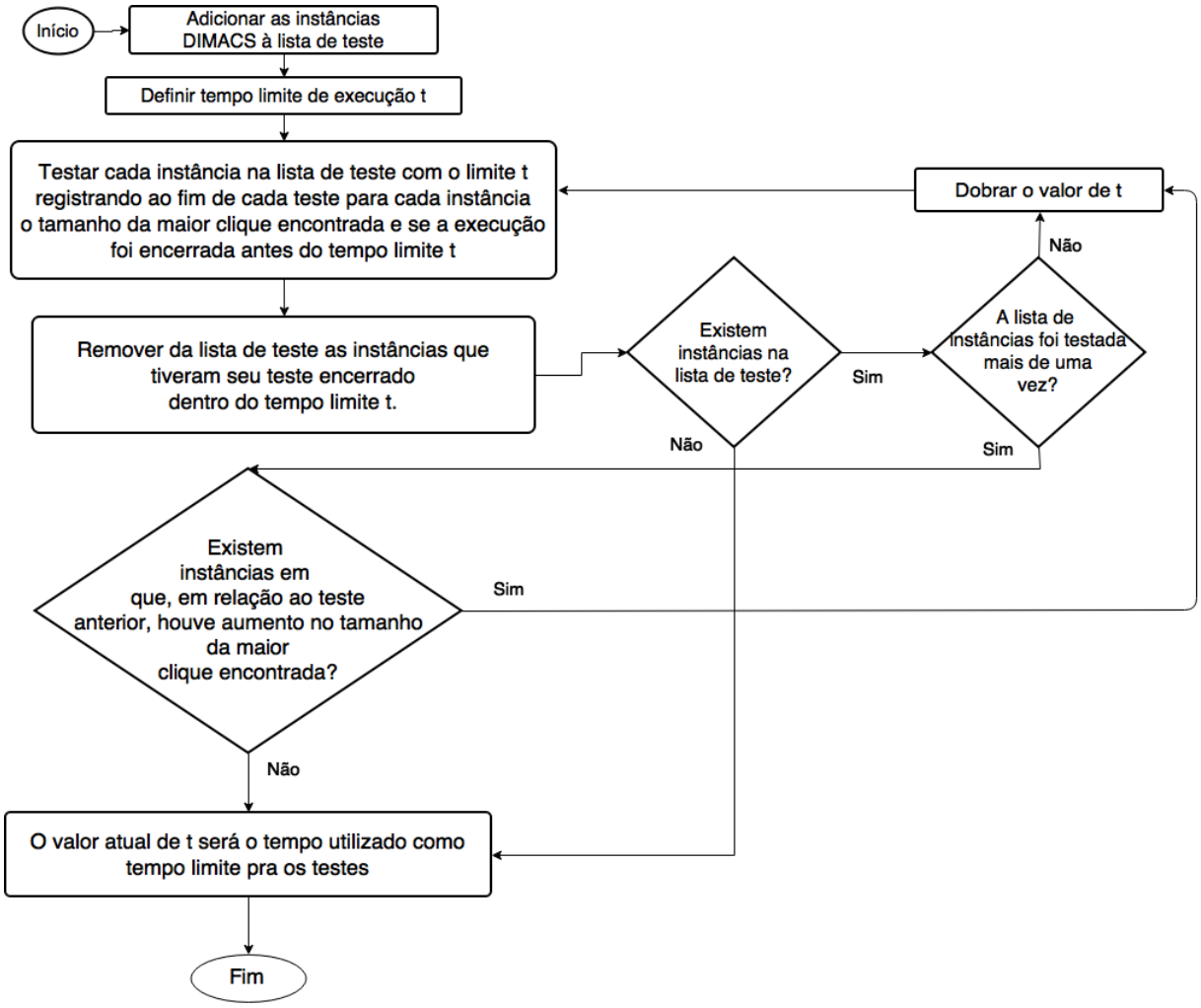

Figura 7: Fluxograma do Experimento de Descoberta

Tabela 25: Resultado do experimento de descoberta para a família de grafos DIMACS

Instância	**t=600**	ω	**t=1.200**	ω	**t=1.800**	ω	**t=2.400**	ω	**t=4.800**	ω	**t=9.600**	ω	**t=19.200**	ω	**t=38.400**	ω
brock200_1	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	**19**	*timeout*	19	*timeout*	19	*timeout*	19
brock200_2	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12
brock200_3	*timeout*	13	*timeout*	13	*timeout*	13	*timeout*	**14**	*timeout*	14	*timeout*	14	*timeout*	**15**	concluiu	15
brock200_4	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	**16**	*timeout*	16
brock400_1	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	**22**	*timeout*	22	*timeout*	22	*timeout*	22
brock400_2	*timeout*	20	*timeout*	20	*timeout*	21	*timeout*	**22**	*timeout*	22	*timeout*	22	*timeout*	22	*timeout*	22
brock400_3	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	22	*timeout*	22
brock400_4	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	**21**	*timeout*	21	*timeout*	21	*timeout*	21
brock800_1	*timeout*	17	*timeout*	17	*timeout*	**18**	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	18
brock800_2	*timeout*	17	*timeout*	17	*timeout*	17	*timeout*	**18**	*timeout*	18	*timeout*	18	*timeout*	**19**	*timeout*	19
brock800_3	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	18	*timeout*	**19**	*timeout*	19
brock800_4	*timeout*	18	*timeout*	18	*timeout*	**19**	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19
c-fat200-1	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12	concluiu	12
c-fat200-2	*timeout*	24	*timeout*	24	*timeout*	24	*timeout*	24	*timeout*	24	concluiu	24	concluiu	24	concluiu	24
c-fat200-5	*timeout*	58	*timeout*	58	*timeout*	58	*timeout*	58	*timeout*	58	*timeout*	58	*timeout*	58	*timeout*	58
c-fat500-1	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14
c-fat500-10	*timeout*	126	*timeout*	126	*timeout*	126	*timeout*	126	*timeout*	126	*timeout*	126	*timeout*	126	*timeout*	126
c-fat500-2	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26
c-fat500-5	*timeout*	64	*timeout*	64	*timeout*	64	*timeout*	64	*timeout*	64	*timeout*	64	*timeout*	64	*timeout*	64
hamming10-2	*timeout*	512	*timeout*	512	*timeout*	512	*timeout*	512	*timeout*	512	*timeout*	512	*timeout*	512	*timeout*	512
hamming10-4	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32
hamming6-2	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32
hamming6-4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4
hamming8-2	*timeout*	128	*timeout*	128	*timeout*	128	*timeout*	128	*timeout*	128	*timeout*	128	*timeout*	128	*timeout*	128
hamming8-4	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16
johnson16-2-4	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	concluiu	8	concluiu	8	concluiu	8	concluiu	8
johnson32-2-4	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16

Tabela 25 – continua na próxima página

Tabela 25 – continuação da página anterior

Instância	**t=600**	ω	**t=1.200**	ω	**t=1.800**	ω	**t=2.400**	ω	**t=4.800**	ω	**t=9.600**	ω	**t=19.200**	ω	**t=38.400**	ω
johnson8-2-4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4	concluiu	4
johnson8-4-4	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14	concluiu	14
keller4	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	concluiu	11	concluiu	11	concluiu	11	concluiu	11
keller5	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21
keller6	*timeout*	40	*timeout*	40	*timeout*	40	*timeout*	40	*timeout*	40	*timeout*	40	*timeout*	**42**	*timeout*	42
MANN_a27	*timeout*	125	*timeout*	125	*timeout*	125	*timeout*	125	*timeout*	125	*timeout*	125	*timeout*	125	*timeout*	125
MANN_a45	*timeout*	340	*timeout*	340	*timeout*	340	*timeout*	340	*timeout*	340	*timeout*	340	*timeout*	340	*timeout*	**341**
MANN_a81	*timeout*	1096	*timeout*	1096	*timeout*	1096	*timeout*	1096	*timeout*	1096	*timeout*	1096	*timeout*	1096	*timeout*	1096
MANN_a9	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	concluiu	16	concluiu	16	concluiu	16	concluiu	16
p_hat1000-1	*timeout*	10	*timeout*	10	*timeout*	10	*timeout*	10	*timeout*	10	*timeout*	10	concluiu	10	concluiu	10
p_hat1000-2	*timeout*	21	*timeout*	21	*timeout*	**22**	*timeout*	22	*timeout*	22	*timeout*	**23**	*timeout*	**24**	*timeout*	24
p_hat1000-3	*timeout*	31	*timeout*	31	*timeout*	31	*timeout*	31	*timeout*	**33**	*timeout*	**35**	*timeout*	35	*timeout*	35
p_hat1500-1	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11
p_hat1500-2	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	32	*timeout*	**33**	*timeout*	33	*timeout*	33
p_hat1500-3	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46
p_hat300-1	concluiu	8	concluiu	8	concluiu	8	concluiu	8	concluiu	8	concluiu	8	concluiu	8	concluiu	8
p_hat300-2	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	**22**	*timeout*	22	*timeout*	22
p_hat300-3	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	26	*timeout*	**29**	*timeout*	29	*timeout*	29
p_hat500-1	concluiu	9	concluiu	9	concluiu	9	concluiu	9	concluiu	9	concluiu	9	concluiu	9	concluiu	9
p_hat500-2	*timeout*	27	*timeout*	27	*timeout*	27	*timeout*	27	*timeout*	27	*timeout*	**28**	*timeout*	28	*timeout*	28
p_hat500-3	*timeout*	34	*timeout*	34	*timeout*	34	*timeout*	34	*timeout*	34	*timeout*	**35**	*timeout*	37	*timeout*	**38**
p_hat700-1	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	*timeout*	11	concluiu	11	concluiu	11	concluiu	11
p_hat700-2	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	21	*timeout*	**23**	*timeout*	23	*timeout*	**24**	*timeout*	24
p_hat700-3	*timeout*	33	*timeout*	33	*timeout*	33	*timeout*	33	*timeout*	33	*timeout*	33	*timeout*	**34**	*timeout*	34
san1000	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	**9**	*timeout*	9
san200_0.7_1	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16
san200_0.7_2	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	**13**	*timeout*	14	*timeout*	14
san200_0.9_1	*timeout*	39	*timeout*	39	*timeout*	39	*timeout*	39	*timeout*	39	*timeout*	39	*timeout*	39	*timeout*	39

Tabela 25 – continua na próxima página

Tabela 25 – continuação da página anterior																
Instância	**t=600**	ω	**t=1.200**	ω	**t=1.800**	ω	**t=2.400**	ω	**t=4.800**	ω	**t=9.600**	ω	**t=19.200**	ω	**t=38.400**	ω
san200_0.9_2	*timeout*	30	*timeout*	30	*timeout*	**31**	*timeout*	31	*timeout*	31	*timeout*	31	*timeout*	**32**	*timeout*	32
san200_0.9_3	*timeout*	28	*timeout*	28	*timeout*	28	*timeout*	28	*timeout*	**29**	*timeout*	29	*timeout*	**30**	*timeout*	30
san400_0.5_1	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	8	*timeout*	**9**	*timeout*	9
san400_0.7_1	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20	*timeout*	20
san400_0.7_2	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15	*timeout*	15
san400_0.7_3	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	16	*timeout*	**17**	*timeout*	17	*timeout*	17	*timeout*	17
san400_0.9_1	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	46	*timeout*	**47**	*timeout*	**48**	*timeout*	48
sanr200_0.7	*timeout*	17	*timeout*	17	*timeout*	17	*timeout*	17	*timeout*	17	*timeout*	17	*timeout*	**18**	*timeout*	18
sanr200_0.9	*timeout*	35	*timeout*	35	*timeout*	35	*timeout*	35	*timeout*	35	*timeout*	**36**	*timeout*	**37**	*timeout*	37
sanr400_0.5	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	12	*timeout*	**13**	*timeout*	13
sanr400_0.7	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19	*timeout*	19

Tabela 25: Resultado do experimento de descoberta para a família de grafos DIMACS

O tempo inicial foi de 600 segundos. Após 8 iterações, apenas 2 instâncias apresentaram aumento no tamanho da maior clique encontrada até o encerramento da execução, MANN_a45 e p_hat500-3. Entretanto, incrementar o tempo para 76.800 segundos (aproximadamente 21 horas) torna-se impraticável para a quantidade de tempo que dispomos. Assim sendo, o tempo máximo definido para todos os testes foi de 38.400 segundos. As características de cada uma das instâncias encontram-se na Seção 3.2.2.

Os experimentos foram conduzidos na servidora *Latrappe* e posteriormente validados na servidora *Achel*. Neste caso estamos usando um TestBed Público. Por esse motivo, para as instâncias DIMACS não se aplicam os conceitos de fatores e níveis, uma vez que esses valores não são explicitamente manipulados. Aquele que desejar comparar seus resultados com os obtidos aqui deverá apenas utilizar a mesma instância. O Modelo de *Design* para os grafos DIMACS encontra-se na Tabela 26.

Como a quantidade de testes é bastante elevada, 66 instâncias para serem executadas por 10 algoritmos, gerando 660 testes, optamos por realizar apenas um teste para cada instância sobre cada algoritmo. Seria ideal executar todos os testes pelo menos três vezes, com objetivo de validação, mas infelizmente não dispomos do tempo necessário. Além disso, como explicamos no início da Seção 4, os testes com os grafos DIMACS demandaram mais tempo do que havíamos previsto inicialmente.

Ainda com a grande quantidade de testes em mente, optamos por utilizar uma amostra das instâncias para efetuarmos testes de validação na servidora *Achel*. Mais informações encontram-se na Seção 4.2.

Design Experimental	
Servidora:	*Latrappe* e *Achel* (validação)
Carga:	registrada isoladamente em cada execução
Pergunta:	Qual o comportamento dos algoritmos nobound, basic, cp, df, χ, $\chi + \mathsf{df}$, mcq, dyn, mcr e mcs quando utilizados para encontrar a clique máxima nos grafos DIMACS?
Indicador de desempenho:	Tempo de CPU em conjunto com o número de Passos de *Branching*
Fatores:	Não aplicável
Níveis:	Não aplicável
Testes:	1 teste por ponto de *design*
Pontos de Design:	Não aplicável
Saídas:	tamanho da clique máxima ω, número de Passos de *Branching* T, tempo total de execução t e se a execução do algoritmo foi encerrada dentro do tempo limite

Tabela 26: Design Experimental - Grafos DIMACS.

Os resultados do experimento realizado utilizando este Modelo de *Design* encontram-se na Seção 4.2, DIMACS.

3.4.3 Grafos de Moon-Moser

Na Tabela 28 apresentamos o Modelo de *Design* definido inicialmente. Os grafos de Moon-Moser são explicados em detalhe na Seção 3.2.3.

Para definirmos o nível de n, assim como nos grafos aleatórios, realizamos um experimento de descoberta para verificar como a variação de n afeta o Tempo de CPU para o algoritmo nobound, que usamos como limitante superior.

Na Tabela 27 podemos ver o resultado desse experimento.

n	ω	T	t
5	2	23	0,0002
10	4	287	0,0025
15	5	2.047	0,0193
20	7	24.575	0,2262
25	9	294.911	3,1109
30	10	2.097.151	22,4024
35	12	25.165.823	273,7484
40	14	301.989.887	3.304,1318
45	15	2.147.483.647	26.940,2687
50	17	25.769.803.775	345.611,4283

Tabela 27: Experimento de descoberta para os grafos de Moon-Moser com o algoritmo nobound.

Como possuímos apenas um fator, o número de vértices n, para aplicarmos o *Design* Fatorial Completo devemos apenas especificar dois níveis para n, um valor para o qual esperamos que os valores das saídas sejam baixos e outro em que se espera que os valores das saídas sejam altos. Como o nível de n é crescente com os níveis obtidos dos indicadores de performance, precisamos apenas definir dois níveis, alto e baixo, para o mesmo.

Definimos então como baixo o nível de 25, por ser o menor resultado acima de 1 segundo (BRYANT; O'HALLARON, 2003). Já para o nível alto, selecionamos $n = 45$ por ser o maior nível de n praticável. O próximo nível de n tem Tempo de CPU de aproximadamente 4 dias para cada teste.

Design Experimental	
Servidora:	*Latrappe* e *Achel* (validação)
Carga:	registrada isoladamente em cada execução
Pergunta:	Qual o comportamento dos algoritmos nobound, basic, cp, df, χ, χ + df , mcq, dyn, mcr e mcs quando utilizados para encontrar a clique máxima nos grafos de Moon-Moser?
Indicador de desempenho:	Tempo de CPU em conjunto com o número de Passos de *Branching*
Fatores:	n
Níveis:	$n \in \{25, 45\}$
Testes:	10 testes por ponto de *design*.
Pontos de *Design*:	$n \in \{25, 45\}$
Saídas:	tamanho da Clique Máxima ω, número de Passos de *Branching* T e tempo total de execução t.

Tabela 28: *Design* Experimental - Grafos de Moon-Moser.

Inicialmente mostramos nas Tabelas 29 e 30 a quantidade de Passos de *Branching* e Tempo de CPU médio com desvio padrão σ para cada algoritmo, em ordem crescente de Passos de *Branching*.

Não mostramos valores médios para T pois todas os testes retornaram exatamente os mesmos valores, dada a natureza do tipo de grafo.

n=**25**	T	t	σ
χ	19	0,0455	0,1261%
χ + df	19	0,0479	0,0240%
dyn	19	0,0065	0,0013%
mcr	31	0,0090	0,0028%
mcs	31	0,0094	0,0015%
mcq	49	0,0037	0,0026%
df	5.833	0,2823	0,1234%
cp	17.495	0,1531	0,0531%
basic	84.155	0,6402	0,4779%
nobound	294.911	3,2480	0,1246%

Tabela 29: Número de Passos de *Branching* e Tempo de CPU, com desvio padrão para n=25

n=**45**	T	t	σ
χ	31	0,191	0,101%
χ + df	31	0,209	0,395%
dyn	31	0,031	0,047%
mcr	31	0,006	0,004%
mcs	31	0,007	0,005%
mcq	89	0,016	0,025%
df	9.565.939	462,479	1,454%
cp	28.697.813	306,140	1,572%
basic	318.792.469	2.713,115	27,715%
nobound	2.147.483.647	26.940,269	22,876%

Tabela 30: Número de Passos de *Branching* e Tempo de CPU, com desvio padrão para n=45

Através das Tabelas 29 e 30, percebemos o esperado quanto aos algoritmos nobound e basic, tendo sido os de pior desempenho de uma forma bastante expressiva. Os algoritmos df e cp também mostraram um desempenho bastante aquém dos seus concorrentes em

ambos os casos.

Já quanto aos outros algoritmos, não é possível concluir qual é o melhor com base apenas nesses dados. Isso se deve ao fato de que todos apresentam um resultado próximo, ou idêntico, como é o caso de n=45, onde 5 algoritmos apresentaram exatamente o mesmo resultado para T.

Decidimos então realizar um novo Experimento de Descoberta, mas desta vez excluindo os algoritmos nobound, basic df e cp, por terem apresentado indicadores de desempenho de valor bastante elevado em relação aos demais algoritmos. Assim, temos a possibilidade de trabalhar com níveis mais altos de n, que podem evidenciar a diferença de desempenho dos demais algoritmos restantes.

Para tanto procedemos da seguinte forma: executamos cada algoritmo com valor inicial de 200, incrementando n em 50 a cada novo teste até que o Tempo de CPU fosse superior a 6.000 segundos.

Os menores níveis de n em que a execução de cada algoritmo ultrapassou 6.000 segundos de Tempo de CPU estão na planilha 31:

Algoritmo	n	ω	T	t
χ	1350	450	901	6280,00
χ + df	1350	450	901	6543,18
dyn	3400	1134	6799	6639,83
mcq	3300	1100	6599	6077,62
mcr	2150	717	2865	6267,47
mcs	2000	667	2665	6264,49

Tabela 31: Menores níveis de n em que a execução cada algoritmo ultrapassou 6.000 segundos.

Alteramos então o Modelo de *Design* da Tabela 28 para o mostrado na Tabela 32. O valor de nível baixo para n foi definido como 200 uma vez que foi o menor nível de n em que o Tempo de CPU de todos os algoritmos foi superior a 1 segundo. O valor de nível alto utilizado no Modelo de *Design* foi definido escolhendo o menor nível de n dentre os presentes na Tabela 31 para que possamos usar o mesmo nível de n para todos os algoritmos dentro do limite máximo de tempo de 6.000 segundos.

Design Experimental	
Servidora:	*Achel* e *Latrappe*
Carga:	registrada isoladamente em cada execução
Pergunta:	Qual o comportamento dos algoritmos, χ,$\chi + \mathsf{df}$, mcq, dyn, mcr e mcs quando utilizados para encontrar a Clique Máxima nos grafos de Moon-Moser?
Indicador de desempenho:	Tempo de CPU em conjunto com o número de Passos de *Branching*
Fatores:	n
Níveis:	$n \in \{200, 1350\}$
Testes:	10 testes por ponto de *design*.
Pontos de Design:	$n \in \{200, 1350\}$
Saídas:	tamanho da Clique Máxima ω, número de Passos de *Branching* T e tempo total de execução t.

Tabela 32: Design Experimental - Grafos de Moon-Moser.

Os resultados do experimento realizado utilizando este Modelo de *Design* encontram-se na Seção 4.3, Moon-Moser.

4 RESULTADOS

Apresentamos os resultados experimentais obtidos de acordo com os modelos de design definidos na Seção 3.4.

Dentro deste capítulo, cada família de grafo recebe sua própria seção, e cada seção é dividida de acordo com os indicadores de desempenho definidos: número de Passos de *Branching* e Tempo de CPU. A exceção é a classe de entrada DIMACS, em que foram analisadas outras características, além dos indicadores de desempenho mencionados anteriormente. Na Seção 4.1 apresentamos os resultados dos grafos aleatórios. Na Seção 4.2 os grafos DIMACS e, por último, na Seção 4.3 os grafos de Moon-Moser.

O algoritmo nobound é o único que não emprega técnica de *bounding*, como descrito na Seção 3.3.2, e é utilizado aqui para representar o valor máximo dos indicadores de desempenho nos testes.

Na análise do indicador de desempenho Passos de *Branching*, apresenta melhor desempenho aquele algoritmo que gerou menos passos. No caso do indicador Tempo de CPU, aquele que apresentou menor tempo.

Após analisarmos os indicadores separadamente, fazemos sua análise conjunta. Buscamos entender de que forma o algoritmo divide seu Tempo de CPU total entre as tarefas de *branching* e *bounding* dividindo t por T. Poderia também ter sido empregado o cálculo inverso de T/t, fornecendo assim uma visão diferente sobre o resultado, mas sem modificar as conclusões finais. Um resultado de valor baixo significa que dentro do Tempo de CPU reportado, tal algoritmo conseguiu gerar uma grande quantidade de Passos de *Branching*. Um valor alto, significa que foram gerados poucos Passos de *Branching*, e o restante do tempo foi consumido pela tarefa de cálculo do valor limitante empregado na técnica de *bounding*. Utilizamos os resultados desse cálculo para procurar entender como a estratégia de cada algoritmo para a redução de número de Passos de *Branching* afeta seu desempenho geral durante sua execução.

Foi registrada a quantidade de memória RAM livre para os experimentos em todas as classes de entrada. O registro foi feito no início e no fim do teste de cada algoritmo sobre todas as instâncias.

Uma vez que os testes demandam uma grande quantidade de tempo para serem executados, pode ter havido variações significativas de consumo de memória durante os

testes. Todos os testes para cada algoritmo foram executados de forma sequencial sem interrupções.

Infelizmente, não dispomos de um *hardware* como das servidoras *Achel* e *Latrappe* de forma exclusiva.

Quando algum evento de força maior interrompesse os testes para um algoritmo, os resultados parciais foram descartados e os testes iniciados novamente. Todas as vezes em que um novo conjunto de testes era iniciado para um algoritmo, verificou-se o *hardware* da máquina para constatar que se mantinha inalterado.

No caso dos grafos DIMACS, em que foram executados uma grande quantidade de testes, tivemos que reiniciar os testes uma grande quantidade de vezes dado que as as servidoras eram reiniciadas ou desativadas para manutenção, o que acarretou em uma demanda de tempo bastante superior do que havíamos previsto e o que provocou a decisão de fazer apenas um teste para cada ponto de *design*.

4.1 GRAFOS ALEATÓRIOS

Temos nesta seção os resultados para a família de grafos aleatórios. Tais resultados são referentes à servidora *Latrappe* e validados na servidora *Achel* e foram obtidos através da aplicação do Modelo de *Design* definido na Seção 3.4.1. Uma definição detalhada de cada instância encontra-se na Seção 3.2, Classes de Entrada.

Utilizamos, quando oportuno, valores médios como resultado, mostrando em conjunto o respectivo desvio padrão. Uma outra forma de tratar tais valores seria usar o valor médio da soma dos mesmos, excluindo assim a necessidade de apresentação de desvio padrão.

Mostramos na Tabela 33 a carga em que o sistema se encontrava no início e no encerramento dos testes de todos as instâncias para cada algoritmo.

Memória RAM Livre		
Algoritmo	Início da Execução	Fim da Execução
nobound	94,27%	88,12%
basic	79,14%	77,40%
cp	85,11%	89,40%
df	87,52%	81,87%
χ	86,56%	87,90%
χ + **df**	90,20%	86,76%
mcq	78,98%	75,03%
dyn	81,67%	78,24%
mcr	72,40%	70,89%
mcs	84,45%	83,94%

Tabela 33: Porcentagem de memória RAM livre no início e na conclusão dos testes.

4.1.1 Passos de *Branching*

Antes de analisarmos esse indicador de desempenho, mostramos na Tabela 34 o tamanho da clique máxima encontrada em cada ponto de *design*. Todos os algoritmos receberam como entrada exatamente os mesmos 20 grafos gerados aleatoriamente. Em todos os testes os tamanhos das cliques máximas foi exatamente o mesmo.

ponto de *design*	**Valor médio de ω**
(500; 0,1)	5
(1.500; 0,1)	6
(30; 0,8)	13
(60; 0,8)	18

Tabela 34: Tamanho da clique máxima encontrada em cada ponto de *design*.

Apresentamos nas Tabelas 35, 36 e 37 e 38 o valor médio do indicador Passos de *Branching* de 20 testes para cada ponto de *design*. Os valores estão ordenados de forma decrescente através do respectivo indicador de desempenho.

(n, p) = (500; 0,1)		
Algoritmo	T	σ
nobound	71.903,30	2,32%
basic	29.280,90	2,15%
cp	26.318,80	2,05%
mcq	1.255,80	5,45%
dyn	1.254,60	5,34%
mcr	1.243,00	5,70%
mcs	1.154,80	3,23%
df	1.148,70	7,74%
χ	972,70	7,07%
χ + df	929,70	6,71%

Tabela 35: Número médio de Passos de *Branching* executados por cada algoritmo para o ponto de *design* (n, p) = (500, 0,1)

(n, p) = (1.750; 0,1)		
Algoritmo	T	σ
nobound	2.901.369,50	0,80%
basic	1.177.123,80	0,89%
cp	1.098.497,50	1,03%
df	114.745,60	0,77%
χ	85.577,70	1,25%
χ + df	83.750,80	1,24%
mcq	51.026,00	2,11%
dyn	50.996,80	2,00%
mcr	49.314,50	1,80%
mcs	41.488,40	1,58%

Tabela 36: Número médio de Passos de *Branching* executados por cada algoritmo para o ponto de *design* (n, p) = (1.750, 0,1).

(n, p) = (30; 0,8)		
Algoritmo	T	σ
nobound	136.256,40	37,16%
basic	2.687,90	54,24%
cp	931,20	26,36%
df	322,60	48,92%
mcq	90,80	23,12%
dyn	89,90	22,60%
χ	70,70	19,48%
mcs	66,40	43,77%
mcr	54,70	24,64%
χ + df	43,70	28,11%

Tabela 37: Número médio de Passos de *Branching* executados por cada algoritmo para o ponto de *design* (n, p) = (30, 0,8).

(n, p) = (60; 0,8)		
Algoritmo	T	σ
nobound	37.206.267,20	31,70%
basic	216.822,80	34,00%
cp	49.238,10	34,72%
df	18.561,40	39,49%
χ	832,80	42,18%
χ + df	763,30	45,64%
mcs	684,30	61,81%
mcq	648,70	40,54%
dyn	572,10	41,20%
mcr	532,10	62,21%

Tabela 38: Número médio de Passos de *Branching* executados por cada algoritmo para o ponto de *design* (n, p) = (60, 0,8).

Quanto ao desempenho dos piores colocados, em todos os Pontos de *Design*, prevalece o mesmo resultado sendo nobound, basic, cp e df, nesta ordem, os de pior desempenho.

Já quando analisamos os algoritmos de melhor desempenho, não existe uma distinção tão clara.

De forma geral, mcq e seus derivados (algoritmos originados de mcq: dyn, mcs e mcr), apresentaram um desempenho mais satisfatório, pois quando não ocuparam as melhores posições, não resultaram em valores tão distantes dos melhores colocados.

Dentre os Pontos de *Design* de baixa densidade (p=0,1), constatamos inicialmente que o algoritmo mcq e seus derivados apresentam resultados próximos, destacando-se mcs.

Para o nível baixo de n (500), χ e $\chi + \mathsf{df}$ apresentaram melhor resultado.

Para o nível alto de n (1.750), mcs apresentou melhor resultado.

Quanto tratamos dos Pontos de *Design* de alta densidade (p=0,8) os melhores casos também não pertencem a um único algoritmo.

Para o nível baixo de n (30), mcr e $\chi + \mathsf{df}$ apresentam melhor desempenho.

Para o nível alto de n (60), mcr e dyn apresentaram melhor desempenho.

Temos nas tabelas 39, 40, 41 e 42 os resultados dos testes de validação, obtidos aplicando o mesmo Modelo de *Design*, mas desta vez na servidora *Achel*.

(n, p) = (500, 0,1)		
Algoritmo	T	σ
nobound	72.471,40	2,30%
basic	29.755,60	2,38%
cp	26.398,50	2,37%
df	1.262,20	8,57%
mcq	1.238,70	2,56%
dyn	1.238,60	2,57%
mcr	1.225,20	4,73%
mcs	1.218,50	5,62%
χ	1.115,80	11,21%
χ + df	1.015,90	8,43%

Tabela 39: Testes de validação realizados na servidora *Achel* para o ponto de *design* (n, p) = (500, 0,1).

(n, p) = (1.750, 0,1)		
Algoritmo	T	σ
nobound	2.896.187,30	0,75%
basic	1.172.569,10	0,77%
cp	1.097.209,30	0,75%
df	114.564,90	0,89%
χ	85.350,60	1,14%
χ + df	83.464,90	1,49%
dyn	51.127,30	1,36%
mcq	51.127,20	1,38%
mcr	49.397,60	1,36%
mcs	41.080,70	1,18%

Tabela 40: Testes de validação realizados na servidora *Achel* Para o ponto de *design* (n, p) = (1.750, 0,1).

(n, p) = (30, 0,8)		
Algoritmo	T	σ
nobound	152.923,10	33,94%
basic	2.471,70	39,98%
cp	991,60	25,64%
df	300,40	52,02%
mcq	102,50	20,12%
dyn	101,60	20,35%
χ	77,90	25,46%
MCS	61,80	30,11%
χ + df	46,60	31,50%
mcr	44,60	20,55%

Tabela 41: Testes de validação realizados na servidora *Achel* para o ponto de *design* (n, p) = (30, 0,8).

(n, p) = (60, 0,8)		
Algoritmo	T	σ
nobound	49.599.477,30	57,23%
basic	199.332,90	36,14%
cp	52.015,60	14,45%
df	19.120,10	25,10%
χ	696,90	39,25%
χ + df	646,50	41,48%
mcs	595,00	31,03%
mcq	538,10	28,86%
mcr	491,20	30,10%
dyn	487,00	22,49%

Tabela 42: Testes de validação realizados na servidora *Achel* para o ponto de *design* (n, p) = (60, 0,8).

Ao compararmos os resultados do experimento original com os resultados dos

testes de validação percebemos que não houve diferenças significativas. Para o ponto de *design* (n, p) = (500, 0,1) apenas o algoritmo df mudou de posição, ultrapassando os algoritmos mcq e seus derivados. Entretanto, o valor do indicador de desempenho desses algoritmos é bastante próximo, não apresentando assim diferença geral nos resultados.

Para o ponto de *design* (n, p) = (1.750, 0,1), os algoritmos dyn e mcq inverteram suas posições, mas novamente, ambos apresentam um resultado bastante próximo.

O mesmo acontece com os Pontos de *Design* de alta densidade. Para (n, p) = (30, 0,8), os algoritmos $\chi + \mathsf{df}$ e mcr alternam-se na melhor posição, ao passo que para (60, 0,8), os algoritmos mcr e dyn ocupam alternadamente a melhor posição. Em todos os casos por uma diferença pequena de valor no indicador de desempenho.

4.1.2 Tempo de CPU

Analisamos agora o indicador dependente de plataforma Tempo de CPU resultante dos testes com grafos aleatórios. Todos os valores desse indicador aqui apresentados encontram-se em segundos.

Mostramos nas Tabelas 43 e 44 os resultados para os Pontos de Design (n, p) = (500, 0,1) e (1.750, 0,1), respectivamente, do mesmo modo, apresentamos nas Tabelas 45, 46, para (n, p) = (30, 0,8) e (60, 0,8), respectivamente. As tabelas apresentam o valor médio e desvio padrão de Tempo de Execução de 20 testes ordenados de forma decrescente pelo indicador de desempenho.

(n, p) = (500; 0,1)		
Algoritmo	t	σ
χ	110,8604	2,34%
$\chi + \mathsf{df}$	102,1329	3,42%
mcs	7,5728	1,85%
mcr	7,5170	1,84%
nobound	4,0021	4,01%
df	1,8921	2,47%
cp	0,8752	2,35%
basic	0,7123	2,57%
dyn	0,4217	2,25%
mcq	0,4201	2,04%

Tabela 43: Valor médio de Tempo de CPU gasto por cada algoritmo para o ponto de *design* (n, p) = (500, 0,1).

(n, p) = (1.750; 0,1)		
Algoritmo	t	σ
χ	26.071,1531	1,49%
$\chi + \mathsf{df}$	24.028,2205	1,60%
nobound	498,1932	1,56%
mcs	340,3141	1,34%
mcr	336,2582	1,36%
df	87,2861	1,51%
cp	29,7050	1,43%
basic	29,3463	1,52%
dyn	17,8397	7,16%
mcq	16,6616	1,82%

Tabela 44: Valor médio de Tempo de CPU gasto por cada algoritmo para o ponto de *design* (n, p) = (1.750, 0,1).

(n, p) = (30; 0,8)		
Algoritmo	t	σ
nobound	1,3608	36,40%
χ	0,1633	23,36%
χ + df	0,1292	33,46%
df	0,0600	33,20%
basic	0,0322	45,46%
mcs	0,0204	12,54%
mcr	0,0178	4,65%
cp	0,0126	21,57%
dyn	0,0060	17,50%
mcq	0,0059	18,34%

Tabela 45: Valor médio de Tempo de CPU gasto por cada algoritmo para o ponto de *design* (n, p) = (30, 0,8).

(n, p) = (60; 0,8)		
Algoritmo	t	σ
nobound	453,8888	31,21%
χ + df	7,7465	51,64%
χ	7,6019	51,17%
df	6,1998	32,63%
basic	3,2681	31,02%
cp	0,7869	29,40%
mcs	0,2907	32,36%
mcr	0,1995	22,71%
mcq	0,1077	34,46%
dyn	0,1041	35,44%

Tabela 46: Valor médio de Tempo de CPU gasto por cada algoritmo para o ponto de *design* (n, p) = (60, 0,8).

Percebemos que os algoritmos dyn e mcq são os mais eficientes quanto a esse

indicador, apresentando resultados bastante próximos, enquanto os algoritmos χ e $\chi + \mathsf{df}$ apresentam o pior resultado, perdendo apenas para o algoritmo nobound nos grafos de alta densidade.

Além disso, nos grafos de baixa densidade, esses algoritmos apresentaram um resultado pior que o algoritmo nobound. No ponto de *design* (n, p) = (500, 0,1), o algoritmo χ apresentou um tempo médio de CPU aproximadamente 14 vezes maior que nobound e, quando analisamos o ponto de *design* (n, p) = (1.750, 0,1), esse valor é aproximadamente 52 vezes maior que do algoritmo nobound.

Podemos constatar que a técnica de *bounding* dos algoritmos χ e $\chi + \mathsf{df}$, baseada em coloração, é demasiado custosa, fazendo com que um algoritmo que não aplica nenhuma técnica de *bounding* tenha um desempenho expressivamente melhor que ambos.

Reforçamos essa ideia partindo do conhecimento de que os algoritmos χ e $\chi + \mathsf{df}$ são especializações do algoritmo cp, que mostrou um desempenho superior a ambos em todos os Pontos de Design. A diferença entre os algoritmos χ, $\chi + \mathsf{df}$ e cp é o emprego de coloração, lembrando que os algoritmos χ, $\chi + \mathsf{df}$ calculam 4 valores de coloração e não apenas uma. O algoritmo cp apenas ordena os vértices em ordem decrescente de grau antes do início da execução.

O algoritmo basic também apresentou um desempenho inesperado. O algoritmo basic utiliza como limitante o tamanho do conjunto de vértices candidatos a serem clique máxima, ou seja, um limitante bastante simples, mas que se mostrou eficiente em grafos aleatórios de baixa densidade e manteve-se superior aos algoritmos df, χ e $\chi + \mathsf{df}$ em grafos aleatórios de alta densidade.

Nas Tabelas 43, 44, 45 e 46 temos os resultados dos testes de validação realizados na servidora *Achel*. A tabela está ordenada de forma decrescente do valor do indicador de desempenho.

Todos os algoritmos mantiveram sua posição em todos os Pontos de Design em relação aos testes realizados na servidora *Latrappe*.

(n, p) = (500, 0,1)		
Algoritmo	t	σ
χ	141,8661	2,16%
$\chi + \mathsf{df}$	128,5084	1,46%
mcs	9,0202	1,28%
mcr	8,9464	1,29%
nobound	5,2945	3,18%
df	2,2735	1,49%
cp	1,0526	1,25%
basic	0,8669	2,89%
dyn	0,5046	1,45%
mcq	0,5014	1,42%

Tabela 47: Testes de validação realizados na servidora *Achel* para o ponto de *design* (n, p) = (500, 0,1).

(n, p) = (1.750, 0,1)		
Algoritmo	t	σ
χ	33.006,1229	1,58%
$\chi + \mathsf{df}$	30.712,0981	1,65%
nobound	636,1804	1,40%
mcs	408,9228	2,03%
mcr	403,4057	1,76%
df	106,0603	1,85%
cp	36,3343	2,33%
basic	35,9976	2,21%
dyn	23,8016	12,05%
mcq	20,2824	2,24%

Tabela 48: Testes de validação realizados na servidora *Achel* para o ponto de *design* (n, p) = (1.750, 0,1).

(n, p) = (30, 0,8)		
Algoritmo	t	σ
nobound	2,1349	31,95%
χ	0,2296	23,59%
χ + df	0,1736	34,95%
df	0,0733	36,56%
basic	0,0389	35,44%
mcs	0,0260	14,36%
mcr	0,0216	8,09%
cp	0,0167	24,76%
dyn	0,0080	15,20%
mcq	0,0079	15,48%

Tabela 49: Testes de validação realizados na servidora *Achel* para todos o ponto de *design* (n, p) = (30, 0,8).

(n, p) = (60, 0,8)		
Algoritmo	t	σ
nobound	766,1501	57,39%
χ + df	7,8388	35,32%
χ	7,7749	33,76%
df	7,5938	25,02%
basic	3,7120	36,14%
cp	1,0087	14,29%
mcs	0,3243	19,34%
mcr	0,2316	11,53%
mcq	0,1124	23,73%
dyn	0,1085	20,19%

Tabela 50: Testes de validação realizados na servidora *Achel* para o ponto de *design* (n, p) = (60, 0,8).

4.1.3 Passos de *Branching* em Conjunto com Tempo de CPU

Por último, analisamos em conjunto os dois indicadores de desempenho.

Para tal análise, dividimos o Tempo de CPU pelo número de Passos de *Branching*.

Como já validamos os resultados dos indicadores de desempenho, apresentamos aqui apenas os resultados da servidora *Latrappe*. O resultado encontra-se nas Tabelas 51, 52, 53 e 54, ordenado de forma decrescente quanto ao resultado do cálculo de t/T.

(n, p) = (500; 0,1)			
Algoritmo	t	T	**t/T**
χ	110,8603	972,70	0,11397
χ + df	102,1328833	916,45	0,11144
mcs	7,5728	1.154,80	0,00656
mcr	7,5170	1.243,00	0,00605
df	1,8921	1.148,70	0,00165
dyn	0,4217	1.254,60	0,00034
mcq	0,4201	1.255,80	0,00033
nobound	4,0020	71.903,30	0,00006
cp	0,8751	26.318,80	0,00003
basic	0,7122	29.280,90	0,00002

Tabela 51: Média dos valores gastos em cada passo de *branching* de cada algoritmo para o ponto de *design* (n, p) = (500, 0,1).

(n, p) = (1.750; 0,1)			
Algoritmo	t	T	**t/T**
χ	26071,15306	85.577,70	0,30465
χ + df	24028,22046	83.750,80	0,28690
mcs	340,3140529	41.488,40	0,00820
mcr	336,258174	49.314,50	0,00682
df	87,28607761	114.745,60	0,00076
dyn	17,83974494	50.996,80	0,00035
mcq	16,66162068	51.026,00	0,00033
nobound	498,1931891	2.901.369,50	0,00017
cp	29,70498145	1.098.497,50	0,00003
basic	29,34633726	1.177.123,80	0,00002

Tabela 52: Média dos valores gastos em cada passo de *branching* de cada algoritmo para o ponto de *design* (n, p) = (1.750, 0,1).

(30; 0,8)			
Algoritmo	t	T	**t/T**
χ + df	0,129174352	43,70	0,00296
χ	0,163327265	70,70	0,00231
mcr	0,01777575	54,70	0,00032
mcs	0,020388854	66,40	0,00031
df	0,059968174	322,60	0,00019
dyn	0,005973661	89,90	0,00007
mcq	0,005922365	90,80	0,00007
cp	0,012632	931,20	0,00001
basic	0,032163656	2.687,90	0,00001
nobound	1,360842478	136.256,40	0,00001

Tabela 53: Média dos valores gastos em cada passo de *branching* de cada algoritmo para o ponto de *design* (30, 08).

(n, p) = (60; 0,8)			
Algoritmo	t	T	**t/T**
$\chi + \mathsf{df}$	7,746483111	763,30	0,01015
χ	7,601922095	832,80	0,00913
mcs	0,290727079	684,30	0,00042
mcr	0,199528062	532,10	0,00037
df	6,199825907	18.561,40	0,00033
dyn	0,104136741	572,10	0,00018
mcq	0,107664192	648,70	0,00017
cp	0,786878443	49.238,10	0,00002
basic	3,268061113	216.822,80	0,00002
nobound	453,8887889	37.206.267,20	0,00001

Tabela 54: Média dos valores gastos em cada passo de *branching* de cada algoritmo para o ponto de *design* (n, p) = (60, 08).

Podemos perceber que, em todos os Pontos de Design, os algoritmos χ e $\chi + \mathsf{df}$ possuem desempenho bem menos eficiente que os demais algoritmos. Um valor alto de t/T indica que o algoritmo gerou uma quantidade relativamente pequena Passos de *Branching* dentro do Tempo de CPU gasto. Isso ocorre quando o algoritmo gasta uma grande parcela de tempo no cálculo do *bounding*.

Espera-se que os algoritmos nobound e basic apresentem os menores valores de t/T, uma vez que nobound não emprega técnica de *bounding* e basic utiliza o tamanho do conjunto de candidatos a clique máxima como limitante, um valor que é facilmente obtido. Isso de fato ocorreu. Após esses dois algoritmos, o que obteve menor valor de t/T foi cp.

4.2 DIMACS

Temos nesta seção os resultados para a família de grafos DIMACS. Tais resultados são referentes à servidora *Latrappe* e validados na servidora *Achel* e foram obtidos através da aplicação do Modelo de *Design* definido na Seção 3.4.1.

A descrição de cada instância encontra-se na Seção 3.2, Classes de Entrada.

Mostramos na Tabela 55 a carga em que o sistema se encontrava no início e no encerramento dos testes de todos as instâncias para cada algoritmo. Devido a peculiaridades

no processo experimental dessa classe de entrada, são adicionadas duas seções à análise dos resultados: Tempo Limite e Tamanho da Clique. Dentro de cada seção expomos sua motivação. Além disso, uma seção é dedicada para a validação dos resultados, buscando melhor organização das informações.

Algoritmo	**Início da Execução**	**Fim da Execução**
nobound	79,45%	63,22%
basic	77,12%	75,29%
cp	73,90%	71,42%
df	88,10%	85,23%
χ	81,33%	76,29%
χ + **df**	81,40%	77,99%
mcq	88,02%	83,24%
dyn	85,65%	81,92%
mcr	89,14%	86,21%
mcs	88,55%	85,82%

Tabela 55: Porcentagem de memória RAM livre no início e na conclusão dos testes.

4.2.1 Tempo Limite

Antes de expormos os resultados de Passos de *Branching* e Tempo de CPU precisamos tratar da questão do tempo limite imposto na execução dos testes para os grafos DIMACS. Tal limite se faz necessário, pois como podemos ver em (ZüGE, 2011) muitas instâncias não podem ser concluídas em tempo hábil. Definimos através do experimento de descoberta já demonstrado na Figura 7, na Seção 3.4.2 que tal tempo seria de 38.400 segundos, ou aproximadamente 10,5 horas. Tal escolha está justificada na descrição da Seção 3.4.2.

Apresentamos na Tabela 56 o resultado de quais algoritmos concluíram sua execução sobre quais instâncias dentro desse tempo limite. Se a execução do algoritmo sobre a instância foi concluída, tal instância é marcada com o preenchimento na cor cinza da célula na interseção algoritmo/instância.

Tabela 56: Testes que concluíram sua execução dentro do tempo limite.

Instância	nobound	basic	χ	$\chi + \mathsf{df}$	cp	df	dyn	mcq	mcr	mcs
brock200_1										
brock200_2										
brock200_3										
brock200_4										
brock400_1										
brock400_2										
brock400_3										
brock400_4										
brock800_1										
brock800_2										
brock800_3										
brock800_4										
c-fat200-1										
c-fat200-2										
c-fat200-5										
c-fat500-1										
c-fat500-10										
c-fat500-2										
c-fat500-5										
hamming10-2										
hamming10-4										
hamming6-2										
hamming6-4										
hamming8-2										
hamming8-4										
johnson16-2-4										
johnson32-2-4										
johnson8-2-4										

Tabela 56 – continua na próxima página

Tabela 56 – continuação da página anterior										
Instância	nobound	basic	χ	$\chi + \mathsf{df}$	cp	df	dyn	mcq	mcr	mcs
johnson8-4-4										
keller4										
keller5										
keller6										
MANN_a27										
MANN_a45										
MANN_a81										
MANN_a9										
p_hat1000-1										
p_hat1000-2										
p_hat1000-3										
p_hat1500-1										
p_hat1500-2										
p_hat1500-3										
p_hat300-1										
p_hat300-2										
p_hat300-3										
p_hat500-1										
p_hat500-2										
p_hat500-3										
p_hat700-1										
p_hat700-2										
p_hat700-3										
san1000										
san200_0.7_1										
san200_0.7_2										
san200_0.9_1										
san200_0.9_2										

Tabela 56 – continua na próxima página

Tabela 56 – continuação da página anterior										
Instância	nobound	basic	χ	χ + df	cp	df	dyn	mcq	mcr	mcs
san200_0.9_3										
san400_0.5_1										
san400_0.7_1										
san400_0.7_2										
san400_0.7_3										
san400_0.9_1										
sanr200_0.7										
sanr200_0.9										
sanr400_0.5										
sanr400_0.7										

Tabela 56: Testes que concluíram sua execução dentro do tempo limite.

Notamos inicialmente que nenhum algoritmo conseguiu concluir a execução sobre todas as instâncias dentro do tempo limite.

Notamos também que apenas as famílias c-fat e san tiveram, em um ou mais algoritmos, a execução de todas as suas instâncias concluídas. No caso de c-fat, os algoritmos χ, χ + df , df, dyn, mcr e mcs concluíram a execução de todas as instâncias. Já para as instâncias da família san, apenas os algoritmos dyn e mcr concluíram todas. Instâncias anotadas em **negrito** não foram concluídas por nenhum algoritmo. A tabela com os tempos de CPU de cada instância para cada algoritmo encontra-se na Tabela 66, na Seção 4.2.4.

Encontramos na tabela 57 o número de instâncias sobre as quais cada algoritmo completou sua execução, ordenada de forma decrescente de número de instâncias concluídas.

Instâncias Concluídas	
Algoritmo	Quantidade
dyn	45 (68,18%)
mcr	44 (67,69%)
mcs	40 (60,60%)
χ	28 (42,42%)
χ + df	28 (42,42%)
df	26 (39,39%)
basic	26 (39,39%)
mcq	26 (39,39%)
cp	24 (36,36%)
nobound	15 (22,72%)

Tabela 57: Instâncias concluídas por algoritmo. Em parênteses a porcentagem do total de instâncias que cada algoritmo concluiu.

Podemos perceber que houve mais casos de não conclusão da execução dentro do tempo limite. 302 testes foram concluídos de 660. Apenas 3 algoritmos, dyn, mcr e mcs, concluíram mais da metade das instâncias. Nota-se também, através da Tabela 57 que o algoritmo que concluiu a execução de mais instâncias é o algoritmo dyn, em segundo lugar próximo, mcr. Sendo o pior neste quesito, sem grandes surpresas, por não empregar nenhuma técnica de otimização de busca, o algoritmo nobound.

Uma surpresa aqui foi constatar que os algoritmos cp, mcq e df tiveram um desempenho igual ou inferior ao algoritmo basic, que emprega uma técnica de *bounding* bastante trivial, como explicado na Seção 3.3.3.

4.2.2 Tamanho da Clique Máxima Encontrada

Uma vez que vários algoritmos não conseguiram executar completamente todas as instâncias dentro do tempo limite, mostramos agora o resultado a respeito do tamanho das cliques encontradas, sejam elas as efetivas Cliques Máximas ou a maior clique encontrada até o momento em que o algoritmo teve sua execução encerrada por atingir o tempo limite.

Na Tabela 58 temos o número de instâncias em que cada algoritmo encontrou uma clique máxima dentro do tempo limite e o número de casos em que o algoritmo encontrou

uma clique máxima independente de ter encerrado a execução ou não.

Para obtermos esses resultados, nos casos em que o algoritmo não completou sua execução, comparamos o tamanho da maior clique encontrada com o tamanho da clique máxima da instância, que é descrito no cabeçalho da mesma e encontra-se na Seção 3.2.2.

Algoritmo	**Quantidade de Instâncias em que a clique máxima foi encontrada**	**Quantidade de Instâncias em que uma clique máxima foi encontrada**
dyn	45	52
mcr	44	50
mcs	40	43
χ	28	32
χ + df	28	32
basic	26	32
mcq	26	32
df	26	30
cp	24	31
nobound	15	26

Tabela 58: Número de instâncias em que cada algoritmo encontrou a clique máxima. Na terceira coluna, quantos casos o algoritmo encontrou a clique máxima, tendo a execução do algoritmo sido encerrada ou não.

Um fato interessante é que mesmo com os melhoramentos de *software* e *hardware*, nenhum algoritmo, no âmbito do nosso experimento, conseguiu encontrar uma clique maior do que a maior encontrada quando os resultados do desafio DIMACS foram publicados em 1996 (onde a execução também foi encerrada usando tempo limite) (JOHNSON; TRICK, 1996a). Suspeitamos que tal ocorrido se deva ao fato de usarmos a linguagem *Python* para a implementação dos algoritmos.

Fica evidente, através da Tabela 58, que o algoritmo dyn foi o melhor em encontrar a clique máxima nos grafos tendo a encontrado em 45 casos (68,18%). Lembramos que a família de grafos DIMACS possui 66 instâncias.

O algoritmo cp apresentou um desempenho pior que basic, que emprega uma técnica trivial de *bounding*. Os algoritmos df e mcq apresentaram desempenho semelhante

ao basic.

4.2.3 Passos de *Branching*

Temos na Tabela 59 o resultado completo de Passos de *Branching* de cada algoritmo.

Tabela 59: Número de Passos de *Branching* executado por cada algoritmo.

Instancia	nobound	basic	χ	$\chi + \mathsf{df}$	cp	df	dyn	mcq	mcr	mcs
brock200_1	792.252.455	876.403.943	491.164	455.516	258.012.511	45.487.955	457.353	876.403.943	917.233	810.885
brock200_2	12.585.199	700.185	10.069	9.921	455.413	55.593	7.247	700.185	7.767	9.475
brock200_3	365.514.611	8.610.083	74.455	74.663	3.589.195	696.877	25.375	8.610.083	32.475	46.007
brock200_4	653.665.494	28.636.755	132.095	132.071	15.766.703	2.263.155	94.459	28.636.755	144.115	106.941
brock400_1	1.372.832.800	1.579.559.329	143.552	128.869	1.411.505.522	50.871.264	62.013.115	1.570.139.627	64.524.246	1.502.283.303
brock400_2	1.373.940.291	1.523.953.438	139.388	139.544	1.403.901.415	45.171.363	46.140.926	1.488.867.444	68.513.460	1.462.157.838
brock400_3	1.557.768.190	1.544.994.800	133.516	139.726	1.340.757.472	49.531.829	1.286.917.370	1.532.761.304	60.811.335	1.509.520.441
brock400_4	1.368.319.292	1.490.862.387	133.002	137.355	1.406.009.465	50.322.425	45.564.505	1.405.415.129	54.905.275	1.402.800.673
brock800_1	1.438.947.939	1.702.002.635	97.001	103.395	1.552.537.663	54.518.463	58.806.772	1.618.560.751	79.209.325	1.640.314.702
brock800_2	1.409.054.091	1.649.041.965	81.990	96.856	1.583.650.878	49.002.281	59.412.938	1.624.630.065	75.197.799	1.599.261.656
brock800_3	1.498.118.610	1.620.665.373	91.216	108.616	1.532.947.234	49.932.338	62.233.087	1.551.138.716	74.193.214	1.558.804.796
brock800_4	1.494.195.878	1.620.236.354	85.968	114.930	1.489.801.507	51.983.014	50.966.767	1.572.500.046	76.024.151	1.560.819.523
c-fat200-1	162.561	811	59	39	515	39	437	811	377	377
c-fat200-2	192.856.065	1.917	49	5	2.405	85	487	1.917	353	353
c-fat200-5	1.364.010.710	140.789	239	127	643.874.170	127	621	140.789	285	285
c-fat500-1	1.125.889	2.151	29	5	1.187	205	1.045	2.151	973	973
c-fat500-10	355.706.421	286.282.778	253	5	346.751.948	321	1.493	282.162.232	749	749
c-fat500-2	780.134.773	8.365	53	5	4.227	285	1.093	8.365	949	949
c-fat500-5	723.569.720	571.779	129	5	29.861.813	303	1.245	571.779	873	873
hamming10-2	188.347.270	118.146.502	1.025	543	111.290.446	1.670.366	2.523	117.563.286	1.025	1.025
hamming10-4	1.862.287.451	1.845.277.361	38.020	39.720	1.672.257.761	65.666.598	21.561.034	1.779.980.222	36.526.945	36.480.716
hamming6-2	1.525.409.292	508.981	65	33	203.817	23.749	127	508.981	65	65
hamming6-4	3.937	2.091	199	179	1.657	313	221	2.091	165	159
hamming8-2	641.221.038	397.723.307	257	129	386.744.131	4.571.958	511	393.187.995	257	257
hamming8-4	1.274.021.383	114.637.761	34.299	28.735	33.923.933	4.186.763	41.023	114.637.761	82.985	53.795
johnson16-2-4	92.413.471	36.684.707	1.903.631	1.937.023	23.284.309	6.314.291	1.302.089	36.684.707	646.073	582.951
johnson32-2-4	3.429.861.301	3.118.368.201	653.465	777.837	3.078.378.672	241.752.422	494.069.783	3.016.086.146	483.182.549	350.365.121

Tabela 59 – continua na próxima página

Tabela 59 – continuação da página anterior										
Instancia	nobound	basic	χ	$\chi + \mathsf{df}$	cp	df	dyn	mcq	mcr	mcs
johnson8-2-4	1.527	717	81	79	569	133	95	717	73	55
johnson8-4-4	7.705.151	272.001	141	131	87.663	23.915	447	272.001	289	345
keller4	127.267.065	19.055.823	64.357	60.083	8.518.051	1.542.373	17.469	19.055.823	22.311	41.603
keller5	1.435.953.623	1.652.515.144	46.646	40.565	1.397.236.905	48.418.003	19.746.343	1.571.521.075	23.360.773	21.247.054
keller6	1.216.523.202	1.423.791.481	62.805	78.042	1.412.480.092	53.540.849	8.136.128	1.298.588.976	3.081.827	4.071.405
MANN_a27	1.842.521.030	1.751.818.471	116.633	122.709	356.583.240	141.534.916	76.509	1.732.935.128	76.041	57.228.463
MANN_a45	1.416.419.765	1.436.860.868	143.450	244.660	214.877.312	103.986.423	168.016	1.435.281.442	199.489	34.339.196
MANN_a81	1.120.079.341	1.130.617.479	1.097	1.082	88.513.354	77.062.739	18.072	1.079.417.006	20.879	19.734.122
MANN_a9	320.505.441	4.521.439	2.061	1.667	1.216.045	590.077	191	4.521.439	143	867
p_hat1000-1	195.934.761	25.226.263	105.912	133.502	16.178.765	1.702.829	340.301	25.226.263	404.185	362.631
p_hat1000-2	1.419.443.679	1.246.580.829	49.553	37.566	908.264.431	26.905.625	12.508.436	1.196.720.590	15.765.624	15.887.015
p_hat1000-3	1.423.870.182	1.103.489.502	23.348	22.326	872.495.041	25.492.655	11.526.969	1.042.770.416	7.767.578	9.636.793
p_hat1500-1	263.689.861	248.019.237	90.212	95.472	139.333.997	15.580.561	2.275.349	247.019.237	2.521.053	2.720.917
p_hat1500-2	1.224.605.071	1.108.990.759	72.460	22.400	955.858.496	24.579.141	9.742.637	1.044.341.143	6.422.538	6.458.550
p_hat1500-3	1.217.544.104	1.033.953.930	14.804	14.316	841.237.399	24.327.751	8.084.665	935.694.895	3.208.352	3.018.965
p_hat300-1	734.645	131.347	4.193	4.117	95.753	9.463	4.325	131.347	4.065	2.793
p_hat300-2	1.270.967.345	222.733.859	73.029	73.875	9.781.107	5.634.539	15.361	222.733.859	9.979	86.639
p_hat300-3	1.400.844.719	1.512.408.727	216.429	214.376	999.105.410	29.784.374	1.251.627	1.166.288.803	3.564.775	19.469.638
p_hat500-1	8.507.185	1.258.995	31.095	31.977	778.165	89.687	21.687	1.258.995	21.573	20.341
p_hat500-2	1.771.576.837	1.200.727.352	118.798	124.476	968.885.358	28.459.444	388.111	1.141.248.373	856.281	7.270.503
p_hat500-3	1.173.860.061	1.116.243.883	85.255	84.451	954.293.027	31.466.837	11.502.144	1.083.664.046	17.523.386	16.842.610
p_hat700-1	39.383.251	4.002.661	60.847	57.775	3.046.223	226.047	56.073	4.002.661	68.513	39.325
p_hat700-2	1.458.963.147	1.124.201.031	63.462	66.353	846.556.017	25.006.718	2.183.587	1.093.780.110	4.843.391	13.873.379
p_hat700-3	1.367.119.982	1.038.809.377	43.742	45.037	929.591.245	36.952.665	8.813.303	1.012.533.803	8.282.230	7.879.094
san1000	1.775.991.073	1.786.797.847	28.694	28.295	1.726.549.262	48.139.216	251.937	1.704.596.479	454.323	125.941
san200_.7_1	3.073.877.821	2.995.389.744	81.829	78.415	1.211.889.697	475.486.269	1.681	2.982.284.485	2.641	3.847
san200_.7_2	3.349.553.027	3.237.857.891	47.503	38.365	2.278.672.573	360.879.277	3.459	3.203.321.315	3.207	3.209
san200_.9_1	1.381.626.560	986.615.405	9.793	8.609	1.262.741.727	31.138.517	57.917	991.229.807	335.593	293.955

Tabela 59 – continua na próxima página

Tabela 59 – continuação da página anterior										
Instancia	nobound	basic	χ	$\chi + \mathsf{df}$	cp	df	dyn	mcq	mcr	mcs
san200_.9_2	1.393.156.189	1.164.293.584	368.894	359.013	1.305.100.820	31.339.861	104.463	1.152.027.586	799.783	1.405.665
san200_.9_3	1.355.060.699	1.422.714.058	394.340	374.919	1.230.043.269	52.015.703	694.687	1.415.136.730	65.169	2.861.673
san400_.5_1	2.689.328.254	2.652.935.965	9.827	9.721	2.149.938.440	182.535.810	5.845	2.627.347.300	4.013	1.419
san400_.7_1	4.799.869.803	4.307.336.931	135.343	138.658	3.188.430.388	601.779.262	85.917	4.251.407.753	212.671	95.731
san400_.7_2	4.814.905.618	4.014.250.326	137.632	137.686	399.780.482	547.888.711	19.579	4.007.433.356	54.853	1.358.825
san400_.7_3	2.796.941.779	2.509.813.178	152.445	155.739	2.279.564.501	156.782.303	1.111.485	2.443.318.218	830.813	4.924.083
san400_.9_1	1.829.460.381	1.280.302.308	113.517	107.745	1.138.385.107	237.267.123	752.761	1.275.217.628	227.235	10.169.641
sanr200_.7	1.388.680.436	127.231.185	392.881	406.553	55.196.725	9.606.435	210.471	127.231.185	360.205	338.665
sanr200_.9	1.361.280.258	1.250.601.843	371.764	356.436	1.035.046.911	34.293.738	12.449.743	1.111.389.531	27.375.042	20.157.930

Tabela 59: Número de Passos de *Branching* executado por cada algoritmo.

É esperado que o comportamento dos algoritmos nobound e basic seja o pior para todas as situações.

Na Tabela 60 encontramos a quantidade de casos em que cada algoritmo apresentou o melhor desempenho em relação aos demais.

Algoritmo	Pior Desempenho	Porcentagem Total
nobound	51	77,27%
basic	15	22,73%
mcq	0	0,00%
cp	0	0,00%
χ	0	0,00%
χ + df	0	0,00%
df	0	0,00%
dyn	0	0,00%
mcr	0	0,00%
mcs	0	0,00%

Algoritmo	Melhor Desempenho	Porcentagem Total
χ + df	26	39,39%
χ	19	28,79%
dyn	9	13,64%
mcs	7	10,61%
mcr	5	7,58%
df	2	3,03%
basic	0	0,00%
cp	0	0,00%
mcq	0	0,00%
nobound	0	0,00%

Tabela 60: Número de instâncias em que cada algoritmo apresentou menor e maior número de Passos de *Branching*.

Percebemos através dessa tabela que os resultados, nesse aspecto, não foram os esperados, isso porque os algoritmos mcq e cp não apresentaram o melhor resultado para nenhuma instância, comparando-se assim ao basic e nobound.

Já quando buscamos o algoritmo que apresentou o melhor comportamento, percebemos que a concentração de melhores casos está nos algoritmos χ + df e χ, apresentando o melhor desempenho em 26 e 19 casos, respectivamente.

O que ambos algoritmos possuem em comum é o uso de coloração através do cálculo de 4 colorações diferentes para determinar o valor usado na técnica de *bounding*, como explicado nas seções 3.3.6 e 3.3.7, o que se mostrou bastante eficiente no que diz respeito à redução do número dos Passos de *Branching*.

O único caso em que um algoritmo apresentou o melhor desempenho para todas as instâncias de uma família foi o algoritmo χ + df sobre a família *c*.

Em 51 (77%) instâncias, nobound apresentou o pior desempenho, sendo basic

responsável pelas demais 15 (23%).

Nas 15 instâncias, apresentadas na Tabela 61 o algoritmo basic apresentou um desempenho pior que nobound. Todavia, como nesses a execução de nenhum dos algoritmos foi completada, não é possível tirarmos conclusões sobre tal fato.

Podemos constatar entretanto, através da Tabela 62, ordenada de forma decrescente quanto aos Passos de *Branching* do algoritmo nobound, que nos 16 casos em que ambos, basic e nobound, completaram sua execução, nobound gerou um maior número de Passos de *Branching*, como esperado.

Instancia	**nobound**	**basic**	**Diferença**
brock200_1	792.252.455	876.403.943	9,60%
brock400_1	1.372.832.800	1.579.559.329	13,09%
brock400_2	1.373.940.291	1.523.953.438	9,84%
brock400_4	1.368.319.292	1.490.862.387	8,22%
brock800_1	1.438.947.939	1.702.002.635	15,46%
brock800_2	1.409.054.091	1.649.041.965	14,55%
brock800_3	1.498.118.610	1.620.665.373	7,56%
brock800_4	1.494.195.878	1.620.236.354	7,78%
keller5	1.435.953.623	1.652.515.144	13,10%
keller6	1.216.523.202	1.423.791.481	14,56%
MANN_a45	1.416.419.765	1.436.860.868	1,42%
MANN_a81	1.120.079.341	1.130.617.479	0,93%
p_hat300-3	1.400.844.719	1.512.408.727	7,38%
san1000	1.775.991.073	1.786.797.847	0,60%
san200_.9_3	1.355.060.699	1.422.714.058	4,76%

Tabela 61: A coluna "Diferença" mostra a porcentagem de passos a mais que o basic gerou em relação aos executado pelo nobound.

Como nobound é utilizado como limitante superior, uma vez que não utiliza nenhuma técnica de *bounding*, analisamos agora o comportamento de cada algoritmo perante o nobound, ou seja, consideramos o nobound o pior caso e analisamos o quão distante do pior caso se encontra o pior desempenho de cada algoritmo.

Após o algoritmo nobound, o algoritmo basic apresenta o pior desempenho em

Instâncias	nobound	basic
johnson8-2-4	1.527	717
hamming6-4	3.937	2.091
c-fat200-1	162.561	811
p_hat300-1	734.645	131.347
c-fat500-1	1.125.889	2.151
johnson8-4-4	7.705.151	272.001
p_hat500-1	8.507.185	1.258.995
brock200_2	12.585.199	700.185
p_hat700-1	39.383.251	4.002.661
johnson16-2-4	92.413.471	36.684.707
keller4	127.267.065	19.055.823
c-fat200-2	192.856.065	1.917
p_hat1000-1	195.934.761	25.226.263
MANN_a9	320.505.441	4.521.439
brock200_3	365.514.611	8.610.083

Tabela 62: Passos de *Branching* em instâncias completadas por ambos os algoritmos nobound e basic.

60 (91%) das instâncias. Vemos na Tabela 63 a porcentagem dos Passos de *Branching* que cada algoritmo usou em relação ao algoritmo nobound. (Porcentagens acima de 100% indicam que tal algoritmo gerou ainda mais passos que nobound.)

Percebemos através dessa tabela que em 26 casos outros algoritmos se mostraram piores que o nobound (desconsiderando o Basic) em Passos de *Branching*. Em todas as instâncias em que isso se deu, o algoritmo nobound não conseguiu concluir sua execução. Tais casos são anotados em **negrito** na Tabela 63.

Notamos também que a redução de Passos de *Branching* foi expressiva na grande maioria dos casos. De 594 casos, 238 deles estão abaixo de 1%, ou seja, o algoritmo usou menos de 1% do número de passos em relação ao nobound para executar a mesma tarefa. Esse número sobe para 281 quando buscamos valores abaixo de 5% e para 397 em que a porcentagem está abaixo de 50%.

Tabela 63: Porcentagem de Passos de *Branching* que cada algoritmo gerou em relação ao algoritmo nobound.

Instância	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
brock200_1	**110,62180%**	0,06200%	0,05750%	32,56696%	5,74160%	0,05773%	**110,62180%**	0,11578%	0,10235%
brock200_2	5,56356%	0,08001%	0,07883%	3,61864%	0,44173%	0,05758%	5,56356%	0,06172%	0,07529%
brock200_3	2,35561%	0,02037%	0,02043%	0,98196%	0,19066%	0,00694%	2,35561%	0,00888%	0,01259%
brock200_4	4,38095%	0,02021%	0,02020%	2,41204%	0,34623%	0,01445%	4,38095%	0,02205%	0,01636%
brock400_1	**115,05839%**	0,01046%	0,00939%	**102,81700%**	3,70557%	4,51716%	**114,37224%**	4,70008%	**109,42944%**
brock400_2	**110,91846%**	0,01015%	0,01016%	**102,18067%**	3,28772%	3,35829%	**108,36479%**	4,98664%	**106,42077%**
brock400_3	99,18002%	0,00857%	0,00897%	86,06913%	3,17967%	82,61289%	98,39470%	3,90375%	96,90276%
brock400_4	**108,95574%**	0,00972%	0,01004%	**102,75449%**	3,67768%	3,32996%	**102,71105%**	4,01261%	**102,51998%**
brock800_1	**118,28104%**	0,00674%	0,00719%	**107,89394%**	3,78877%	4,08679%	112,48223%	5,50467%	**113,99403%**
brock800_2	**117,03184%**	0,00582%	0,00687%	**112,39106%**	3,47767%	4,21651%	115,29934%	5,33676%	**113,49895%**
brock800_3	**108,18004%**	0,00609%	0,00725%	**102,32482%**	3,33300%	4,15408%	103,53911%	4,95243%	**104,05083%**
brock800_4	**108,43534%**	0,00575%	0,00769%	99,70590%	3,47900%	3,41098%	105,24056%	5,08796%	**104,45883%**
c-fat200-1	0,49889%	0,03629%	0,02399%	0,31680%	0,02399%	0,26882%	0,49889%	0,23191%	0,23191%
c-fat200-2	0,00099%	0,00003%	0,00000%	0,00125%	0,00004%	0,00025%	0,00099%	0,00018%	0,00018%
c-fat200-5	0,01032%	0,00002%	0,00001%	47,20448%	0,00001%	0,00005%	0,01032%	0,00002%	0,00002%
c-fat500-1	0,19105%	0,00258%	0,00044%	0,10543%	0,01821%	0,09282%	0,19105%	0,08642%	0,08642%
c-fat500-10	80,48288%	0,00007%	0,00000%	97,48262%	0,00009%	0,00042%	79,32447%	0,00021%	0,00021%
c-fat500-2	0,00107%	0,00001%	0,00000%	0,00054%	0,00004%	0,00014%	0,00107%	0,00012%	0,00012%

Tabela 63 – continua na próxima página

Tabela 63 – continuação da página anterior									
Instância	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
c-fat500-5	0,07902%	0,00002%	0,00000%	4,12701%	0,00004%	0,00017%	0,07902%	0,00012%	0,00012%
hamming10-2	62,72801%	0,00054%	0,00029%	59,08790%	0,88685%	0,00134%	62,41836%	0,00054%	0,00054%
hamming10-4	99,08660%	0,00204%	0,00213%	89,79590%	3,52613%	1,15777%	95,58032%	1,96140%	1,95892%
hamming6-2	0,03337%	0,00000%	0,00000%	0,01336%	0,00156%	0,00001%	0,03337%	0,00000%	0,00000%
hamming6-4	53,11151%	5,05461%	4,54661%	42,08788%	7,95022%	5,61341%	53,11151%	4,19101%	4,03861%
hamming8-2	62,02593%	0,00004%	0,00002%	60,31370%	0,71301%	0,00008%	61,31864%	0,00004%	0,00004%
hamming8-4	8,99810%	0,00269%	0,00226%	2,66274%	0,32863%	0,00322%	8,99810%	0,00651%	0,00422%
johnson16-2-4	39,69628%	2,05991%	2,09604%	25,19580%	6,83265%	1,40898%	39,69628%	0,69911%	0,63081%
johnson32-2-4	90,91820%	0,01905%	0,02268%	89,75228%	7,04846%	14,40495%	87,93610%	14,08752%	10,21514%
johnson8-2-4	46,95481%	5,30452%	5,17354%	37,26261%	8,70989%	6,22135%	46,95481%	4,78062%	3,60183%
johnson8-4-4	3,53012%	0,00183%	0,00170%	1,13772%	0,31038%	0,00580%	3,53012%	0,00375%	0,00448%
keller4	14,97310%	0,05057%	0,04721%	6,69305%	1,21192%	0,01373%	14,97310%	0,01753%	0,03269%
keller5	**115,08137%**	0,00325%	0,00282%	97,30376%	3,37184%	1,37514%	**109,44094%**	1,62685%	1,47965%
keller6	**117,03776%**	0,00516%	0,00642%	**116,10795%**	4,40114%	0,66880%	**106,74593%**	0,25333%	0,33468%
MANN_a27	95,07726%	0,00633%	0,00666%	19,35301%	7,68159%	0,00415%	94,05239%	0,00413%	3,10599%
MANN_a45	**101,44315%**	0,01013%	0,01727%	15,17045%	7,34150%	0,01186%	**101,33164%**	0,01408%	2,42437%
MANN_a81	**100,94084%**	0,00010%	0,00010%	7,90242%	6,88011%	0,00161%	96,36969%	0,00186%	1,76185%
MANN_a9	1,41072%	0,00064%	0,00052%	0,37941%	0,18411%	0,00006%	1,41072%	0,00004%	0,00027%
Tabela 63 – continua na próxima página									

Tabela 63 – continuação da página anterior									
Instância	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
p_hat1000-1	12,87483%	0,05405%	0,06814%	8,25722%	0,86908%	0,17368%	12,87483%	0,20629%	0,18508%
p_hat1000-2	87,82179%	0,00349%	0,00265%	63,98735%	1,89550%	0,88122%	84,30913%	1,11069%	1,11924%
p_hat1000-3	77,49931%	0,00164%	0,00157%	61,27631%	1,79038%	0,80955%	73,23494%	0,54553%	0,67680%
p_hat1500-1	94,05718%	0,03421%	0,03621%	52,84010%	5,90867%	0,86289%	93,67794%	0,95607%	1,03186%
p_hat1500-2	90,55905%	0,00592%	0,00183%	78,05443%	2,00711%	0,79557%	85,27983%	0,52446%	0,52740%
p_hat1500-3	84,92127%	0,00122%	0,00118%	69,09297%	1,99810%	0,66401%	76,85101%	0,26351%	0,24796%
p_hat300-1	17,87898%	0,57075%	0,56041%	13,03391%	1,28811%	0,58872%	17,87898%	0,55333%	0,38018%
p_hat300-2	17,52475%	0,00575%	0,00581%	0,76958%	0,44333%	0,00121%	17,52475%	0,00079%	0,00682%
p_hat300-3	**107,96405%**	0,01545%	0,01530%	71,32164%	2,12617%	0,08935%	83,25611%	0,25447%	1,38985%
p_hat500-1	14,79920%	0,36551%	0,37588%	9,14715%	1,05425%	0,25493%	14,79920%	0,25359%	0,23910%
p_hat500-2	67,77732%	0,00671%	0,00703%	54,69056%	1,60645%	0,02191%	64,41992%	0,04833%	0,41040%
p_hat500-3	95,09173%	0,00726%	0,00719%	81,29530%	2,68063%	0,97986%	92,31629%	1,49280%	1,43481%
p_hat700-1	10,16336%	0,15450%	0,14670%	7,73482%	0,57397%	0,14238%	10,16336%	0,17396%	0,09985%
p_hat700-2	77,05479%	0,00435%	0,00455%	58,02450%	1,71401%	0,14967%	74,96969%	0,33197%	0,95091%
p_hat700-3	75,98524%	0,00320%	0,00329%	67,99632%	2,70296%	0,64466%	74,06327%	0,60582%	0,57633%
san1000	**100,60849%**	0,00162%	0,00159%	97,21610%	2,71056%	0,01419%	95,98001%	0,02558%	0,00709%
san200_.7_1	97,44661%	0,00266%	0,00255%	39,42543%	15,46861%	0,00005%	97,02027%	0,00009%	0,00013%
san200_.7_2	96,66537%	0,00142%	0,00115%	68,02915%	10,77395%	0,00010%	95,63429%	0,00010%	0,00010%

Tabela 63 – continua na próxima página

Tabela 63 – continuação da página anterior									
Instância	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
san200_.9_1	71,40970%	0,00071%	0,00062%	91,39530%	2,25376%	0,00419%	71,74368%	0,02429%	0,02128%
san200_.9_2	83,57237%	0,02648%	0,02577%	93,67943%	2,24956%	0,00750%	82,69192%	0,05741%	0,10090%
san200_.9_3	**104,99264%**	0,02910%	0,02767%	90,77403%	3,83863%	0,05127%	**104,43346%**	0,00481%	0,21118%
san400_.5_1	98,64679%	0,00037%	0,00036%	79,94333%	6,78741%	0,00022%	97,69530%	0,00015%	0,00005%
san400_.7_1	89,73862%	0,00282%	0,00289%	66,42743%	12,53741%	0,00179%	88,57340%	0,00443%	0,00199%
san400_.7_2	83,37132%	0,00286%	0,00286%	8,30298%	11,37901%	0,00041%	83,22974%	0,00114%	0,02822%
san400_.7_3	89,73419%	0,00545%	0,00557%	81,50204%	5,60549%	0,03974%	87,35678%	0,02970%	0,17605%
san400_.9_1	69,98251%	0,00620%	0,00589%	62,22519%	12,96924%	0,04115%	69,70458%	0,01242%	0,55588%
sanr200_.7	9,16202%	0,02829%	0,02928%	3,97476%	0,69177%	0,01516%	9,16202%	0,02594%	0,02439%
sanr200_.9	91,86953%	0,02731%	0,02618%	76,03481%	2,51923%	0,91456%	81,64296%	2,01098%	1,48081%
sanr400_.5	6,68312%	0,02868%	0,03221%	4,30827%	0,51919%	0,06014%	6,68312%	0,07242%	0,06014%
sanr400_.7	90,98229%	0,00822%	0,00839%	79,68386%	3,14181%	2,87443%	89,55377%	3,80561%	2,76842%

Tabela 63: Porcentagem de Passos de *Branching* que cada algoritmo gerou em relação ao algoritmo nobound.

Na Tabela 64 encontramos o número médio de passos apresentados na Tabela 63 separados por família. Os melhores resultados são anotados em **negrito** e os piores em *itálico*.

	basic	χ	$\chi + \mathsf{df}$	cp	df	MCD	mcq	mcr	mcs
brock	*84,080%*	**0,020%**	**0,020%**	71,310%	2,887%	9,152%	81,944%	3,224%	70,957%
c-fat	11,609%	0,006%	**0,003%**	*21,320%*	0,006%	0,052%	11,444%	0,046%	0,046%
hamming	*47,664%*	0,843%	**0,759%**	42,327%	2,234%	1,129%	46,910%	1,027%	1,000%
johnson	*45,275%*	1,846%	**1,823%**	38,337%	5,725%	5,510%	44,529%	4,893%	3,613%
keller	*82,364%*	0,020%	**0,019%**	73,368%	2,995%	0,686%	77,053%	0,633%	0,616%
MANN	*74,718%*	**0,004%**	0,006%	10,701%	5,522%	**0,004%**	73,291%	0,005%	1,823%
p_hat	*62,132%*	**0,082%**	0,083%	46,501%	1,911%	0,471%	58,375%	0,488%	0,618%
san	*89,652%*	**0,007%**	0,007%	70,811%	7,870%	0,015%	88,551%	0,015%	0,100%
sanr	*49,674%*	**0,023%**	0,024%	41,000%	1,718%	0,966%	46,760%	1,479%	1,083%

Tabela 64: Média de Passos de *Branching* em relação ao algoritmo nobound separadas por família.

Fica claro que os algoritmos χ e $\chi + \mathsf{df}$ são os quando tratamos do indicador de desempenho Passos de *Branching*. O desempenho dos mesmos é bastante parecido, sendo necessário na maioria dos casos a terceira casa decimal para decidir qual apresenta menor valor.

Esse resultado é corroborado pela Tabela 65, onde podemos ver o número de casos em que cada algoritmo representou a maior e menor economia de passos em relação ao algoritmo nobound. Ambas as colunas estão ordenadas de forma decrescente. Podemos ver que os algoritmos basic e mcq concentram os piores casos enquanto os algoritmos χ e $\chi + \mathsf{df}$ concentram os melhores. A soma dos valores de ambas as colunas excede o número de instâncias devido a empates de resultados.

Algoritmo	Pior Desempenho
basic	60
mcq	22
cp	6
χ	0
χ + df	0
df	0
dyn	0
mcr	0
mcs	0

Algoritmo	Melhor Desempenho
χ + df	26
χ	19
dyn	9
mcs	7
mcr	5
df	2
basic	0
cp	0
mcq	0

Tabela 65: Número de instâncias em que cada algoritmo representou a maior e pior economia de Passos de *Branching* em relação ao algoritmo nobound.

4.2.4 Tempo de CPU

Analisamos agora o Tempo de CPU de cada algoritmo sobre cada instância. Na Tabela 66 temos o resultado completo dos testes.

Tabela 66: Tempo de CPU, em segundos, de cada algoritmo para cada instância.

Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
brock200_1	38400,00	22101,96	38400,08	38400,09	7016,14	38400,00	312,24	22520,15	492,99	610,93
brock200_2	362,10	14,49	544,24	489,45	9,69	32,38	3,20	14,73	5,26	6,43
brock200_3	21320,26	178,95	4653,89	4674,69	86,48	415,40	12,53	189,24	17,04	25,75
brock200_4	38400,00	663,04	9172,66	9504,07	374,27	1550,26	41,32	678,03	59,81	59,95
brock400_1	38400,00	38400,00	38400,47	38400,25	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock400_2	38400,00	38400,00	38400,09	38400,71	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock400_3	38400,00	38400,00	38400,16	38400,07	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock400_4	38400,00	38400,00	38400,38	38400,51	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock800_1	38400,00	38400,00	38400,56	38400,10	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock800_2	38400,00	38400,00	38400,75	38400,44	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock800_3	38400,00	38400,00	38400,47	38400,94	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
brock800_4	38400,00	38400,00	38400,51	38401,64	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
c-fat200-1	4,62	0,08	3,14	3,11	0,10	0,14	0,08	0,08	0,57	0,47
c-fat200-2	5431,09	0,18	0,81	0,37	0,15	0,24	0,15	0,18	1,11	1,00
c-fat200-5	38400,00	9,64	23,86	17,56	38400,00	1,04	0,85	10,24	2,96	2,81
c-fat500-1	54,29	0,56	1,22	1,66	0,60	1,07	0,41	0,45	3,41	3,40
c-fat500-10	38400,00	38400,00	57,10	9,43	38400,00	11,54	8,54	38400,00	37,86	37,95
c-fat500-2	38400,00	0,64	1,60	1,97	0,79	1,29	0,54	0,58	6,41	6,41

Tabela 66 – continua na próxima página

Tabela 66 – continuação da página anterior										
Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
c-fat500-5	38400,00	32,93	11,05	2,34	1949,93	3,49	2,03	44,75	16,83	16,80
hamming10-2	38400,00	38400,00	3646,54	2561,53	38400,00	38400,24	527,18	38400,00	74,00	78,43
hamming10-4	38400,00	38400,00	38402,02	38402,52	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
hamming6-2	38400,00	18,16	0,91	0,56	7,69	37,12	0,07	18,57	0,07	0,83
hamming6-4	0,06	0,03	1,42	1,12	0,03	0,08	0,02	0,03	0,03	0,02
hamming8-2	38400,00	38400,00	52,35	36,30	38400,00	38400,10	4,75	38400,00	1,29	1,30
hamming8-4	38400,00	2640,84	4036,40	5048,92	810,80	3584,62	29,56	2708,65	55,45	44,64
johnson16-2-4	1808,59	442,67	38400,01	38400,07	275,95	981,24	118,53	442,22	49,11	58,29
johnson32-2-4	38400,00	38400,00	38400,00	38401,18	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
johnson8-2-4	0,02	0,01	0,21	0,27	0,01	0,02	0,01	0,01	0,01	0,01
johnson8-4-4	104,02	5,62	2,75	4,52	2,08	12,21	0,17	5,60	0,20	0,18
keller4	3555,98	330,91	3187,01	2988,21	144,65	614,28	7,30	342,04	9,42	17,59
keller5	38400,00	38400,00	38400,73	38401,48	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
keller6	38400,00	38400,00	38400,40	38400,26	38400,00	38400,00	38400,01	38400,00	38400,00	38400,01
MANN_a27	38400,00	38400,00	38400,27	38400,11	38400,00	38400,00	2170,04	38400,00	1715,20	38400,01
MANN_a45	38400,00	38400,00	38400,24	38400,08	38400,00	38400,00	38400,45	38400,00	38401,81	38400,00
MANN_a81	38400,00	38400,00	38400,14	38400,11	38400,00	38400,00	38400,14	38400,00	38401,91	38400,01
MANN_a9	3953,11	72,72	19,81	17,75	20,15	141,08	0,05	73,97	0,08	0,35

Tabela 66 – continua na próxima página

Tabela 66 – continuação da página anterior										
Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
p_hat1000-1	17965,73	555,75	38401,14	38400,37	345,07	1255,99	154,95	592,01	313,79	404,89
p_hat1000-2	38400,00	38400,00	38400,81	38401,01	38400,00	38400,00	38400,00	38400,00	38400,00	38400,00
p_hat1000-3	38400,00	38400,00	38403,18	38403,06	38400,00	38400,00	38400,00	38400,00	38400,01	38400,02
p_hat1500-1	38400,00	5944,09	38400,93	38401,30	3382,21	13254,73	1060,08	6299,25	1650,19	2450,28
p_hat1500-2	38400,00	38400,00	38405,42	38404,86	38400,00	38400,01	38400,01	38400,00	38400,00	38400,09
p_hat1500-3	38400,00	38400,00	38405,08	38400,57	38400,00	38400,00	38400,02	38400,00	38400,02	38400,00
p_hat300-1	28,87	2,69	239,68	191,77	1,92	6,13	0,81	2,70	5,65	5,80
p_hat300-2	38400,00	6554,11	9302,36	9180,40	311,92	7295,30	12,91	6851,90	18,16	115,22
p_hat300-3	38400,00	38400,00	38400,19	38400,21	38400,00	38400,00	2126,53	38400,00	4560,07	38400,00
p_hat500-1	453,97	26,27	3714,26	3316,40	16,20	57,51	5,92	27,87	30,41	32,39
p_hat500-2	38400,00	38400,00	38400,07	38400,34	38400,00	38400,00	616,11	38400,00	1171,40	14987,52
p_hat500-3	38400,00	38400,00	38400,91	38400,55	38400,00	38400,00	38400,00	38400,00	38400,00	38400,01
p_hat700-1	3432,44	97,95	13653,42	11739,93	65,96	213,46	31,87	101,34	87,85	89,93
p_hat700-2	38400,00	38400,00	38400,32	38400,73	38400,00	38400,00	5868,05	38400,00	10536,45	38400,00
p_hat700-3	38400,00	38400,00	38400,66	38401,61	38400,00	38400,00	38400,01	38400,00	38400,01	38400,00
san1000	38400,00	38400,00	38400,27	38400,12	38400,00	38400,00	358,26	38400,00	3642,93	612,66
san200_.7_1	38400,00	38400,00	6100,95	6071,77	38400,00	38400,00	2,46	38400,00	6,63	7,13
san200_.7_2	38400,00	38400,00	3565,79	3017,89	38400,00	38400,00	2,78	38400,00	4,11	6,09

Tabela 66 – continua na próxima página

Tabela 66 – continuação da página anterior										
Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
san200_.9_1	38400,00	38400,00	1250,63	1237,34	38400,00	38400,01	129,38	38400,00	431,70	913,97
san200_.9_2	38400,00	38400,00	38400,11	38400,28	38400,00	38400,02	301,93	38400,00	948,02	2502,45
san200_.9_3	38400,00	38400,00	38400,62	38400,26	38400,00	38400,00	1712,41	38400,00	96,48	8561,04
san400_.5_1	38400,00	38400,00	2041,90	2069,92	38400,00	38400,00	6,58	38400,00	24,98	19,26
san400_.7_1	38400,00	38400,00	38400,47	38400,04	38400,00	38400,00	163,05	38400,00	571,20	274,31
san400_.7_2	38400,00	38400,00	38400,13	38400,21	38400,00	38400,00	63,86	38400,00	139,88	1755,94
san400_.7_3	38400,00	38400,00	38400,44	38400,19	38400,00	38400,00	889,02	38400,00	1067,32	4930,04
san400_.9_1	38400,00	38400,00	38400,19	38400,30	38400,00	38400,00	10398,81	38400,00	2728,54	38400,01
sanr200_.7	38400,00	2995,25	28738,06	29998,11	1.3	6743,79	113,64	2996,13	148,32	194,69
sanr200_.9	38400,00	38400,00	38400,16	38400,12	38400,00	38400,00	24012,09	38400,00	38400,00	38400,00
sanr400_.5	38400,00	1109,32	38400,15	38400,36	746,72	2398,07	184,34	1140,04	209,96	250,37
sanr400_.7	38400,00	38400,00	38400,15	38400,47	38400,00	38400,01	38400,00	38400,00	38400,00	38400,00

Tabela 66: Tempo de CPU, em segundos, de cada algoritmo para cada instância.

Para analisarmos quais algoritmos tiveram o melhor desempenho, analisamos as 45 instâncias em que pelo menos um dos algoritmos completou sua execução. Já com relação à análise de pior desempenho, podemos analisar apenas 13 instâncias, já que nesses casos precisamos que todos os algoritmos tenham encerrado a execução sobre tal instância.

Podemos ver através da Tabela 67 que em 39 (59%) dos 66 casos, em que pelo menos um dos algoritmos encerrou a execução sobre dada instância, dyn apresentou melhor desempenho. Além disso, foi o único que conseguiu resolver a instância *sanr*200_0.9 para qual os outros algoritmos encerraram sua execução devido ao tempo limite de 38.400 segundos, o algoritmo dyn encontrou a clique máxima em 24.012,09 segundos. Em 6 (9%) instâncias, o algoritmo mcr apresentou melhor desempenho. Quando tratamos dos piores casos, nobound é responsável por 7 das 13 instâncias em que foi possível chegar a essa conclusão, χ foi responsável por 5 e $\chi + \mathsf{df}$ por apenas 1.

Algoritmo	Pior Desempenho	Porcentagem Total
nobound	7	10,61%
χ	5	7,58%
$\chi + \mathsf{df}$	1	1,52%
cp	0	0,00%
mcq	0	0,00%
basic	0	0,00%
mcr	0	0,00%
mcs	0	0,00%
df	0	0,00%
dyn	0	0,00%

Algoritmo	Melhor Desempenho	Porcentagem Total
dyn	39	59,09%
mcr	6	9,09%
nobound	0	0,00%
basic	0	0,00%
χ	0	0,00%
$\chi + \mathsf{df}$	0	0,00%
cp	0	0,00%
df	0	0,00%
mcq	0	0,00%
mcs	0	0,00%

Tabela 67: Nesta tabela foram computados apenas os casos em que todos algoritmos completaram a execução sobre a instância.

Constata-se então que dyn apresentou um melhor desempenho de forma geral para este indicador. Como em uma grande quantidade de instâncias (21), nenhum dos algoritmos completou sua execução, analisamos nesse grupo qual encontrou a maior clique dentro do tempo limite. Na Tabela 68 temos a relação de algoritmos que encontraram a maior clique quando nenhum conseguiu completar sua execução. O símbolo dyn+ indica

que o algoritmo dyn empatou com um ou mais algoritmos.

O algoritmo dyn apresentou o melhor desempenho para o indicador Tempo de CPU.

Instância	Melhor Algoritmo
brock400_1	dyn+
brock400_2	dyn
brock400_3	dyn+
brock400_4	dyn+
brock800_1	dyn+
brock800_2	dyn+
brock800_3	dyn+
brock800_4	dyn
hamming10-4	dyn+
johnson32-2-4	dyn+
keller5	dyn+
keller6	dyn
MANN_a45	dyn+
MANN_a81	dyn+
p_hat1000-2	dyn
p_hat1000-3	mcr e mcs
p_hat1500-2	mcr e mcs
p_hat1500-3	mcr e mcs
p_hat500-3	dyn
p_hat700-3	mcr e mcs
sanr400_.7	dyn

Tabela 68: Melhores algoritmos quanto ao tamanho da clique encontrada em instâncias para as quais nenhum algoritmo completou sua execução.

Na Tabela 69 mostramos qual a porcentagem do tempo demandado pelo algoritmo nobound que cada um dos demais algoritmos necessitou. Quanto menor a porcentagem, menor o Tempo de CPU, ou seja, melhor.

Tabela 69: Porcentagem do tempo que cada algoritmo demandou em relação ao algoritmo nobound.

Instancia	**basic**	χ	χ + **df**	**cp**	**df**	**MCD**	**mcq**	**mcr**	**mcs**
brock200_1	57,5572%	100,0002%	100,0002%	18,2712%	100,0000%	0,8131%	58,6462%	1,2838%	1,5910%
brock200_2	4,0006%	150,3010%	135,1702%	2,6772%	8,9417%	0,8826%	4,0686%	1,4532%	1,7761%
brock200_3	0,8393%	21,8285%	21,9260%	0,4056%	1,9484%	0,0588%	0,8876%	0,0799%	0,1208%
brock200_4	1,7267%	23,8871%	24,7502%	0,9747%	4,0371%	0,1076%	1,7657%	0,1558%	0,1561%
brock400_1	100,0000%	100,0012%	100,0006%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock400_2	100,0000%	100,0002%	100,0019%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock400_3	100,0000%	100,0004%	100,0002%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock400_4	100,0000%	100,0010%	100,0013%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock800_1	100,0000%	100,0015%	100,0003%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock800_2	100,0000%	100,0020%	100,0012%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock800_3	100,0000%	100,0012%	100,0025%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
brock800_4	100,0000%	100,0013%	100,0043%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
c-fat200-1	1,8023%	68,0694%	67,4173%	2,0970%	3,1236%	1,6464%	1,8170%	12,2814%	10,1955%
c-fat200-2	0,0033%	0,0149%	0,0067%	0,0028%	0,0044%	0,0027%	0,0033%	0,0205%	0,0185%
c-fat200-5	0,0251%	0,0621%	0,0457%	100,0000%	0,0027%	0,0022%	0,0267%	0,0077%	0,0073%
c-fat500-1	1,0226%	2,2382%	3,0639%	1,1029%	1,9709%	0,7481%	0,8224%	6,2805%	6,2566%
c-fat500-10	100,0000%	0,1487%	0,0246%	100,0000%	0,0300%	0,0223%	100,0000%	0,0986%	0,0988%
c-fat500-2	0,0017%	0,0042%	0,0051%	0,0021%	0,0034%	0,0014%	0,0015%	0,0167%	0,0167%

Tabela 69 – continua na próxima página

Tabela 69 – continuação da página anterior									
Instancia	**basic**	χ	χ + **df**	**cp**	**df**	**MCD**	**mcq**	**mcr**	**mcs**
c-fat500-5	0,0857%	0,0288%	0,0061%	5,0779%	0,0091%	0,0053%	0,1165%	0,0438%	0,0437%
hamming10-2	100,0000%	9,4962%	6,6707%	100,0000%	100,0006%	1,3729%	100,0000%	0,1927%	0,2042%
hamming10-4	100,0000%	100,0053%	100,0066%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
hamming6-2	0,0473%	0,0024%	0,0014%	0,0200%	0,0967%	0,0002%	0,0484%	0,0002%	0,0022%
hamming6-4	55,1253%	2248,0180%	1783,3113%	46,5713%	132,2227%	33,6830%	54,1766%	46,4093%	37,9219%
hamming8-2	100,0000%	0,1363%	0,0945%	100,0000%	100,0003%	0,0124%	100,0000%	0,0033%	0,0034%
hamming8-4	6,8772%	10,5115%	13,1482%	2,1115%	9,3350%	0,0770%	7,0538%	0,1444%	0,1163%
johnson16-2-4	24,4759%	2123,2043%	2123,2080%	15,2576%	54,2543%	6,5538%	24,4512%	2,7153%	3,2232%
johnson32-2-4	100,0000%	100,0000%	100,0031%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
johnson8-2-4	50,1537%	1134,5095%	1469,2112%	54,7457%	111,5612%	31,2767%	49,7586%	31,4040%	32,6600%
johnson8-4-4	5,4005%	2,6402%	4,3411%	2,0017%	11,7373%	0,1677%	5,3883%	0,1899%	0,1745%
keller4	9,3058%	89,6239%	84,0332%	4,0679%	17,2746%	0,2054%	9,6188%	0,2649%	0,4946%
keller5	100,0000%	100,0019%	100,0038%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
keller6	100,0000%	100,0010%	100,0007%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
MANN_a27	100,0000%	100,0007%	100,0003%	100,0000%	100,0000%	5,6512%	100,0000%	4,4667%	100,0000%
MANN_a45	100,0000%	100,0006%	100,0002%	100,0000%	100,0000%	100,0012%	100,0000%	100,0047%	100,0000%
MANN_a81	100,0000%	190,0759%	224,8362%	100,0000%	100,0000%	100,0004%	100,0000%	100,0050%	100,0000%
MANN_a9	1,8395%	0,5010%	0,4490%	0,5098%	3,5689%	0,0014%	1,8712%	0,0019%	0,0090%
Tabela 69 – continua na próxima página									

Tabela 69 – continuação da página anterior									
Instancia	**basic**	χ	χ + **df**	**cp**	**df**	**MCD**	**mcq**	**mcr**	**mcs**
p_hat1000-1	3,0934%	213,7466%	213,7423%	1,9207%	6,9911%	0,8625%	3,2952%	1,7466%	2,2537%
p_hat1000-2	100,0000%	100,0021%	100,0026%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
p_hat1000-3	100,0000%	100,0083%	100,0080%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
p_hat1500-1	15,4794%	100,0024%	100,0034%	8,8078%	34,5175%	2,7606%	16,4043%	4,2974%	6,3809%
p_hat1500-2	100,0000%	100,0141%	100,0127%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0002%
p_hat1500-3	100,0000%	100,0132%	100,0015%	100,0000%	100,0000%	100,0001%	100,0000%	100,0000%	100,0000%
p_hat300-1	9,3272%	830,1243%	664,1910%	6,6493%	21,2168%	2,7953%	9,3628%	19,5773%	20,0792%
p_hat300-2	17,0680%	24,2249%	23,9073%	0,8123%	18,9982%	0,0336%	17,8435%	0,0473%	0,3001%
p_hat300-3	100,0000%	100,0005%	100,0006%	100,0000%	100,0000%	5,5378%	100,0000%	11,8752%	100,0000%
p_hat500-1	5,7857%	818,1649%	730,5269%	3,5690%	12,6688%	1,3049%	6,1401%	6,6986%	7,1340%
p_hat500-2	100,0000%	100,0002%	100,0009%	100,0000%	100,0000%	1,6044%	100,0000%	3,0505%	39,0300%
p_hat500-3	100,0000%	100,0024%	100,0014%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
p_hat700-1	2,8535%	397,7760%	342,0289%	1,9218%	6,2189%	0,9285%	2,9524%	2,5593%	2,6199%
p_hat700-2	100,0000%	100,0008%	100,0019%	100,0000%	100,0000%	15,2814%	100,0000%	27,4387%	100,0000%
p_hat700-3	100,0000%	100,0017%	100,0042%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%
san1000	100,0000%	100,0007%	100,0003%	100,0000%	100,0000%	0,9330%	100,0000%	9,4868%	1,5955%
san200_.7_1	100,0000%	15,8879%	15,8119%	100,0000%	100,0000%	0,0064%	100,0000%	0,0173%	0,0186%
san200_.7_2	100,0000%	9,2859%	7,8591%	100,0000%	100,0000%	0,0072%	100,0000%	0,0107%	0,0159%

Tabela 69 – continua na próxima página

Tabela 69 – continuação da página anterior									
Instancia	**basic**	χ	χ + **df**	**cp**	**df**	**MCD**	**mcq**	**mcr**	**mcs**
san200_.9_1	100,0000%	3,2568%	3,2222%	100,0000%	100,0000%	0,3369%	100,0000%	1,1242%	2,3801%
san200_.9_2	100,0000%	100,0003%	100,0007%	100,0000%	100,0000%	0,7863%	100,0000%	2,4688%	6,5168%
san200_.9_3	100,0000%	100,0016%	100,0007%	100,0000%	100,0000%	4,4594%	100,0000%	0,2513%	22,2944%
san400_.5_1	100,0000%	5,3175%	5,3904%	100,0000%	100,0000%	0,0171%	100,0000%	0,0651%	0,0502%
san400_.7_1	100,0000%	100,0012%	100,0001%	100,0000%	100,0000%	0,4246%	100,0000%	1,4875%	0,7143%
san400_.7_2	100,0000%	100,0003%	100,0005%	100,0000%	100,0000%	0,1663%	100,0000%	0,3643%	4,5728%
san400_.7_3	100,0000%	100,0011%	100,0005%	100,0000%	100,0000%	2,3152%	100,0000%	2,7795%	12,8386%
san400_.9_1	100,0000%	100,0005%	100,0008%	100,0000%	100,0000%	27,0802%	100,0000%	7,1056%	100,0000%
sanr200_.7	7,8001%	74,8387%	78,1201%	100,0000%	17,5619%	0,2960%	7,8024%	0,3862%	0,5070%
sanr200_.9	100,0000%	100,0004%	100,0003%	100,0000%	100,0000%	62,5315%	100,0000%	100,0000%	100,0000%
sanr400_.5	2,8889%	100,0004%	100,0009%	1,9446%	6,2450%	0,4800%	2,9689%	0,5468%	0,6520%
sanr400_.7	100,0000%	100,0004%	100,0012%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%	100,0000%

Tabela 69: Porcentagem do tempo que cada algoritmo demandou em relação ao algoritmo nobound.

Podemos ver na Tabela 70 a quantidade de instâncias para as quais os algoritmos apresentaram o melhor ou pior desempenho. De todos os 45 casos em que pelo menos um dos algoritmos completou sua execução, o algoritmo dyn é o melhor em 39 (87%) casos enquanto o algoritmo mcr foi melhor em 6 (13%). Esse resultado reforça as constatações feitas anteriormente sobre este indicador, resultando na conclusão de que o algoritmo dyn, dentre os analisados, é o mais apropriado quando buscamos um algoritmo mais rápido para os grafos DIMACS.

Algoritmo	**Pior Desempenho**
χ	8
χ + df	3
mcr	2
df	2
mcs	0
basic	0
cp	0
mcq	0
dyn	0

Algoritmo	**Melhor Desempenho**
dyn	39
mcr	6
df	0
mcq	0
basic	0
χ	0
χ + df	0
cp	0
mcs	0

Tabela 70: Melhores e piores casos da comparação do Tempo de CPU dos algoritmos com o algoritmo nobound.

4.2.5 Passos de *Branching* em conjunto com Tempo de CPU

Por último, analisamos em conjunto os dois indicadores de desempenho.

Para tal análise, dividimos o Tempo de CPU pelo número de Passos de *Branching*. O resultado encontra-se na Tabela 71.

Tabela 71: Relação entre Tempo de CPU e Passos de *Branching*.

Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
brock200_1	0,00005	0,00003	0,07818	0,08430	0,00003	0,00084	0,00068	0,00003	0,00054	0,00075
brock200_2	0,00003	0,00002	0,05405	0,04933	0,00002	0,00058	0,00044	0,00002	0,00068	0,00068
brock200_3	0,00006	0,00002	0,06251	0,06261	0,00002	0,00060	0,00049	0,00002	0,00052	0,00056
brock200_4	0,00006	0,00002	0,06944	0,07196	0,00002	0,00068	0,00044	0,00002	0,00042	0,00056
brock400_1	0,00003	0,00002	0,26750	0,29798	0,00003	0,00075	0,00062	0,00002	0,00060	0,00003
brock400_2	0,00003	0,00003	0,27549	0,27519	0,00003	0,00085	0,00083	0,00003	0,00056	0,00003
brock400_3	0,00002	0,00002	0,28761	0,27482	0,00003	0,00078	0,00003	0,00003	0,00063	0,00003
brock400_4	0,00003	0,00003	0,28872	0,27957	0,00003	0,00076	0,00084	0,00003	0,00070	0,00003
brock800_1	0,00003	0,00002	0,39588	0,37139	0,00002	0,00070	0,00065	0,00002	0,00048	0,00002
brock800_2	0,00003	0,00002	0,46836	0,39647	0,00002	0,00078	0,00065	0,00002	0,00051	0,00002
brock800_3	0,00003	0,00002	0,42098	0,35355	0,00003	0,00077	0,00062	0,00002	0,00052	0,00002
brock800_4	0,00003	0,00002	0,44668	0,33413	0,00003	0,00074	0,00075	0,00002	0,00051	0,00002
c-fat200-1	0,00003	0,00010	0,05330	0,07986	0,00019	0,00370	0,00017	0,00010	0,00150	0,00125
c-fat200-2	0,00003	0,00009	0,01649	0,07328	0,00006	0,00282	0,00031	0,00009	0,00315	0,00284
c-fat200-5	0,00003	0,00007	0,09982	0,13827	0,00006	0,00816	0,00137	0,00007	0,01040	0,00987
c-fat500-1	0,00005	0,00026	0,04190	0,33267	0,00050	0,00522	0,00039	0,00021	0,00350	0,00349
c-fat500-10	0,00011	0,00013	0,22570	1,88619	0,00011	0,03594	0,00572	0,00014	0,05055	0,05067
c-fat500-2	0,00005	0,00008	0,03024	0,39384	0,00019	0,00453	0,00049	0,00007	0,00675	0,00675

Tabela 71 – continua na próxima página

Tabela 71 – continuação da página anterior										
Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
c-fat500-5	0,00005	0,00006	0,08569	0,46759	0,00007	0,01153	0,00163	0,00008	0,01928	0,01924
hamming10-2	0,00020	0,00033	3,55760	4,71738	0,00035	0,02299	0,20895	0,00033	0,07220	0,07652
hamming10-4	0,00002	0,00002	1,01005	0,96683	0,00002	0,00058	0,00178	0,00002	0,00105	0,00105
hamming6-2	0,00003	0,00004	0,01400	0,01686	0,00004	0,00156	0,00053	0,00004	0,00114	0,01273
hamming6-4	0,00002	0,00002	0,00711	0,00627	0,00002	0,00027	0,00010	0,00002	0,00018	0,00015
hamming8-2	0,00006	0,00010	0,20370	0,28136	0,00010	0,00840	0,00929	0,00010	0,00500	0,00507
hamming8-4	0,00003	0,00002	0,11768	0,17571	0,00002	0,00086	0,00072	0,00002	0,00067	0,00083
johnson16-2-4	0,00002	0,00001	0,02017	0,01982	0,00001	0,00016	0,00009	0,00001	0,00008	0,00010
johnson32-2-4	0,00001	0,00001	0,05876	0,04937	0,00001	0,00016	0,00008	0,00001	0,00008	0,00011
johnson8-2-4	0,00001	0,00001	0,00255	0,00339	0,00002	0,00015	0,00006	0,00001	0,00008	0,00011
johnson8-4-4	0,00001	0,00002	0,01948	0,03447	0,00002	0,00051	0,00039	0,00002	0,00068	0,00053
keller4	0,00003	0,00002	0,04952	0,04973	0,00002	0,00040	0,00042	0,00002	0,00042	0,00042
keller5	0,00003	0,00002	0,82324	0,94667	0,00003	0,00079	0,00194	0,00002	0,00164	0,00181
keller6	0,00003	0,00003	0,61142	0,49205	0,00003	0,00072	0,00472	0,00003	0,01246	0,00943
MANN_a27	0,00002	0,00002	0,32924	0,31294	0,00011	0,00027	0,02836	0,00002	0,02256	0,00067
MANN_a45	0,00003	0,00003	0,26769	0,15695	0,00018	0,00037	0,22855	0,00003	0,19250	0,00112
MANN_a81	0,00003	0,00003	35,00456	35,48984	0,00043	0,00050	2,12484	0,00004	1,83926	0,00195
MANN_a9	0,00001	0,00002	0,00961	0,01065	0,00002	0,00024	0,00029	0,00002	0,00053	0,00041

Tabela 71 – continua na próxima página

Tabela 71 – continuação da página anterior										
Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
p_hat1000-1	0,00009	0,00002	0,36258	0,28764	0,00002	0,00074	0,00046	0,00002	0,00078	0,00112
p_hat1000-2	0,00003	0,00003	0,77494	1,02223	0,00004	0,00143	0,00307	0,00003	0,00244	0,00242
p_hat1000-3	0,00003	0,00003	1,64482	1,72010	0,00004	0,00151	0,00333	0,00004	0,00494	0,00398
p_hat1500-1	0,00015	0,00002	0,42567	0,40223	0,00002	0,00085	0,00047	0,00003	0,00065	0,00090
p_hat1500-2	0,00003	0,00003	0,53002	1,71450	0,00004	0,00156	0,00394	0,00004	0,00598	0,00595
p_hat1500-3	0,00003	0,00004	2,59424	2,68235	0,00005	0,00158	0,00475	0,00004	0,01197	0,01272
p_hat300-1	0,00004	0,00002	0,05716	0,04658	0,00002	0,00065	0,00019	0,00002	0,00139	0,00208
p_hat300-2	0,00003	0,00003	0,12738	0,12427	0,00003	0,00129	0,00084	0,00003	0,00182	0,00133
p_hat300-3	0,00003	0,00003	0,17743	0,17913	0,00004	0,00129	0,00170	0,00003	0,00128	0,00197
p_hat500-1	0,00005	0,00002	0,11945	0,10371	0,00002	0,00064	0,00027	0,00002	0,00141	0,00159
p_hat500-2	0,00002	0,00003	0,32324	0,30850	0,00004	0,00135	0,00159	0,00003	0,00137	0,00206
p_hat500-3	0,00003	0,00003	0,45042	0,45471	0,00004	0,00122	0,00334	0,00004	0,00219	0,00228
p_hat700-1	0,00009	0,00002	0,22439	0,20320	0,00002	0,00094	0,00057	0,00003	0,00128	0,00229
p_hat700-2	0,00003	0,00003	0,60509	0,57873	0,00005	0,00154	0,00269	0,00004	0,00218	0,00277
p_hat700-3	0,00003	0,00004	0,87789	0,85267	0,00004	0,00104	0,00436	0,00004	0,00464	0,00487
san1000	0,00002	0,00002	1,33827	1,35713	0,00002	0,00080	0,00142	0,00002	0,00802	0,00486
san200_.7_1	0,00001	0,00001	0,07456	0,07743	0,00003	0,00008	0,00147	0,00001	0,00251	0,00185
san200_.7_2	0,00001	0,00001	0,07506	0,07866	0,00002	0,00011	0,00080	0,00001	0,00128	0,00190

Tabela 71 – continua na próxima página

Tabela 71 – continuação da página anterior										
Instância	**nobound**	**basic**	χ	χ + **df**	**cp**	**df**	**dyn**	**mcq**	**mcr**	**mcs**
san200_.9_1	0,00003	0,00004	0,12771	0,14373	0,00003	0,00123	0,00223	0,00004	0,00129	0,00311
san200_.9_2	0,00003	0,00003	0,10410	0,10696	0,00003	0,00123	0,00289	0,00003	0,00119	0,00178
san200_.9_3	0,00003	0,00003	0,09738	0,10242	0,00003	0,00074	0,00247	0,00003	0,00148	0,00299
san400_.5_1	0,00001	0,00001	0,20778	0,21293	0,00002	0,00021	0,00113	0,00001	0,00622	0,01357
san400_.7_1	0,00001	0,00001	0,28373	0,27694	0,00001	0,00006	0,00190	0,00001	0,00269	0,00287
san400_.7_2	0,00001	0,00001	0,27901	0,27890	0,00010	0,00007	0,00326	0,00001	0,00255	0,00129
san400_.7_3	0,00001	0,00002	0,25190	0,24657	0,00002	0,00024	0,00080	0,00002	0,00128	0,00100
san400_.9_1	0,00002	0,00003	0,33828	0,35640	0,00003	0,00016	0,01381	0,00003	0,01201	0,00378
sanr200_.7	0,00003	0,00002	0,07315	0,07379	0,00070	0,00070	0,00054	0,00002	0,00041	0,00057
sanr200_.9	0,00003	0,00003	0,10329	0,10773	0,00004	0,00112	0,00193	0,00003	0,00140	0,00190
sanr400_.5	0,00005	0,00002	0,16096	0,14331	0,00002	0,00056	0,00037	0,00002	0,00035	0,00050
sanr400_.7	0,00002	0,00002	0,25669	0,25146	0,00003	0,00067	0,00073	0,00002	0,00055	0,00076

Tabela 71: Relação entre Tempo de CPU e Passos de *Branching*.

Temos na Tabela 72 a média dos valores da Tabela 71, ordenados em ordem crescente de tempo.

Algoritmo	**Tempo em Segundos - Média**	σ
nobound	0,000036	0,0031%
basic	0,000039	0,0051%
mcq	0,000039	0,0048%
cp	0,000069	0,0120%
df	0,002201	0,5380%
mcs	0,004560	1,1382%
mcr	0,035453	22,6903%
dyn	0,040842	26,3178%
χ	1,369276	430,4300%
$\chi + \mathsf{df}$	1,647763	437,5300%

Tabela 72: Média dos valores gastos em cada passo de *branching* de cada algoritmo.

Os valores altos de desvio padrão para os algoritmos mcr, dyn, χ e $\chi + \mathsf{df}$ devem-se aos valores obtidos quanto à instância MANN_a81, que destoaram consideravelmente das demais. Isso se deve ao fato de o grafo MANN_a81 possuir 3.321 vértices e alta densidade de arestas, p=0,9988, resultando em 5.506.380 arestas, ou seja, um grafo consideravelmente maior que os demais, produzindo assim indicadores de desempenho de valores que destoam dos demais. Lembramos que os algoritmos χ e $\chi + \mathsf{df}$ efetuam o cálculo de coloração do grafo de 4 formas diferentes para determinar o limitante usando na técnica de *bounding*.

Percebemos através da Tabela 72, que os algoritmos que apresentaram o menor valor foram os algoritmos nobound e basic. Isso indica que os mesmos levaram um tempo relativamente menor para gerar tal quantidade de passos. Isso se explica facilmente, uma vez que esses algoritmos não consomem tempo calculando valores para aplicação da técnica de *bounding*.

Já os algoritmos χ e $\chi + \mathsf{df}$ ocupam as piores posições: casos em que o algoritmo gerou poucos casos de *branching* em relação ao Tempo de CPU total gasto. Ambos os algoritmos, χ e $\chi + \mathsf{df}$, empregam técnicas baseadas em coloração para definir o limitante usado na fase de *bounding* do algoritmo. A descrição da técnica empregada em ambos os algoritmos encontra-se nas seções 3.3.6 e 3.3.7.

Durante a análise de número de Passos de *Branching*, chegamos à conclusão de que os algoritmos χ e $\chi + \mathsf{df}$ geram o menor número de passos em uma grande parcela das instâncias. Entretanto, percebemos na análise de Tempo de CPU que, excluindo o algoritmo nobound, os algoritmos anteriormente mencionados são responsáveis por uma grande parcela dos piores casos.

Tudo isso se deve ao fato de as operações de coloração desempenhadas por ambos os algoritmos necessitar de grande Tempo de CPU, e serem realizada diversas vezes durante a execução dos mesmos.

Percebemos então que a técnica empregada é bastante eficiente quanto à redução de Passos de *Branching*, mas necessita de muito Tempo de CPU. O algoritmo dyn, considerado o melhor quanto ao indicador Tempo de CPU, é o de antepenúltimo pior desempenho, de acordo com a Tabela 72. E por fim, o algoritmo que apresentou o melhor custo/benefício entre Tempo de CPU e Passos de *Branching* foi o algoritmo mcq (dentre os que aplicam técnicas de *bounding*), o que não significa dizer que o algoritmo é o mais rápido, uma vez que um baixo valor, neste caso, indica apenas que o algoritmo não gasta uma grande parcela do seu Tempo de CPU geral calculando limitantes para empregar na fase de *bounding*.

4.2.6 Validação

Temos na Tabela 73 o resultado dos testes de validação. Como a quantidade de instâncias é grande, escolhemos uma amostra das instâncias para realizarmos os testes de validação. Essa amostra foi escolhida da seguinte forma, a partir de cada família foram selecionadas duas instâncias, a maior e a menor, considerando o número de vértices e arestas. A descrição de cada instância encontra-se na Seção 3.2.2. Podemos perceber que os resultados não apresentam grande variação. As instâncias concluídas pela servidora *Latrappe*, usada no experimento, foram também concluídas pela servidora *Achel*. As instâncias não concluídas dentro do tempo limite para a servidora *Latrappe* apresentaram o mesmo comportamento na servidora *Achel*.

Em 6 casos, anotados em **negrito**, a servidora *Achel* encontrou uma clique maior do que a encontrada na servidora *Latrappe*. Em todos esses casos, para cada algoritmo, o número de Passos de *Branching* foi maior na servidora *Achel*.

Algoritmo	Instância	ω (Achel)	ω (Latrappe)	T (Achel)	T (Latrappe)	t (Achel)	t (Latrappe)
basic	brock200_1	21	21	876.403.943	876.403.943	22.723,9061	22.101,9635
	brock800_4	20	20	1.523.193.243	1.620.236.354	38.400,0001	38.400,0000
	c-fat200-1	12	12	811	811	0,0631	0,0833
	c-fat500-10	126	126	275.722.183	286.282.778	38.400,0004	38.400,0000
	hamming10-4	32	32	1.796.608.187	1.845.277.361	38.400,0002	38.400,0001
	hamming6-2	32	32	508.981	508.981	18,8428	18,1629
	johnson32-2-4	16	16	3.040.816.706	3.118.368.201	38.400,0000	38.400,0000
	johnson8-2-4	4	4	717	717	0,0112	0,0091
	keller4	11	11	19.055.823	19.055.823	340,0931	330,9121
	keller6	44	44	1.835.286.834	1.423.791.481	38.400,0000	38.400,0001
	MANN_a81	1096	1096	1.234.353.148	1.130.617.479	38.400,0001	38.400,0001
	MANN_a9	16	16	4.521.439	4.521.439	69,8440	72,7189
	p_hat1500-3	**53**	51	1.281.555.450	1.033.953.930	38.400,0001	38.400,0000
	p_hat300-1	8	8	131.347	131.347	2,0057	2,6930
	san1000	9	9	1.861.073.133	1.786.797.847	38.400,0002	38.400,0000
	san200_0.7_1	17	17	3.165.739.989	2.995.389.744	38.400,0001	38.400,0001
	sanr200_0.7	18	18	127.231.185	127.231.185	3.069,0028	2.995,2490
	sanr400_0.7	21	21	1.634.760.055	1.655.058.972	38.400,0000	38.400,0001

Tabela 73 – continua na próxima página

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
χ	brock200_1	21	21	505.485	491.164	38.400,0061	38.400,0841
	brock800_4	19	19	85.894	85.968	38.400,6548	38.400,5067
	c-fat200-1	12	12	59	59	3,0827	3,1445
	c-fat500-10	126	126	253	253	54,6898	57,1023
	hamming10-4	32	32	39.076	38.020	38.401,2946	38.402,0200
	hamming6-2	32	32	65	65	0,9094	0,9102
	johnson32-2-4	16	16	681.735	653.465	38.400,3428	38.400,0000
	johnson8-2-4	4	4	81	81	0,2298	0,2068
	keller4	11	11	64.357	64.357	3.060,4660	3.187,0094
	keller6	44	44	78.536	62.805	38.400,7065	38.400,3977
	MANN_a81	1096	1096	1.097	1.097	69.433,6113	38.400,0024
	MANN_a9	16	16	2.061	2.061	17,9411	19,8070
	p_hat1500-3	**58**	57	17.557	14.804	38.400,9374	38.405,0793
	p_hat300-1	8	8	4.193	4.193	210,1489	239,6763
	san1000	10	10	33.763	28.694	38.400,1881	38.400,2668
	san200_0.7_1	30	30	81.289	81.829	5.714,2912	6.100,9520
	sanr200_0.7	18	18	392.881	392.881	28.506,5297	28.738,0627

Tabela 73 – continua na próxima página

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	19	19	153.011	149.595	38.400,2815	38.400,1478
$\chi + \mathbf{df}$	brock200_1	21	21	525.038	455.516	38.400,0220	38.400,0911
	brock800_4	18	18	148.313	114.930	38.400,0939	38.401,6401
	c-fat200-1	12	12	39	39	2,3755	3,1144
	c-fat500-10	126	126	5	5	2,7104	9,4310
	hamming10-4	33	33	45.625	39.720	38.402,7254	38.402,5187
	hamming6-2	32	32	33	33	0,4448	0,5563
	johnson32-2-4	16	16	787.455	777.837	38.400,0003	38.401,1804
	johnson8-2-4	4	4	79	79	0,1823	0,2677
	keller4	11	11	60.083	60.083	2.744,2851	2.988,2056
	keller6	42	42	98.536	78.042	38.400,2975	38.400,2557
	MANN_a81	1095	1095	1.082	1.082	38.400,4340	38.400,0040
	MANN_a9	16	16	1.667	1.667	14,8219	17,7506
	p_hat1500-3	**57**	56	18.825	14.316	38.400,6887	38.400,5686
	p_hat300-1	8	8	4.117	4.117	144,1586	191,7675
	san1000	10	10	29.089	28.295	38.401,8993	38.400,1154
	san200_0.7_1	30	30	78.415	78.415	5.707,1910	6.071,7742
	sanr200_0.7	18	18	406.553	406.553	28.706,0079	29.998,1074
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	19	19	132.694	152.709	38.400,4590	38.400,4732
cp	brock200_1	21	21	258.012.511	258.012.511	6.900,2007	7.016,1390
	brock800_4	20	20	1.496.013.699	1.489.801.507	38.400,0000	38.400,0000
	c-fat200-1	12	12	515	515	0,1138	0,0969
	c-fat500-10	124	124	368.614.352	346.751.948	38.400,0002	38.400,0001
	hamming10-4	32	32	1.650.185.183	1.672.257.761	38.400,0000	38.400,0001
	hamming6-2	32	32	203.817	203.817	7,6067	7,6936
	johnson32-2-4	16	16	3.127.228.489	3.078.378.672	38.400,0000	38.400,0000
	johnson8-2-4	4	4	569	569	0,0086	0,0100
	keller4	11	11	8.518.051	8.518.051	143,0875	144,6540
	keller6	39	39	1.486.012.930	1.412.480.092	38.400,0000	38.400,0001
	MANN_a81	438	438	93.897.445	88.513.354	38.400,0002	38.400,0015
	MANN_a9	16	16	1.216.045	1.216.045	19,2891	20,1528
	p_hat1500-3	**47**	46	918.611.384	841.237.399	38.400,0001	38.400,0000
	p_hat300-1	8	8	95.753	95.753	1,3916	1,9198
	san1000	**9**	8	1.871.161.632	1.726.549.262	38.400,0000	38.400,0000
	san200_0.7_1	15	15	1.571.665.168	1.211.889.697	38.400,0001	38.400,0000
	sanr200_0.7	18	18	55.196.725	55.196.725	1.326,7202	1.420,7600
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	21	21	1.424.702.500	1.449.529.234	38.400,0001	38.400,0000
df	brock200_1	21	21	47.484.223	45.487.955	38.400,0029	38.400,0000
	brock800_4	20	20	53.235.510	51.983.014	38.400,0005	38.400,0009
	c-fat200-1	12	12	39	39	0,1619	0,1443
	c-fat500-10	126	126	321	321	11,3132	11,5363
	hamming10-4	33	33	66.877.320	65.666.598	38.400,0009	38.400,0018
	hamming6-2	32	32	23.749	23.749	36,7008	37,1210
	johnson32-2-4	16	16	246.958.088	241.752.422	38.400,0002	38.400,0013
	johnson8-2-4	4	4	133	133	0,0204	0,0203
	keller4	11	11	1.542.373	1.542.373	608,7034	614,2826
	keller6	42	42	60.452.849	53.540.849	38.400,0006	38.400,0002
	MANN_a81	1096	1096	82.566.157	77.062.739	38.400,0006	38.400,0014
	MANN_a9	16	16	590.077	590.077	129,3492	141,0810
	p_hat1500-3	55	55	26.387.299	24.327.751	38.400,0049	38.400,0041
	p_hat300-1	8	8	9.463	9.463	4,3671	6,1258
	san1000	9	9	53.911.720	48.139.216	38.400,0033	38.400,0048
	san200_0.7_1	16	16	483.420.730	475.486.269	38.400,0054	38.400,0009
	sanr200_0.7	18	18	9.606.435	9.606.435	6.844,4338	6.743,7862
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	20	20	57.689.356	57.152.713	38.400,0002	38.400,0075
dyn	brock200_1	21	21	457.353	457.353	306,9260	312,2400
	brock800_4	20	21	61.228.300	50.966.767	38.400,0009	38.400,0014
	c-fat200-1	12	12	437	437	0,0768	0,0761
	c-fat500-10	126	126	1.493	1.493	8,5353	8,5443
	hamming10-4	40	40	21.990.935	21.561.034	38.400,0011	38.400,0002
	hamming6-2	32	32	127	127	0,0749	0,0677
	johnson32-2-4	16	16	502.149.605	494.069.783	38.400,0163	38.400,0001
	johnson8-2-4	4	4	95	95	0,0056	0,0057
	keller4	11	11	17.469	17.469	7,0710	7,3023
	keller6	51	51	8.335.863	8.136.128	38.400,0038	38.400,0063
	MANN_a81	1100	1100	18.511	18.072	38.403,7066	38.400,1380
	MANN_a9	16	16	191	191	0,0544	0,0547
	p_hat1500-3	61	61	8.351.465	8.084.665	38.400,0009	38.400,0219
	p_hat300-1	8	8	4.325	4.325	0,7921	0,8071
	san1000	15	15	251.937	251.937	328,2758	358,2579
	san200_0.7_1	30	30	1.681	1.681	2,3523	2,4646
	sanr200_0.7	18	18	210.471	210.471	119,3624	113,6450
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	21	21	55.528.313	52.288.774	38.400,0013	38.400,0031
mcq	brock200_1	21	21	876.403.943	876.403.943	22.801,8044	22.520,1511
	brock800_4	20	20	1.541.253.026	1.572.500.046	38.400,0001	38.400,0001
	c-fat200-1	12	12	811	811	0,0569	0,0839
	c-fat500-10	126	126	280.099.256	282.162.232	38.400,0001	38.400,0002
	hamming10-4	32	32	1.784.497.016	1.779.980.222	38.400,0000	38.400,0001
	hamming6-2	32	32	508.981	508.981	18,1587	18,5681
	johnson32-2-4	16	16	3.028.838.207	3.016.086.146	38.400,0000	38.400,0000
	johnson8-2-4	4	4	717	717	0,0081	0,0091
	keller4	11	11	19.055.823	19.055.823	331,8431	342,0430
	keller6	44	44	1.413.686.255	1.298.588.976	38.400,0000	38.400,0000
	MANN_a81	1096	1096	1.115.112.221	1.079.417.006	38.400,0000	38.400,0001
	MANN_a9	16	16	4.521.439	4.521.439	72,2653	73,9686
	p_hat1500-3	51	51	929.249.343	935.694.895	38.400,0001	38.400,0002
	p_hat300-1	8	8	131.347	131.347	2,7357	2,7033
	san1000	9	9	1.864.465.567	1.704.596.479	38.400,0001	38.400,0001
	san200_0.7_1	17	17	3.132.110.341	2.982.284.485	38.400,0000	38.400,0000
	sanr200_0.7	18	18	127.231.185	127.231.185	2.832,8951	2.996,1335
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	21	21	1.621.622.594	1.629.072.833	38.400,0000	38.400,0000
mcr	brock200_1	21	21	917.233	917.233	470,5124	492,9881
	brock800_4	20	20	79.184.337	76.024.151	38.400,0025	38.400,0021
	c-fat200-1	12	12	377	377	0,5189	0,5673
	c-fat500-10	126	126	749	749	36,8787	37,8626
	hamming10-4	40	40	37.259.963	36.526.945	38.400,0003	38.400,0038
	hamming6-2	32	32	65	65	0,0286	0,0743
	johnson32-2-4	16	16	491.432.843	483.182.549	38.400,0000	38.400,0003
	johnson8-2-4	4	4	73	73	0,0058	0,0057
	keller4	11	11	22.311	22.311	9,0378	9,4204
	keller6	50	50	2.952.226	3.081.827	38.400,0087	38.400,0038
	MANN_a81	1100	1100	22.697	20.879	38.407,7345	38.401,9056
	MANN_a9	16	16	143	143	0,0730	0,0757
	p_hat1500-3	88	88	3.628.252	3.208.352	38.400,0248	38.400,0166
	p_hat300-1	8	8	4.065	4.065	3,7774	5,6524
	san1000	15	15	454.323	454.323	3.320,9628	3.642,9332
	san200_0.7_1	30	30	2.641	2.641	5,9860	6,6315
	sanr200_0.7	18	18	360.205	360.205	147,5332	148,3188
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	21	21	71.286.660	69.227.799	38.400,0014	38.400,0013
mcs	brock200_1	21	21	810.885	810.885	627,6218	610,9301
	brock800_4	**21**	20	1.752.312.431	1.560.819.523	38.400,0057	38.400,0000
	c-fat200-1	12	12	377	377	0,5233	0,4710
	c-fat500-10	126	126	749	749	37,3711	37,9494
	hamming10-4	37	37	39.045.634	36.480.716	38.400,0031	38.400,0032
	hamming6-2	32	32	65	65	0,0320	0,8274
	johnson32-2-4	16	16	371.348.835	350.365.121	38.400,0019	38.400,0011
	johnson8-2-4	4	4	55	55	0,0067	0,0060
	keller4	11	11	41.603	41.603	17,3011	17,5881
	keller6	50	50	4.307.114	4.071.405	38.400,0099	38.400,0065
	MANN_a81	1096	1096	20.963.401	19.734.122	38.400,0103	38.400,0069
	MANN_a9	16	16	867	867	0,3495	0,3541
	p_hat1500-3	88	88	3.460.950	3.018.965	38.400,0137	38.400,0006
	p_hat300-1	8	8	2.793	2.793	4,0257	5,7973
	san1000	15	15	125.941	125.941	504,0685	612,6612
	san200_0.7_1	30	30	3.847	3.847	6,3325	7,1301
	sanr200_0.7	18	18	338.665	338.665	197,4562	194,6892
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	21	21	49.818.037	50.360.317	38.400,0012	38.400,0028
nobound	brock200_1	19	19	840.276.672	792.252.455	38.400,0000	38.400,0003
	brock800_4	19	19	1.371.516.569	1.494.195.878	38.400,0001	38.400,0003
	c-fat200-1	12	12	162.561	162.561	5,6753	4,6195
	c-fat500-10	126	126	337.146.098	355.706.421	38.400,0002	38.400,0001
	hamming10-4	32	32	1.439.235.740	1.862.287.451	38.400,0000	38.400,0000
	hamming6-2	32	32	1.400.104.644	1.525.409.292	38.400,0000	38.400,0000
	johnson32-2-4	16	16	3.610.544.012	3.429.861.301	38.400,0001	38.400,0002
	johnson8-2-4	4	4	1.527	1.527	0,0269	0,0182
	keller4	11	11	127.267.065	127.267.065	3.223,1408	3.555,9808
	keller6	42	42	111.148.758	1.216.523.202	38.400,0069	38.400,0001
	MANN_a81	1096	1096	103.784.313	1.120.079.341	38.400,0004	38.400,0009
	MANN_a9	16	16	320.505.441	320.505.441	4.109,5928	3.953,1104
	p_hat1500-3	46	46	1.187.924.324	1.217.544.104	38.400,0008	38.400,0006
	p_hat300-1	8	8	734.645	734.645	23,8024	28,8723
	san1000	9	9	1.500.497.059	1.775.991.073	38.400,0000	38.400,0001
	san200_0.7_1	16	16	3.099.882.079	3.073.877.821	38.400,0002	38.400,0000
	sanr200_0.7	18	18	1.137.209.322	1.388.680.436	38.400,0000	38.400,0000
Tabela 73 – continua na próxima página							

Tabela 73 – continuação da página anterior							
Algoritmo	**Instância**	ω **(Achel)**	ω **(Latrappe)**	T **(Achel)**	T **(Latrappe)**	t **(Achel)**	t **(Latrappe)**
	sanr400_0.7	19	19	1.654.192.045	1.819.100.184	38.400,0004	38.400,0001

Tabela 73: Resultado dos testes de validação juntamente com os resultados do experimento.

4.2.7 Outras Considerações

Na Tabela 74, ordenada de forma decrescente quanto a porcentagem de instâncias concluídas, temos o número de casos em um dos algoritmos utilizados conseguiu encerrar sua execução para uma instância de tal família.

Família	**Instâncias Concluídas**	**Total de Instâncias**	**Porcentagem De Instâncias Concluídas**
c-fat	62	70	89%
johnson	28	40	70%
p_hat	61	150	41%
keller	12	30	40%
sanr	16	40	40%
hamming	38	100	38%
san	40	110	36%
MANN	12	40	30%
brock	35	120	29%

Tabela 74: Quantidade de testes concluídos para cada família.

Podemos perceber assim que a família c-fat é a de mais fácil resolução, uma vez que de 70 casos de teste (7 instâncias testadas com 10 algoritmos), 62 testes foram concluídos dentro do tempo limite. Já a família brock é a de resolução mais difícil, uma vez que teve apenas 29% dos testes (12 instâncias testadas com 10 algoritmos) concluídos dentro do tempo limite.

4.3 GRAFOS DE MOON-MOSER

Temos nesta seção os resultados para a família de grafos de Moon-Moser. Tais resultados são referentes à servidora *Latrappe* e validados na servidora *Achel* e foram obtidos através da aplicação do Modelo de *Design* definido na Seção 3.4.3. A definição dessa família de grafos pode ser encontrada na Seção 3.2.

Mostramos na Tabela 75 a carga em que o sistema se encontrava no início e no encerramento dos testes de todos as instâncias para cada algoritmo.

Memória RAM Livre		
Algoritmo	Início da Execução	Fim da Execução
nobound	71,29%	65,44%
basic	68,29%	63,14%
cp	63,12%	61,01%
df	56,26%	51,40%
χ	52,90%	54,32%
χ + **df**	45,12%	41,26%
mcq	64,91%	61,63%
dyn	61,45%	56,24%
mcr	51,94%	42,34%
mcs	53,88%	50,44%

Tabela 75: Porcentagem de memória RAM livre no início e na conclusão dos testes.

4.3.1 Passos de *Branching*

Iniciamos a análise pelo indicador Passos de *Branching*. Seguimos o Modelo de *Design* descrito na Figura 32. Temos o resultado da execução do experimento na servidora *Latrappe* nas tabelas 76 e 77. Para n=200, o tamanho da clique encontrada foi de 67, e para n=1350, 450.

n=**200**	T
χ	135
χ + df	135
mcr	265
mcs	265
dyn	399
mcq	399

Tabela 76: Passos de *Branching* para n=200

n=**1350**	T
χ	901
$\chi + \mathsf{df}$	901
mcr	901
mcs	901
dyn	2699
mcq	2699

Tabela 77: Passos de *Branching* para n=1350

Se analisássemos apenas a Tabela 76, chegaríamos a conclusão de que os algoritmos χ e $\chi + \mathsf{df}$ são isolados os mais eficientes quanto à geração de Passos de *Branching*. Entretanto, quando analisamos a Tabela 77, percebemos que este não é o caso. Os algoritmos dyn e mcq prevalecem como os piores, mas os algoritmos χ e $\chi + \mathsf{df}$ têm seu desempenho igualado ao mcr e mcs.

Recaímos então na mesma situação. Vários algoritmos apresentando o mesmo resultado. Suspeitamos então que, a medida que o nível de n cresça, será possível perceber a diferença no desempenho entre esses quatro algoritmos. Todavia, os valores de Tempo de CPU para os níveis de n obtidos já se mostram altos, tornando-se inviável, para o tempo de que dispomos, realizar testes com níveis maiores de n.

Como os grafos de Moon-Moser possuem uma estrutura claramente definida, analisamos os resultados obtidos pelo experimento de descoberta definido anteriormente na Seção 3.4.3 procurando uma relação entre o número de Passos de *Branching* e o número de vértices do grafo.

Observando os resultados do experimento de descoberta podemos constatar a presença de um padrão no valor de T. Como a quantidade de arestas é dada em função do número de vértices, buscamos então expressar T como função do número de vértices.

Tendo isso em mente, analisamos os resultados do experimento de descoberta de cada algoritmo através do resultado da divisão do número de Passos de *Branching* pelo número de vértices do grafo. Tais resultados encontram-se nas tabelas 78, 79, 80, 81, 82 e 83. Logo abaixo de cada tabela mostramos a fórmula encontrada para calcular o valor do número de Passos de *Branching* para cada algoritmo em função do número de vértices do grafo utilizado. Como os grafos de Moon-Moser são formados por cliques de tamanho 3,

na maioria dos algoritmos existe uma diferença entre o número de passos executado para grafos com número de vértices divisível por 3 e não divisível por 3.

n	T	**T/n**
200	135	0,6750
250	169	0,6760
300	201	0,6700
350	235	0,6714
400	269	0,6725
450	301	0,6689
500	335	0,6700
550	369	0,6709
600	401	0,6683
650	435	0,6692
700	469	0,6700
750	501	0,6680
800	535	0,6688
850	569	0,6694
900	601	0,6678
950	635	0,6684
1000	669	0,6690
1100	735	0,6682
1150	769	0,6687
1200	801	0,6675
1300	869	0,6685
1350	901	0,6674

Tabela 78: A razão T/n para o algoritmo χ

Para o algoritmo χ o número de Passos de *Branching* é dado pelas seguintes fórmulas:

$$T = \frac{(n * 2)}{3} + 1 \tag{4.1}$$

$$T = \lceil \frac{n * 2}{3} \rceil + 1 \tag{4.2}$$

$$T = \lceil \frac{n * 2}{3} \rceil + 2 \tag{4.3}$$

Usamos a fórmula 4.1 quando n é divisível por 3.

Usamos a fórmula 4.2 quando $n + 1$ é divisível por 3.

Usamos a fórmula 4.3 quando $n + 2$ é divisível por 3.

n	T	**T/n**
200	135	0,6800
250	169	0,6800
300	201	0,6700
350	235	0,6700
400	269	0,6700
450	301	0,6689
500	335	0,6700
550	369	0,6709
600	401	0,6683
650	435	0,6692
700	469	0,6700
750	501	0,6680
800	535	0,6688
850	569	0,6694
900	601	0,6678
950	635	0,6684
1000	669	0,6690
1050	701	0,6676
1100	735	0,6682
1150	769	0,6687
1200	801	0,6675
1250	835	0,6680

Tabela 79 – continua na próxima página

Tabela 79 – continuação da página anterior		
n	T	**T/n**
1300	869	0,6685
1350	901	0,6674

Tabela 79: A razão T/n para o algoritmo $\chi + \mathsf{df}$

Para o algoritmo $\chi + \mathsf{df}$ o número de Passos de *Branching* é dado pelas seguintes fórmulas:

$$T = \frac{(n * 2)}{3} + 1 \tag{4.4}$$

$$T = \lceil \frac{n * 2}{3} \rceil + 1 \tag{4.5}$$

$$T = \lceil \frac{n * 2}{3} \rceil + 2 \tag{4.6}$$

Usamos a fórmula 4.4 quando n é divisível por 3.

Usamos a fórmula 4.5 quando $n + 1$ é divisível por 3.

Usamos a fórmula 4.6 quando $n + 2$ é divisível por 3.

n	T	**T/n**
200	399	1,99500
250	499	1,99600
300	599	1,99667
350	699	1,99714
400	799	1,99750
450	899	1,99778
500	999	1,99800
550	1099	1,99818
600	1199	1,99833
650	1299	1,99846
700	1399	1,99857
Tabela 80 – continua na próxima página		

Tabela 80 – continuação da página anterior		
n	T	**T/n**
750	1499	1,99867
800	1599	1,99875
850	1699	1,99882
900	1799	1,99889
950	1899	1,99895
1000	1999	1,99900
1050	2099	1,99905
1100	2199	1,99909
1150	2299	1,99913
1200	2399	1,99917
1250	2499	1,99920
1300	2599	1,99923
1350	2699	1,99926
1400	2799	1,99929
1450	2899	1,99931
1500	2999	1,99933
1550	3099	1,99935
1600	3199	1,99938
1650	3299	1,99939
1700	3399	1,99941
1750	3499	1,99943
1800	3599	1,99944
1850	3699	1,99946
1900	3799	1,99947
1950	3899	1,99949
2000	3999	1,99950
2050	4099	1,99951
2100	4199	1,99952

Tabela 80 – continua na próxima página

Tabela 80 – continuação da página anterior		
n	T	**T/n**
2150	4299	1,99953
2200	4399	1,99955
2250	4499	1,99956
2300	4599	1,99957
2350	4699	1,99957
2400	4799	1,99958
2450	4899	1,99959
2500	4999	1,99960
2550	5099	1,99961
2600	5199	1,99962
2650	5299	1,99962
2700	5399	1,99963
2750	5499	1,99964
2800	5599	1,99964
2900	5799	1,99966
2950	5899	1,99966
3000	5999	1,99967
3050	6099	1,99967
3100	6199	1,99968
3150	6299	1,99968
3200	6399	1,99969
3250	6499	1,99969
3300	6599	1,99970
3350	6699	1,99970
3400	6799	1,99971

Tabela 80: A razão T/n para o algoritmo dyn

Para o algoritmo dyn o número de Passos de *Branching* é dado por uma única

fórmula, 4.7, inexistindo diferenciamento entre n divisível por 3 ou não:

$$T = (n * 2) - 1 \tag{4.7}$$

n	T	**T/n**
200	399	1,9950
250	499	1,9960
300	599	1,9967
350	699	1,9971
400	799	1,9975
450	899	1,9978
500	999	1,9980
550	1099	1,9982
600	1199	1,9983
650	1299	1,9985
700	1399	1,9986
750	1499	1,9987
800	1599	1,9988
850	1699	1,9988
900	1799	1,9989
950	1899	1,9989
1000	1999	1,9990
1050	2099	1,9990
1100	2199	1,9991
1150	2299	1,9991
1200	2399	1,9992
1250	2499	1,9992
1300	2599	1,9992
1350	2699	1,9993
1400	2799	1,9993
1450	2899	1,9993

Tabela 81 – continua na próxima página

Tabela 81 – continuação da página anterior		
n	T	**T/n**
1500	2999	1,9993
1550	3099	1,9994
1600	3199	1,9994
1650	3299	1,9994
1700	3399	1,9994
1750	3499	1,9994
1800	3599	1,9994
1850	3699	1,9995
1900	3799	1,9995
1950	3899	1,9995
2000	3999	1,9995
2050	4099	1,9995
2100	4199	1,9995
2150	4299	1,9995
2200	4399	1,9995
2250	4499	1,9996
2300	4599	1,9996
2350	4699	1,9996
2400	4799	1,9996
2450	4899	1,9996
2500	4999	1,9996
2550	5099	1,9996
2600	5199	1,9996
2650	5299	1,9996
2700	5399	1,9996
2750	5499	1,9996
2800	5599	1,9996
2850	5699	1,9996
Tabela 81 – continua na próxima página		

Tabela 81 – continuação da página anterior		
n	T	**T/n**
2900	5799	1,9997
2950	5899	1,9997
3000	5999	1,9997
3050	6099	1,9997
3100	6199	1,9997
3150	6299	1,9997
3200	6399	1,9997
3250	6499	1,9997
3300	6599	1,9997

Tabela 81: A razão T/n para o algoritmo mcq

Para o algoritmo mcq o número de Passos de *Branching* é dado por uma única fórmula, 4.8, inexistindo diferenciamento entre n divisível por 3 ou não:

$$T = (n * 2) - 1 \tag{4.8}$$

n	T	**T/n**
200	265	1,3250
250	331	1,3240
300	201	0,6700
350	465	1,3286
400	531	1,3275
450	301	0,6689
500	665	1,3300
550	731	1,3291
600	401	0,6683
650	865	1,3308
700	931	1,3300
750	501	0,6680
Tabela 82 – continua na próxima página		

Tabela 82 – continuação da página anterior		
n	T	**T/n**
800	1065	1,3313
850	1131	1,3306
900	601	0,6678
950	1265	1,3316
1000	1331	1,3310
1050	701	0,6676
1100	1465	1,3318
1150	1531	1,3313
1200	801	0,6675
1250	1665	1,3320
1300	1731	1,3315
1350	901	0,6674
1400	1865	1,3321
1450	1931	1,3317
1500	1001	0,6673
1550	2065	1,3323
1600	2131	1,3319
1650	1101	0,6673
1700	2265	1,3324
1750	2331	1,3320
1800	1201	0,6672
1850	2465	1,3324
1900	2531	1,3321
1950	1301	0,6672
2000	2665	1,3325
2050	2731	1,3322
2100	1401	0,6671
2150	2865	1,3326
Tabela 82 – continua na próxima página		

Tabela 82 – continuação da página anterior		
n	T	**T/n**

Tabela 82: A razão T/n para o algoritmo mcr

Para o algoritmo mcr o número de Passos de *Branching* é dado pelas seguintes fórmulas:

$$T = \frac{(n * 2)}{3} + 1 \tag{4.9}$$

$$T = \lfloor n + \frac{n}{3} \rfloor - 1 \tag{4.10}$$

$$T = \lfloor n + \frac{n}{3} \rfloor - 2 \tag{4.11}$$

Usamos a fórmula 4.9 caso n seja divisível por 3.

Usamos a fórmula 4.10 caso n+1 seja divisível por 3.

Usamos a fórmula 4.11 caso n+2 seja divisível por 3.

n	T	**T/n**
200	265	1,3250
250	331	1,3240
300	201	0,6700
350	465	1,3286
400	531	1,3275
450	301	0,6689
500	665	1,3300
550	731	1,3291
600	401	0,6683
650	865	1,3308
700	931	1,3300
750	501	0,6680
800	1065	1,3313
Tabela 83 – continua na próxima página		

Tabela 83 – continuação da página anterior		
n	T	**T/n**
850	1131	1,3306
900	601	0,6678
950	1265	1,3316
1000	1331	1,3310
1050	701	0,6676
1100	1465	1,3318
1150	1531	1,3313
1200	801	0,6675
1250	1665	1,3320
1300	1731	1,3315
1345	1791	1,3316
1400	1865	1,3321
1450	1931	1,3317
1500	1001	0,6673
1550	2065	1,3323
1600	2131	1,3319
1650	1101	0,6673
1700	2265	1,3324
1750	2331	1,3320
1800	1201	0,6672
1850	2465	1,3324
1900	2531	1,3321
1950	1301	0,6672
2000	2665	1,3325

Tabela 83: A razão T/n para o algoritmo mcs

Para o algoritmo mcs o número de Passos de *Branching* é dado pelas seguintes fórmulas:

$$T = \frac{(n * 2)}{3} + 1 \tag{4.12}$$

$$T = \lfloor n + \frac{n}{3} \rfloor - 1 \tag{4.13}$$

$$T = \lfloor n + \frac{n}{3} \rfloor - 2 \tag{4.14}$$

Usamos a fórmula 4.12 caso n seja divisível por 3.

Usamos a fórmula 4.13 caso n+1 seja divisível por 3.

Usamos a fórmula 4.14 caso n+2 seja divisível por 3.

Com base nas fórmulas obtidas anteriormente, montamos a Tabela 84:

Algoritmos	**n divisível por 3**	**n+1 divisível por 3**	**n+2 divisível por 3**
χ e $\chi + \mathsf{df}$	$T = \frac{(n*2)}{3} + 1$	$T = \lceil \frac{n*2}{3} \rceil + 1$	$T = \lceil \frac{n*2}{3} \rceil + 2$
mcr e mcs	$T = \frac{(n*2)}{3} + 1$	$T = \lfloor n + \frac{n}{3} \rfloor - 1$	$T = \lfloor n + \frac{n}{3} \rfloor - 2$
dyn e mcq	$T = (n*2) - 1$	$T = (n*2) - 1$	$T = (n*2) - 1$

Tabela 84: Fórmulas usadas para calcular os Passos de *Branching* em função do número do vértices n.

Percebemos inicialmente que os algoritmos χ e $\chi + \mathsf{df}$ apresentam a mesma fórmula. Bem como as duplas dyn e mcq e mcr e mcs. Ou seja, geram exatamente os mesmos números de Passos de *Branching*.

Reproduzimos a Figura 8 apresentada inicialmente na Seção 3.3. Através dela percebemos que os algoritmos χ e $\chi + \mathsf{df}$ possuem origem direta do algoritmo cp, o algoritmo dyn origina-se diretamente do algoritmo mcq e o mesmo acontece com os algoritmos mcr e mcs.

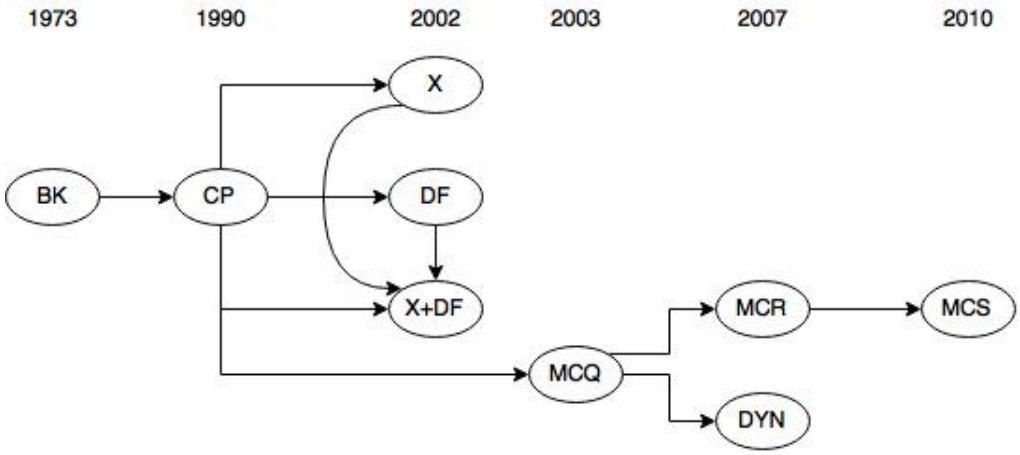

Figura 8: Relação entre os algoritmos e ordem de publicação

Podemos observar que para n divisível por 3, os algoritmos χ, $\chi + \mathsf{df}$, mcr e mcs a fórmula que calcula os Passos de *Branching* é a mesma, ou seja, os 4 algoritmos apresentam exatamente o mesmo desempenho. Como nesses 4 algoritmos n é multiplicado por $2/3$ e no caso de dyn e mcq n é multiplicado por 2, χ, $\chi + \mathsf{df}$, mcr e mcs são os melhores algoritmos para grafos cujo nível de n é divisível por 3.

Quando $n + 1$ é divisível por 3, em χ e $\chi + \mathsf{df}$, n é multiplicado por $2/3$. Já em mcr e mcs, n é multiplicado por $3/4$, e por fim, em dyn e mcq por 2. Constatamos então que χ e $\chi + \mathsf{df}$ são os mais eficientes para $n + 1$ divisível por 3. O mesmo ocorre nos casos em que $n + 2$ é divisível por 3. Sendo assim, de forma geral, χ e $\chi + \mathsf{df}$ são os algoritmos mais eficientes quanto ao indicador de desempenho Passos de *Branching* para grafos de Moon-Moser.

4.3.2 Tempo de CPU

Vemos nas tabelas 85 e 87 os tempos médios de execução, na servidora *Latrappe*, e intercalados com esses, nas Tabelas 86 e 88, os resultados do experimento de validação realizado na servidora *Achel*. Os resultados estão ordenados de forma decrescente de tempo com desvio padrão.

n=**200**	t	σ
$\chi + \mathsf{df}$	19,637	0,259%
χ	17,422	0,129%
dyn	1,539	0,003%
mcq	1,443	0,005%
mcr	5,579	0,063%
mcs	5,427	0,109%

Tabela 85: Tempo, em segundos, demandado pelos algoritmos para encontrar a clique máxima com n=200 na servidora *Latrappe*.

n=**200**	t	σ
$\chi + \mathsf{df}$	15,06	0,08%
χ	12,99	0,15%
dyn	1,15	0,01%
mcq	1,07	0,00%
mcr	4,15	0,03%
mcs	3,92	0,01%

Tabela 86: Tempo, em segundos, demandado pelos algoritmos para encontrar a clique máxima com n=200 na servidora *Achel*.

n=**1350**	t	σ
$\chi + \mathsf{df}$	6.914,200	18,363%
χ	6.257,673	36,696%
dyn	510,642	10,062%
mcq	459,129	7,339%
mcr	117,941	1,383%
mcs	119,898	1,142%

Tabela 87: Tempo, em segundos, demandado pelos algoritmos para encontrar a clique máxima com n=1350 na servidora *Latrappe*.

n=**1350**	t	σ
$\chi + \mathsf{df}$	5.098,39	44,44%
χ	4.820,78	22,57%
dyn	372,41	6,34%
mcq	328,96	3,66%
mcr	87,61	1,08%
mcs	85,92	1,36%

Tabela 88: Tempo, em segundos, demandado pelos algoritmos para encontrar a clique máxima com n=1350 na servidora *Achel*.

Percebemos que a posição de cada algoritmo permanece a mesma no experimento de validação. Os resultados evidenciam que para os grafos de Moon-Moser, os algoritmos mcs e mcr são os mais eficientes quando ao indicador de desempenho Tempo de CPU. Por outro lado, os algoritmos χ e $\chi + \mathsf{df}$ ocupam as piores posições.

4.3.3 Passos de *Branching* em conjunto com Tempo de CPU

Temos nas Tabelas 89 e 90 o resultado da relação entre os indicadores Tempo de CPU e Passos de *Branching*. Como ambos os indicadores já foram validados nas seções anteriores, usamos apenas os resultados da servidora *Latrappe*.

n=**200**	T	t	**t/T**
χ	135	17,422	0,1291
$\chi + \mathsf{df}$	135	19,637	0,1455
dyn	399	1,539	0,0039
mcq	399	1,443	0,0036
mcr	265	5,579	0,0211
mcs	265	5,427	0,0205

Tabela 89: Relação entre os indicadores Tempo de CPU e Passos de *Branching* para n=200.

n=**1350**	T	t	**t/T**
χ	901	6257,673	6,9453
$\chi + \mathsf{df}$	901	6914,200	7,6739
dyn	2699	510,642	0,1892
mcq	2699	459,129	0,1701
mcr	901	117,941	0,1309
mcs	901	119,898	0,1331

Tabela 90: Relação entre os indicadores Tempo de CPU e Passos de *Branching* para n=1350.

Mais uma vez fica evidente que, embora os algoritmos χ e $\chi + \mathsf{df}$ geram a menor quantidade de Passos de *Branching*, o fazem pagando um valor alto de Tempo de CPU. O algoritmo mcs prevalece como o algoritmo com o melhor custo/benefício entre ambos os

indicadores. Como os algoritmos que competem com mcq e os algoritmos que se originaram dele, mcr, dyn e mcs, não é surpresa que os últimos apresentem melhor resultado, dado o alto custo de tempo de CPU que os algoritmos χ e $\chi + \mathsf{df}$ demandam para os cálculos necessários na fase de *bounding*.

5 CONCLUSÃO

Nossa proposta para este trabalho era apresentar os principais conceitos de Análise Experimental constantes no livro "A Guide To Experimental Algorithmics", da autora Catherine McGeogh, mas não somente apresentá-los e sim exemplificá-los através da análise comparativa de um grupo de algoritmos que resolvem um mesmo problema, o problema da clique máxima.

É bastante fácil perder-se no mar de dados que analisar algoritmos experimentalmente pode gerar. Quanto mais se analisa, mais ângulos de testes e experimentos se abrem, e é tentador seguir esses caminhos em busca de dados novos, corroboração dos dados obtidos e explicações acerca de tais dados. Muitos fatores envolvem a execução da implementação de um algoritmo, tantos que é praticamente impossível prever de forma universal o comportamento de um algoritmo quando colocado para "trabalhar no mundo real". Buscamos aqui conduzir o processo experimental de forma que seja possível reproduzi-lo e compará-lo.

Buscamos estruturar o experimento e apresentar seus resultados de forma a não sobrecarregar o leitor com dados, focando sempre na pergunta a ser respondida por cada experimento.

Esperamos que nossa experiência aqui relatada possa servir de exemplo para quem deseje colocar em prática as ferramentas que a Análise Experimental fornece. Por esse motivo expomos não apenas os pontos positivos e frutíferos do processo como todo, mas aqueles que não previámos e que acabaram dificultando o processo além do que imaginávamos, como a falta de controle absoluto sobre o *hardware* utilizado atrelado a testes de longo tempo de processamento.

Nenhum dos experimentos foi idêntico. Cada um conta com suas peculiaridades. Nosso intuito era não só de mostrar técnicas diferentes, mas também demonstrar que vários caminhos podem ser utilizados para obter resultados.

REFERÊNCIAS

BARTZ-BEIELSTEIN, T. et al. *Experimental methods for the analysis of optimization algorithms*. [S.l.]: Springer, 2010. 1-12 p.

BERMAN, P.; PELC, A. Distributed probabilistic fault diagnosis for multiprocessor systems. *20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-20)*, IEEE Computer Society Press, Newcastle upon Tyne, UK, p. 340–6, 1990. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=89383isnumber=2913>.

BOLLOBáS, B. *Random Graphs*. Second. Cambridge University Press, 2001. Cambridge Books Online. ISBN 9780511814068. Disponível em: <http://dx.doi.org/10.1017-/CBO9780511814068>.

BROCKINGTON, M.; CULBERSON, J. C. Camouflaging independent sets in quasi-random graphs. *Cliques Coloring and Satisfiability Second DIMACS Implementation Challenge*, American Mathematical Society, v. 26, p. 75–88, 1996.

BRYANT, R.; O'HALLARON, D. Computer systems: a programmer's perspective. In: _____. Prentice Hall, 2003. cap. 9. ISBN 9780130340740. Disponível em: <https://books.google.com.br/books?id=4n4ZAQAAIAAJ>.

CARRAGHAN, R.; PARDALOS, P. M. An exact algorithm for the maximum clique problem. *Operations Research Letters*, v. 9, n. 6, 1990. Disponível em: <http://dx.doi.org/DOI:10.1016/0167-6377(90)90057-C>.

CORRÁDI, K.; SZABÓ, S. A combinatorial approach for keller's conjecture. *Periodica Mathematica Hungarica*, Akadémiai Kiadó, co-published with Springer Science+Business Media B.V., Formerly Kluwer Academic Publishers B.V., v. 21, n. 2, p. 95–100, 1990.

FAHLE, T. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002. p. 47–86. Disponível em: <http://dx.doi.org/10.1007/3-540-45749-6_44>.

GENDREAU, M.; SORIANO, P.; SALVAIL, L. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, Baltzer Science Publishers, Baarn/Kluwer Academic Publishers, v. 41, n. 4, p. 385–403, 1993. ISSN 0254-5330. Disponível em: <http://dx.doi.org/10.1007/BF02023002>.

HASSELBERG, J.; PARDALOS, P.; VAIRAKTARAKIS, G. Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization*, Kluwer Academic Publishers, v. 3, n. 4, p. 463–482, 1993. ISSN 0925-5001. Disponível em: <http://dx.doi.org/10.1007/BF01096415>.

JOHNSON, D. A theoretician's guide to the experimental analysis of algorithms. 1996.

JOHNSON, D.; TRICK, M. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*. American Mathematical Society, 1996. (Center for Discrete Mathematics and Theoretical Computer Science New Brunswick, NJ: DIMACS series in discrete mathematics and theoretical computer science). ISBN 9780821870723. Disponível em: <https://books.google.com.br/books?id=-ysHoxUwdMUC>.

JOHNSON, D. J.; TRICK, M. A. (Ed.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. Boston, MA, USA: American Mathematical Society, 1996. ISBN 0821866095.

JOHNSON, D. S. et al. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 39, n. 3, p. 378–406, maio 1991.

KONC, J.; JANEŽIČ, D. An improved branch and bound algorithm for the maximum clique problem. *MATCH Communications in Mathematical and in Computer Chemistry*, June 2007. Disponível em: <http://www.sicmm.org/konc/articles/match2007.pdf>.

LAGARIAS, J. C.; SHOR, P. W. Keller's cube-tiling conjecture is false in high dimensions. *BAMS: Bulletin of the American Mathematical Society*, v. 27, n. 2, p. 279–283, 1992.

MCGEOCH, C. C. *A guide to experimental algorithmics*. [S.l.]: Cambridge University Press, 2012.

MOON, J.; MOSER, L. On cliques in graphs. *Israel Journal of Mathematics*, v. 3, n. 1, p. 23–28, March 1965. Disponível em: <http://dx.doi.org/10.1007/BF02760024>.

SANCHIS, L. A. Generating test cases for the vertex cover problem. In: *Proceedings of the DIMACS Workshop on Computational Support for Discrete mathematics*. [S.l.: s.n.], 1992. Technical Report received as Postscript.

SEDGEWICK, R.; SCHIDLOWSKY, M. Algorithms in java, third edition, parts 1-4: Fundamentals, data structures, sorting, searching. In: _____. 3rd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998. cap. 2. ISBN 0201361205.

STEIN, W. et al. *Sage Mathematics Software*. [S.l.]. Disponível em: <http://www-.sagemath.org>.

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, January 2007. ISSN 0925-5001. Disponível em: <http://dx.doi.org/10.1007/s10898-006-9039-7>.

TOMITA, E.; SEKI, T. *An Efficient Branch-and-Bound Algorithm for Finding a Maximum Clique*. Springer Berlin/Heidelberg, 2003. 278-289 p. Disponível em: <http://www.springerlink.com/content/7jbjyglyqc8ca5n9>.

TOMITA, e. et al. A simple and faster branch-and-bound algorithm for finding a maximum clique. In: RAHMAN, M.; FUJITA, S. (Ed.). *WALCOM: Algorithms and Computation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. v. 5942, cap. 18, p. 191–203. ISBN 978-3-642-11439-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-11440-3_18>.

ZüGE, A. *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>.