Universidade Federal do Paraná

Paulo Guilherme Inça
Rodrigo Camargo Reis

Estimativa on-line de tempo de execução para algoritmos de branch-and-bound

Curitiba PR
2017

Paulo Guilherme Inça
Rodrigo Camargo Reis

Estimativa on-line de tempo de execução para algoritmos de branch-and-bound

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Renato Carmo.

Curitiba PR
2017

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Graduação II

Título do Trabalho: Estimativa on-line de tempo de execução para algoritmos de branch-and-bound

Autor(es):
GRR20133846 Nome: Paulo Guilherme Inça

GRR20134164 Nome: Rodrigo Camargo Reis

Apresentação: Data: 10/7/2017 Hora: 13h30 Local: Auditório do Departamento de Informática

Orientador: Renato Carmo

Membro da banca 1: Fabiano Silva

Membro da banca 2: Ricardo Tavares Oliveira

(nome) (assinatura)

AVALIAÇÃO – Produto escrito		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Conteúdo	(00-40)				
Referência Bibliográfica	(00-10)				
Formato	(00-05)				

AVALIAÇÃO – Apresentação Oral		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Domínio do Assunto	(00-15)				
Desenvolvimento do Assunto	(00-05)				
Técnica de Apresentação	(00-03)				
Uso do Tempo	(00-02)				

AVALIAÇÃO – Desenvolvimento		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Nota do Orientador	(00-20)		**********	**********	100
NOTA FINAL		**********	**********	**********	

Pesos indicados são uma sugestão. Conforme ofício PROGRAD 200/2013 o orientador é responsável por entregar na secretaria do curso DVD identificado com o nome do aluno, ano/semestre da defesa e a inscrição "TG II", contendo o arquivo do texto do trabalho em pdf juntamente com outros arquivos relevantes no prazo de 7 dias após a defesa, incluindo as correções solicitadas pela banca, se houver . Esta ata também deve ser entregue logo após a defesa na secretaria do curso para arquivamento. A falta da ata ou entrega do DVD impedem a inclusão do aluno na lista de formandos. Alternativamente ao DVD pode ser entregue declaração assinada pelo aluno justificando a recusa em entregar o texto em pdf.

Dedicamos este trabalho à todos os pesquisadores da área da computação que contribuem ativamente para o avanço do conhecimento.

Agradecimentos

Agradecemos os nossos professores que tiveram uma importância fundamental em nossa vida acadêmica e se mostraram sempre prontos para nos ajudar.

Agradecemos especialmente o Professor Renato Carmo, pelas horas incontáveis de ensinamentos e por se fazer tão presente no desenvolvimento deste trabalho, nos guiando sempre pelo melhor caminho e fazendo jus ao termo orientador.

Agradecemos também os nossos amigos que sempre estiveram próximos em todos esses anos de faculdade, contribuindo direta ou indiretamente em nossa vida acadêmica.

O autor Paulo Guilherme Inça agradece a:

Agradeço a todos que estiveram ao meu lado durante os últimos anos. Agradeço o meu pai, Paulo Jośe Inça, por me dar todo o suporte necessário para que eu pudesse focar nos meus estudos e por sempre acreditar em mim. Agradeço também a minha mãe, Vilce Maria Inça, pelo carinho e apoio emocional que só uma mãe sabe proporcionar. Finalmente agradeço a minha irmã, Fernanda Carolina Inça, que considero minha segunda mãe, pelos ótimos conselhos.

O autor Rodrigo Camargo dos Reis agradece a:

Agradeço primeiramente a Deus por me possibilitar chegar até aqui. Agradeço a minha família pelo apoio constante, em especial minha mãe, Edna Gomes Camargo dos Reis, por sempre acreditar em mim, pelas palavras de afeto e pelos gestos que mesmo simples, fazem toda a diferença. Um agradecimento especial ao meu pai, Jason José dos Reis, pelo incentivo, por me proporcionar infinitas risadas e por ser o meu exemplo de pessoa. Minhas irmãs Letícia e Luana, pelo carinho e companheirismo. Minha amiga e companheira Raísa Alves Moreira, por me ajudar em todos os momentos, pelo seu dom de tornar as coisas mais leves e divertidas, me fazendo enxergar o lado bom de tudo. Meus avós, pelas palavras de alegria e sabedoria, e pelo jeito simples de me fazer ver a vida. E meu padrinho Desso, pelos conselhos, pelo exemplo de pessoa que é, e por sempre fazer com que eu acreditasse no meu potencial.

Resumo

Algoritmos de *backtracking* e *branch-and-bound* apresentam uma grande amplitude no tempo de execução, podendo durar desde segundos até séculos para concluir a execução. É difícil afirmar precisamente quanto tempo será gasto, porém é possível estabelecer uma previsão através de estimadores, que analisam a árvore gerada pela execução. Neste trabalho, apresentamos um método originalmente concebido por Knuth. Apresentamos também outros três estimadores mais sofisticados, que o aperfeiçoam. Discutimos as características destes estimadores, fazemos sua análise e apresentamos uma avaliação comparativa experimental entre eles.

Palavras-chave: estimador, backtracking, branch-and-bound.

Abstract

Backtracking and *branch-and-bound* algorithms have a large amplitude of execution time, and can last from seconds to centuries to finish. It is difficult to say precisely how much time will be spent, but it is possible to establish a prediction through estimators, which analyze the tree generated by their execution. In this work, we present a method originally conceived by Knuth. We also present three other more sophisticated estimators, that seek to correct some problems found in the original method. Their characteristics are discussed, as well as an analysis of the operation, and a comparison between the methods.

Keywords: estimator, backtracking, branch-and-bound.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

WBE	Weighted Backtrack Estimator
MIP	Mixed Integer Programming

Lista de Símbolos

ϕ	phi, conjunto de folhas da árvore
ψ	psi, coleção dos tamanhos dos ramos da árvore
Θ	theta, limitante do crescimento da função
γ	gamma, razão entre a largura de um nível e a do nível anterior

Introdução

Sempre tentamos fazer com que os algoritmos que construímos sejam “rápidos”, mas é dificil dizer precisamente o que de fato seria isso. Talvez um segundo, um minuto, uma hora, um dia. Estamos sempre tentando otimizar o funcionamento deles, mas muitas vezes torna-se difícil julgar se estamos de fato sendo ótimos no desenvolvimento ou não. Além de conhecer a rapidez de execução de um algoritmo, pode ser interessante prever quanto tempo isso deve demorar. Um algoritmo de *backtracking*, por exemplo, pode levar menos de um segundo para executar em certos casos, mas em outra situação pode demorar mais de um século. Esses algoritmos são imprevisíveis e um dos desafios é saber quanto tempo será necessário para resolver uma instância do problema. Se estamos lidando com uma situação onde o algoritmo já está sendo executado há uma semana, é aceitável esperar mais um dia para a sua conclusão. Em contrapartida, se a previsão para essa mesma execução concluir que será necessário esperar mais outra semana, pode ser pertinente avaliar a situação e verificar se é viável prosseguir com a execução. Diante dessa incerteza, o nosso interesse principal é poder estimar o tempo de execução de algoritmos de *backtracking* e de *branch-and-bound*. O intuito é utilizar estimadores para prever o tempo de execução e mostrar uma estimativa de qual será a duração dela.

No Capítulo 1 analisamos os algoritmos de *backtracking* e *branch-and-bound* através de exemplos, e apresentamos uma implementação do *branch-and-bound*. O Capítulo 1 também apresenta uma análise da estimativa do tempo de execução, conceitos mais específicos e fundamentos estatísticos. Do Capítulo 2 ao Capítulo 5 é feita uma análise de quatro estimadores, mostrando o funcionamento e a implementação dos mesmos. O Capítulo 6 tem foco direcionado aos resultados dos estimadores, e comparações entre eles.

Capítulo 1

Algoritmos de Enumeração

Neste capítulo, tratamos ideias gerais a respeito de dois tipos de algoritmos de enumeração, *backtracking* e *branch-and-bound*, e a estrutura de árvore gerada a partir de suas execuções. No presente trabalho, consideramos como algoritmos de enumeração esses dois tipos de algoritmos em específico, que são utilizados nos próximos capítulos pelos estimadores apresentados. Os algoritmos de enumeração se assemelham à uma busca em árvore que consiste em percorrer ramos em busca da solução do problema. Esta árvore não é construída de fato, ela é uma abstração da execução do algoritmo, e representa as "escolhas" implementadas no algoritmo para encontrar a solução do problema.

É importante evidenciar que apenas os problemas de otimização são tratados neste trabalho. Existe uma importante diferença deles em relação aos problemas de decisão: nos problemas de decisão o algoritmo para ao encontrar a solução, enquanto que nos problemas de otimização o algoritmo enumera todas as soluções.

A árvore de execução de um algoritmo de enumeração é uma estrutura formada por vértices, ou nós, e por arestas que os ligam, estabelecendo uma relação entre dois nós. Para este trabalho, é suficiente pensarmos na árvore como um conjunto finito T de um ou mais nós, onde o primeiro deles recebe o nome de raiz e os demais são particionados em novas árvores $T_1, \ldots, T_m$, chamadas de subárvores da raiz. Recursivamente as subárvores seguem a mesma definição apresentada. Cada nó é raiz de alguma subárvore contida na árvore e é pai dos nós que estão imediatamente abaixo na hierarquia. Se visualizarmos de uma outra perspectiva, os nós que estão abaixo da raiz são filhos dela, e são pais dos nós abaixo deles. Seguindo a mesma ideia, dois nós com o mesmo pai são ditos irmãos. A raiz de uma árvore tem nível igual a zero e se um nó tem nível i, seus filhos têm nível $i + 1$. A profundidade de uma árvore é igual ao maior nível dos seus nós. Cada nó apresenta um grau, valor que é dado pelo número de subárvores filhas dele. Um nó com grau zero é chamado de terminal ou folha, e um nó com grau maior que zero é chamado de interno. O grau de uma árvore é dado pelo maior grau de seus nós.

Na Figura 1.1 podemos observar que o nó A é raiz, pai de B e C, e o seu nível é 0. Observamos também que o nível de B é 1 e que ele não tem filhos, logo é uma folha. C também está no nível 1 e tem dois filhos que são folhas, D e E.

A árvore da Figura 1.1 é construída durante a execução de uma instância do problema, na medida em que o algoritmo busca uma solução para a instância. Uma solução é um conjunto S de valores $\{x_1, x_2, \ldots, x_n\}$ que é possível resposta para a instância do problema. Os valores para o conjunto solução vão sendo encontrados na medida em que o algoritmo percorre a árvore, formando soluções parciais. Durante a execução de algoritmos de enumeração, não conhecemos os caminhos da raiz às folhas que terão que ser percorridos para encontrar uma solução. Os

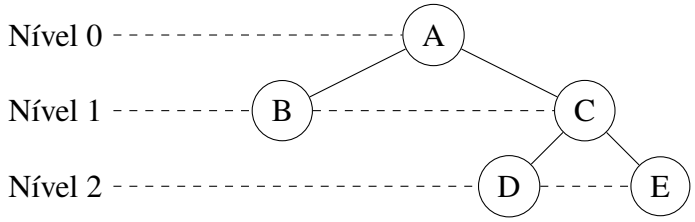

Figura 1.1: Exemplo de Árvore

caminhos só passam a ser conhecidos depois que andamos por eles. No entanto, podemos utilizar parte da árvore já percorrida para construir uma estimativa do todo.

As árvores geradas possuem características diferentes, dependendo do método usado, e é importante observarmos essa diferença. Apesar disso, os estimadores que estimam o tamanho final da árvore, apresentados nos próximos capítulos, são executados em cima da árvore em si, independentemente se ela foi criada a partir de um algoritmo de *backtracking* ou de *branch-and-bound*.

1.1 Backtracking

Nesta seção veremos um exemplo de uso do método de *backtracking*. Uma definição mais precisa pode ser vista em [Kreher e Stinson, 1999, sec. 4.4].

1.1.1 Problema da Clique Máxima

Vamos usar como exemplo o Problema da Clique Máxima. O problema consiste em, dado um grafo G, encontrar a clique de tamanho máximo em G, ou seja, o maior subgrafo completo de G. Uma clique máxima é uma clique de maior tamanho possível em um dado grafo. Vejamos agora o CM_BACK, um algoritmo que resolve o Problema da Clique Máxima e na Figura 1.2, um exemplo de entrada para o algoritmo.

CM_BACK(G, Q, K)

```
1:  C ← Q
2:  if K ≠ ∅ then
3:      v ← pop(K)
4:      C_1 ← CM_BACK(G, Q ∪ {v}, K ∩ Γ_G(v))
5:      if |C_1| > |C| then
6:          C ← C_1
7:      C_2 ← CM_BACK(G, Q, K − {v})
8:      if |C_2| > |C| then
9:          C ← C_2
10: return C
```

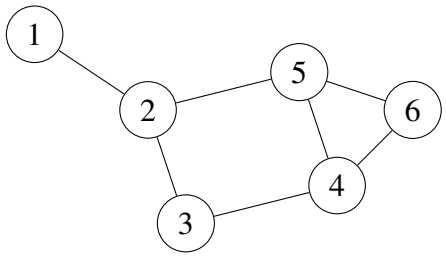

Figura 1.2: Grafo com clique máxima de tamanho 3

Os valores dos vértices da árvore são guardados em duas listas Q e K. A primeira delas guarda os vértices que fazem parte da clique e a segunda guarda os vértices que são candidatos a entrar na clique. O algoritmo mostrado consiste em desempilhar um vértice de K e analisar os seus vizinhos. É interessante observar que os valores são consumidos para impedir que a mesma clique seja encontrada diversas vezes a partir de diferentes vértices do grafo. Vejamos agora na Figura 1.3, a árvore resultante do algoritmo CM_BACK, para o grafo visto na Figura 1.2. Observe o vértice que guarda "[5][2,4]". Neste instante existe a possibilidade de encontrarmos uma clique de tamanho três, visto que já temos um valor guardado e existem outros dois como candidatos, porém ao adicionarmos o 2 notamos que o 4 desaparece, isso ocorre pelo fato dele não ser adjacente ao 5 e ao 2 ao mesmo tempo. Diferente deste caso, temos o vértice "[6][4,5]", quando incluímos o 5 na clique o 4 continua em K, justamente porque ele é adjacente ao 6 e ao 5, por fim, observamos no próximo passo que ele também é incluído na clique. Olhando para o grafo de exemplo, observamos que ele apresenta 6 cliques de tamanho 1, 7 cliques de

tamanho 2 e 1 clique de tamanho 3, somando 14 possíveis cliques. Se olharmos para a árvore gerada, observamos que existem 14 folhas, destacadas em cinza, e elas representam exatamente as cliques encontradas no grafo, a mais escura delas destaca a clique máxima.

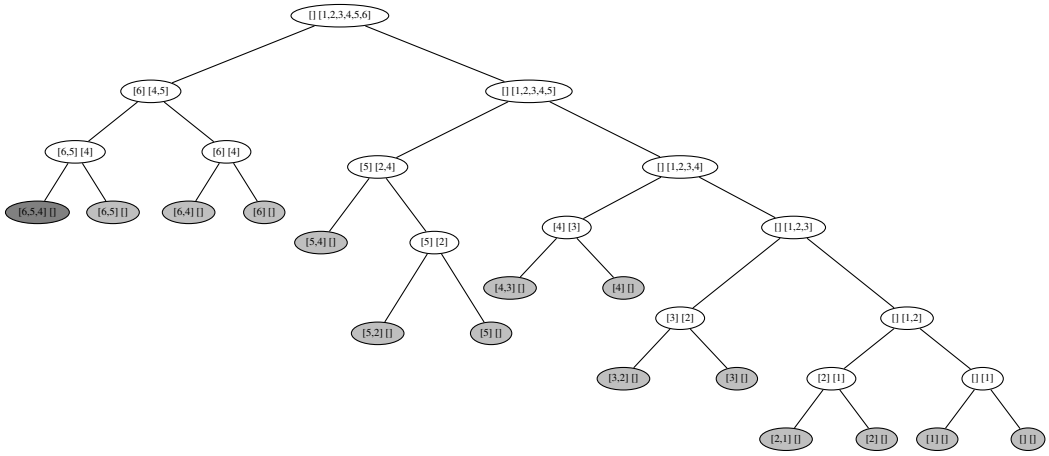

Figura 1.3: Árvore de *backtracking* para o grafo da Figura 1.2

1.2 Branch-and-bound

O método de *branch-and-bound* propõe uma forma mais sofisticada para a busca que vimos anteriormente. O método consiste numa enumeração de soluções candidatas, organizadas em forma de árvore. O algoritmo explora ramos dessa árvore que representam soluções parciais do problema, porém antes de continuar por um ramo, ele é testado contra os limitantes inferiores e superiores para a solução ótima. Caso o ramo não possa produzir uma solução melhor que a encontrada até então, ele é desconsiderado. A estratégia agora é diminuir o número de candidatos para a solução, de forma a otimizar o processo de solução do problema.

1.2.1 Problema da Clique Máxima

Vejamos agora como seria a solução para o Problema da Clique Máxima, visto na seção anterior, se utilizarmos o método de *branch-and-bound* ao invés do *backtracking*. Usaremos o mesmo grafo, representado na Figura 1.4 como entrada do algoritmo CM_BNB, que resolve o Problema da Clique Máxima através do método de *branch-and-bound*.

CM_BNB(G, Q, K)

```
1:  C ← Q
2:  if K ≠ ∅ and |Q| + |K| > |C| then
3:      v ← pop(K)
4:      C_1 ← CM_BNB(G, Q ∪ {v}, K ∩ Γ_G(v))
5:      if |C_1| > |C| then
6:          C ← C_1
7:      C_2 ← CM_BNB(G, Q, K − {v})
8:      if |C_2| > |C| then
9:          C ← C_2
10: return C
```

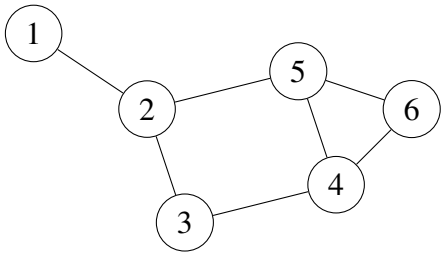

Figura 1.4: Grafo com clique máxima de tamanho 3

Diferente do que vimos anteriormente, precisamos agora definir um limitante para o algoritmo que vai executar essa busca. Podemos por exemplo definir um limitante superior, tal que $|Q| + |K| > |C|$, isso é, um ramo só será candidato se o número de vértices na clique (lista Q), somado ao número de vértices vizinhos à clique (lista K), for maior que a clique máxima encontrada até aquele momento (clique C). Se a soma da quantidade de vértices das listas for menor ou igual ao tamanho da clique máxima, descartamos o ramo, uma vez que ele não poderá resultar numa clique maior da que já foi encontrada. Seguindo essa estratégia, teremos uma árvore para execução do *branch-and-bound* como na Figura 1.5. Para o exemplo, o vértice inicial já faz parte de uma clique máxima, e dessa forma boa parte dos candidatos acabam sendo cortados. É

notável as diferenças entre as árvores geradas pelo *backtracking* e pelo *branch-and-bound*. A partir do próprio tamanho delas podemos deduzir quem precisa percorrer menos ramos para encontrar a solução do problema.

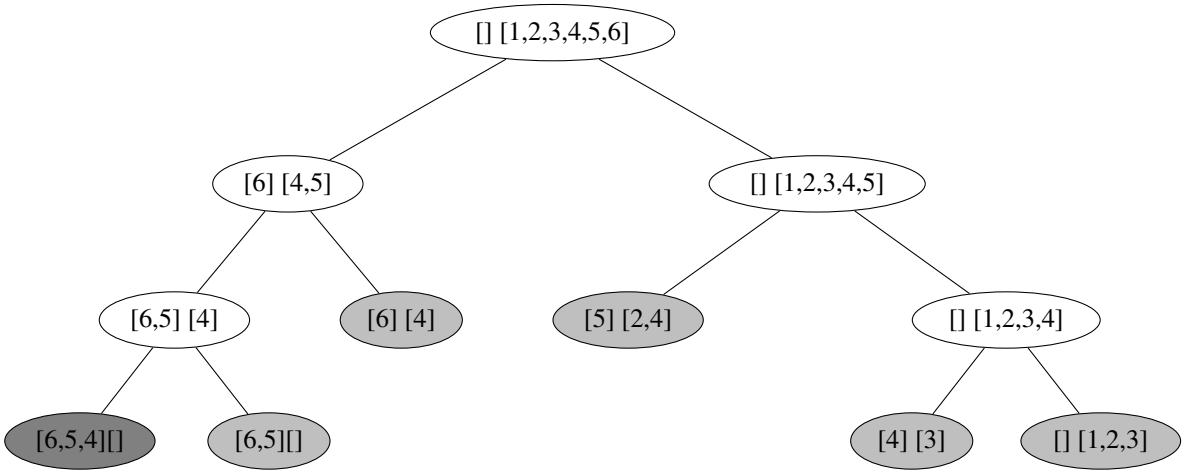

Figura 1.5: Árvore de *branch-and-bound* para o grafo 1.4

1.3 Implementação do branch-and-bound

Para testar na prática os estimadores estudados utilizamos a implementação de algoritmos de enumeração para encontrar a clique máxima em um grafo de [Carmo e Züge, 2011]. Os autores de [Carmo e Züge, 2011] estudam vários algoritmos que encontram a solução exata para o problema da clique máxima utilizando a técnica de *branch-and-bound*. Além disso, eles descrevem um algoritmo genérico de *branch-and-bound* e mostram que cada um dos algoritmos estudados são uma modificação do algoritmo genérico. As implementações foram feitas utilizando a linguagem Python.

Apresentamos aqui a definição de *IMP_CM_BNB* que é o algoritmo genérico de *branch-and-bound* de [Carmo e Züge, 2011]. As definições de *pre-process-instance(G)*, *pre-process-state(G, Q, K, C)*, *upper-bound(G, K, C)*, *remove-vertex(G, K)* e *post-process-instance(G, C)* dependem de cada especialização do *branch-and-bound*. Suas definições completas podem ser vistas em [Carmo e Züge, 2011].

IMP_CM_BNB(G)

```
 1: (C, S) ← pre-process-instance(G)
 2: while S ≠ 0 do
 3:     (Q, K) ← pop(S)
 4:     pre-process-state(G,Q,K,C)
 5:     while K ≠ 0 e |C| < |Q| + upper-bound(G,K,C) do
 6:         v ← remove-vertex(G,K)
 7:         push(S, (Q, K))
 8:         (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
 9:         pre-process-state(G,Q,K,C)
10:         if |C| < |Q| then
11:             C ← Q
                          ▷ Ao final do loop interno o par (Q, K) representa uma folha em T.
12: return post-process-instance(G,C)
```

Seja G um grafo e C a clique máxima atual. O algoritmo *IMP_CM_BNB* substitui a recursão de *CM_BNB* por uma pilha S. Fica evidente nesse exemplo que a árvore não é de fato construída. A pilha de estados é apenas uma representação dessa árvore. Em cada passagem pelo final do loop interno de *IMP_CM_BNB* a lista Q é um clique maximal e a lista K pode estar vazia ou não, caracterizando uma folha. Nesse momento devemos fazer a chamada para um dos estimadores.

Para representar em Python um nó de T, foi empregado um objeto $State$ que contém três listas. As listas Q e K, e a lista map que é utilizada por alguns algoritmos de *branch-and-bound* e contém um número inteiro associado a cada vértice de K.

1.4 Estimativa do Tempo de Execução

O tempo necessário para que um algoritmo de enumeração encontre uma solução ótima está fortemente relacionado à árvore de estados gerada durante a execução do mesmo. Podemos definir como o custo de um nó o tempo necessário para que ele seja processado pelo algoritmo de enumeração. No tempo de processamento levamos em consideração o custo para gerar as novas soluções, ou seja, seus filhos, e o custo de passar da solução atual para uma nova solução gerada. Busca-se que um nó tenha custo polinomial. Sendo assim, o que domina o tempo de execução de um algoritmo de enumeração é o número de nós em sua árvore de busca. Estimar o tamanho da árvore nos leva a ter uma ideia do tempo necessário para que esse algoritmo execute até terminar.

Neste capítulo apresentamos quatro estimadores para o tamanho da árvore de busca. O primeiro estimador foi apresentado em [Knuth, 1975], e será nosso estimador base para comparação com novas soluções surgidas posteriormente e também analisadas nesse capítulo. A principal característica deste estimador é o fato de seu valor esperado ser exatamente o número de nós da árvore de estados cujo tamanho está sendo estimado.

Dois outros estimadores foram apresentados em [Kilby et al., 2006]. O primeiro propõe uma melhoria em relação ao de Knuth. Já o segundo traz uma nova forma de estimativa baseada na ideia de que a parte da árvore já explorada pelo *backtracking* será similar a parte que ainda não foi explorada. O último estimador analisado neste trabalho foi apresentado em [Cornuéjols et al., 2006] e baseia-se em estimar o formato da árvore de execução de algoritmos de *branch-and-bound* analisando apenas a árvore parcial resultante de um curto período da execução do algoritmo.

1.4.1 Definições e notação

Denotamos uma árvore genérica por T, e seu número de nós por $|T|$. Seja n a profundidade de T, ou seja o tamanho do maior ramo de T. Existe um caminho em T da raiz até uma folha, e denotamos por P. Um nó em T é denotado por X. O conjunto de filhos de um nó X é denotado por $C(X)$, e consequentemente o número de filhos de X é $|C(X)|$. Se T for uma árvore binária, então o filho à esquerda de um nó X é dado por $esq(X)$ e o filho à direita de X é dado por $dir(X)$. O pai de um nó é dado por $pai(X)$. A aplicação da função $pai(X)$ i vezes é denotado por $pai^i(X)$.

1.4.2 Estimadores

Em estatística, uma inferência tem o objetivo de fazer generalizações sobre uma população ou um modelo estatístico com base em valores amostrais. Chamamos esta generalização de parâmetro da população e o denotamos por θ. Um estimador $\hat{\theta}$ é uma função que associa elementos de um espaço amostral com elementos do espaço de parâmetros. Um estimador é uma variável aleatória. Portanto, $\hat{\theta}$ tem uma certa distribuição, e para uma particular amostra o valor estimado pode ser mais ou menos de θ. Estimadores possuem três propriedades básicas [Bohm e Zech, 2010]:

Não-tendenciosidade: um estimador $\hat{\theta}$ é dito não-tendencioso com relação um parâmetro θ se seu valor esperado for igual a θ. Ou seja, $\mathbb{E}\left[\hat{\theta}\right] = \theta$. Caso contrário, o estimador é tendencioso e $\mathbb{E}\left[\hat{\theta}\right] - \theta$ é chamado viés.

Consistência: essa propriedade está relacionada à precisão de $\hat{\theta}$ quando o tamanho da amostra k aumenta. São condições suficientes para um estimador ser consistente:

$$\lim_{k\to\infty} \mathbb{E}\left[\hat{\theta}\right] = \theta \text{ e } \lim_{k\to\infty} Var(\hat{\theta}) = 0 \tag{1.1}$$

Eficiência: um estimador é dito eficiente se tiver a menor variância entre todos os possíveis estimadores não-tendenciosos de θ. Ou seja, dados dois estimadores $\hat{\theta_1}$ e $\hat{\theta_2}$, o estimador $\hat{\theta_1}$ é eficiente em relação ao $\hat{\theta_2}$ se $Var(\hat{\theta_1}) \leq Var(\hat{\theta_2})$.

Nesse trabalho, o parâmetro que queremos estimar é o número nós de T. Nosso espaço amostral são todos os caminhos P possíveis encontrados em T. Ou seja, nosso estimador é uma variável aleatória, onde o espaço amostral depende da árvore da qual queremos estimar seu tamanho.

Capítulo 2

O Estimador de Knuth

O estimador de Knuth computa a estimativa de $|T|$ percorrendo um caminho aleatório da raiz até uma folha.

2.1 Tamanho de uma árvore especial

Antes de apresentarmos seu funcionamento em uma árvore qualquer, veremos como o estimador computa o tamanho da árvore em um caso especial [Kreher e Stinson, 1999]. Suponha que c_0, c_1, ..., c_{n-1} são inteiros, onde cada nó na profundidade i tem c_i filhos, para cada $0 \leq i \leq n-1$. Dados $0 \leq i \leq n-1$ e um nó X em T, temos c_i escolhas para filho de X. Nesse caso, percebemos que o número de nós na profundidade i é dado pela equação:

$$S(i) = \begin{cases} 1, & \text{se } i = 0 \\ S(i-1)c_{i-1}, & \text{se } 0 < i \leq n \end{cases}$$

e o número de nós em T é exatamente a soma do valor computado para cada profundidade, e é dado pela equação abaixo:

$$|T| = 1 + c_0 + c_0c_1 + c_0c_1c_2 + \ldots + c_0c_1c_2, \ldots, c_{n-1} = \sum_{i=0}^{n} S(i) \tag{2.1}$$

Em particular se atribuirmos o valor 2 para todo c_i, onde $0 \leq i \leq n-1$, temos uma árvore binária completa. É importante ressaltar que $|T|$ é exato, nesse caso, pelo fato de que todos os nós de uma profundidade qualquer em T têm o mesmo número de filhos.

2.2 Estimativa de uma árvore genérica

Para uma árvore genérica, estabelecemos que todos os nós de uma profundidade qualquer têm o mesmo número de filhos do nó que está no caminho P sendo percorrido. Computamos então uma quantidade $G(X)$, análoga à Equação 2.1, porém alteramos os $c_i s$ para o número de filhos dos nós pertencentes ao caminho. Temos então a seguinte equação, que dado um nó na profundidade i computa a estimativa do número de nós nessa profundidade:

$$G(X) = \begin{cases} 1, & \text{se } X = raiz \\ G(pai(X)) \cdot |C(pai(X))|, & \text{se } X \neq raiz \end{cases} \tag{2.2}$$

logo, se X é raiz, então $G(X) = 1$. Se existem c escolhas possíveis para o nó Y, então $G(Y) = G(X) \cdot c$, para todo Y filho de X, o que é exatamente o mesmo cálculo feito na Equação 2.1.

Analogamente ao caso especial, o número estimado de nós em T é obtido somando o valor computado pela Equação 2.2 em cada nível de T:

$$\hat{N}_T(P) = \sum_{X_i \in P} G(X_i) \tag{2.3}$$

onde $\hat{N}_T(P)$ é a estimativa do tamanho de T. A variável $\hat{N}_T(P)$ pode, de fato, ser maior ou menor que $|T|$. Entretanto, para melhorar a consistência da estimativa, podemos computar a estimativa do parâmetro diversas vezes e calcular a média dos valores obtidos.

O algoritmo $EST\_KNUTH(T)$ retorna $\hat{N}_T$, que depende do caminho P percorrido em T. A variável $\hat{N}_T$ tem o mesmo valor dado pela Equação 2.3, portanto $\hat{N}_T = \hat{N}_T(P)$. Mostramos no Teorema 2 que a esperança de $\hat{N}_T$ é exatamente $|T|$ [Knuth, 1975].

EST_KNUTH(T)

```
1:  global variables
2:      N̂_T, m
3:
4:  procedure PROBE(X)
5:      c ← C(X)                          ▷ c contém os nós filhos de X
6:      if |c| ≠ 0 then
7:          m ← m * |c|
8:          N̂_T ← N̂_T + m
9:          X ← um elemento aleatório de c
10:         Probe(X)
11:
12: X ← raiz(T)
13: N̂_T ← 1
14: m ← 1
15: Probe(X)
16: return N̂_T
```

Exemplo 1 *Na Figura 2.1 temos uma árvore T com a estimativa do número de nós em cada nível dado pela Equação 2.2:*

Ao computarmos a esperança de $\hat{N}_T$ para esse exemplo temos:

$$\mathbb{E}\left[\hat{N}_T\right] = 22 \cdot \frac{1}{12} + 22 \cdot \frac{1}{12} + 10 \cdot \frac{1}{6} + 4 \cdot \frac{1}{3} + 31 \cdot \frac{1}{18} + 67 \cdot \frac{1}{36} + 67 \cdot \frac{1}{36} + 13 \cdot \frac{1}{9} + 13 \cdot \frac{1}{9} = 15$$

Teorema 2 *Para qualquer árvore, se $\hat{N}_T = \hat{N}_T(P)$, então o valor esperado de $\hat{N}_T$ é $|T|$.*

Prova: Seja T uma árvore e X uma folha em T. Seja também $P(X) = (X_0, \ldots, X_i = X)$ o caminho da raiz até X em T. Vamos definir a probabilidade da folha X ser escolhida pelo algoritmo $EST\_KNUTH(T)$. Em particular, a probabilidade de X_1 ser escolhido, quando $pai(X_1)$ já foi escolhido, é $1/|C(pai(X_1))| = 1/|C(X_0)|$. Ainda, $\mathbb{P}(X_2) = 1/|C(pai(X_2))| = 1/|C(X_1)|$, quando $pai(X_2)$ já foi escolhido. Em geral, $\mathbb{P}(X_i) = 1/|C(pai(X_i))| = 1/|C(X_{i-1})|$, dado que

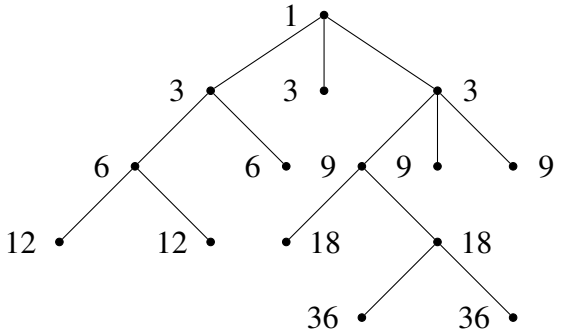

Figura 2.1: Resultado de $EST\_KNUTH(T)$ para o exemplo

$X_0, \ldots, X_{i-1}$ já foram escolhidos. Portanto, a probabilidade de uma folha ser escolhida pelo algoritmo $EST\_KNUTH(T)$ é dada por:

$$\mathbb{P}\left(P(X)\right) = \frac{1}{|C(X_0)|} \cdot \frac{1}{|C(X_1)|} \cdot \ldots \cdot \frac{1}{|C(X_{i-1})|} = \frac{1}{G(X)}$$

Seja $\phi(T)$ o conjunto das folhas em T. A esperança de $\hat{N}_T$ é a média das estimativas do tamanho de T, dado por $\hat{N}_T(P(X))$, para toda folha X, multiplicado pela probabilidade de $P(X)$ ser o caminho escolhido. A esperança de $\hat{N}_T$ é dado pela equação:

$$\mathbb{E}\left[\hat{N}_T\right] = \sum_{X \in \phi(T)} \left(\mathbb{P}\left(P(X)\right) \cdot \hat{N}_T(P(X))\right)$$

substituindo $\mathbb{P}\left(P(X)\right)$ de acordo com a Equação 2.2 e $\hat{N}_T(P(X))$ de acordo com a Equação 2.3, temos:

$$\mathbb{E}\left[\hat{N}_T\right] = \sum_{X \in \phi(T)} \frac{1}{G(X)} \sum_{Y \in P(X)} G(Y)$$

ou seja, para cada folha X pertencente à árvore, o somatório interno nos dá uma estimativa de T, e multiplicamos isso pela probabilidade de X ser escolhida. Invertendo a ordem dos somatórios, temos então:

$$\mathbb{E}\left[\hat{N}_T\right] = \sum_{Y \in T} G(Y) \sum_{X \in \phi(T) \,/\, Y \in P(X)} \frac{1}{G(X)}$$

isto é, para cada nó Y pertencente à árvore, o somatório interno nos dá a soma das probabilidades das folhas descendentes de Y serem escolhidas.

Basta agora mostrar que $\frac{1}{G(Y)}$ é igual ao somatório das probabilidades das folhas descendentes de Y serem escolhidas pelo algoritmo EST_KNUTH. Vamos começar por:

$$\frac{1}{G(Y)} = \sum_{Z \,/\, \text{Z é filho de Y}} \frac{1}{G(Z)} \tag{2.4}$$

iterando a Equação 2.4 temos:

$$\frac{1}{G(Y)} = \sum_{X \,/\, \text{X é uma folha descendente de Y}} \frac{1}{G(X)}$$

isso é equivalente a dizer que:

$$\sum_{X \in \phi(T) \,/\, Y \in P(X)} \frac{1}{G(X)} = \frac{1}{G(Y)}$$

temos então o que queríamos provar:

$$\mathbb{E}\left[\hat{N}_T\right] = \sum_{Y \in T} 1 = |T|$$

□

Mostramos assim que o estimador de Knuth é um estimador não-tendencioso.

2.3 Análise do estimador

Alguns fatos devem ser levados em consideração sobre o estimador apresentado por Knuth. Primeiro, devemos ter a clara noção de que essa é uma maneira bastante simples de estimar o tamanho da árvore de estados, já que estamos estimando o tamanho da árvore inteira baseado em apenas um ramo. Ainda mais, estamos assumindo que a estrutura do resto da árvore é similar ao observado neste ramo, o que não se mostra verdadeiro na maioria dos algoritmos de enumeração [Knuth, 1975]. Um simples exemplo onde isso pode ser observado é se imaginarmos um experimento onde a estimativa de $|T|$ é 1 com probabilidade 0.999, e 1,000,001 com probabilidade 0.001. O valor esperado é 1001, no entanto uma quantidade limitada de estimativas pode nos convencer que o tamanho da árvore é 1.

Segundo, foi observado em [Knuth, 1975] que a maioria das árvores de busca geradas por algoritmos de enumeração tem o maior número de nós concentrados em seus níveis intermediários. Seja K_i o número de nós na profundidade i da árvore, se plotarmos um gráfico de $\log K_i$ em função de i temos uma curva em formato de sino ("bell-shaped"). Um gráfico esquemático do formato da curva pode ser observado na Figura 2.2.

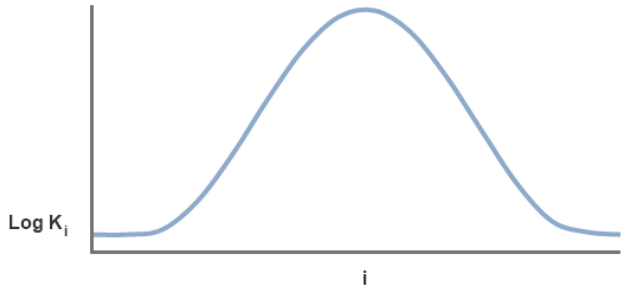

Figura 2.2: Número de nós em T em cada nível

Por outro lado, a estimativa do número de nós em uma certa profundidade, dada pela Equação 2.2, é composta por uma série de estimativas da forma $K'_i = |C(X_0)| \cdot |C(X_1)| \cdot \ldots \cdot |C(X_{i-1})|$. Como, para $0 < k \leq i - 1$, temos que $|C(X_{k-1})| \leq |C(X_k)|$, então $\log K'_i$ em função de i não é uma curva em formato de sino. Se plotarmos um gráfico de $\log K'_i$ em função de i temos uma curva que cresce exponencialmente até cair abruptamente para zero. Um gráfico esquemático do formato da curva pode ser observado na Figura 2.3. No entanto, ao obtermos várias amostras da estimativa, obtemos uma curva em formato de sino como uma média das curvas exponenciais [Knuth, 1975].

Observando a grande variância citada acima, Knuth buscou melhorias para o estimador, que são mencionadas, mas não são estudadas a fundo nesse trabalho. Ao executar a linha 9

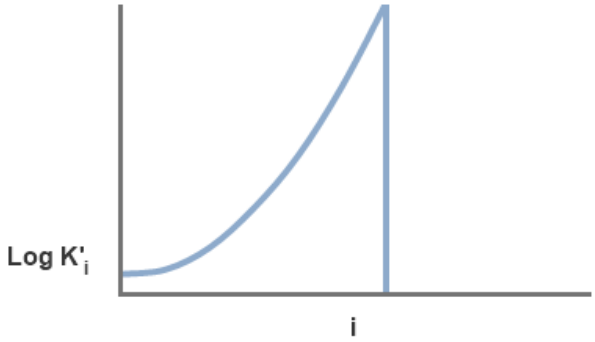

Figura 2.3: Número estimado de nós em T em cada nível

do algoritmo EST_KNUTH, um nó X é escolhido de acordo com uma distribuição uniforme, ou seja, todo elemento do conjunto c tem a mesma probabilidade de ser escolhido. Em outras palavras, o estimador é imparcial ao escolher o caminho percorrido. A primeira melhoria proposta é escolher os caminhos que encontrem "partes mais interessantes" da árvore. Para isso devemos usar outras distribuições para a escolha de um nó em c. Isso também é conhecido como amostragem de importância. O resultado do Teorema 2 se mantém, independente das probabilidades usadas na escolha do próximo nó.

O fato desse estimador ser "off-line", ou seja, ele não pode ser usado durante a execução do algoritmo de enumeração, já que precisa da árvore completa para computar a estimativa de $|T|$, nos fez buscar novas soluções.

2.4 Implementação

Nessa seção apresentamos a implementação do estimador de Knuth. Como vimos no Capítulo 1, o nosso algoritmo de enumeração para o problema da clique máxima gera árvores binárias. O estimador computa uma estimativa de $|T|$ em cada folha de T. Por serem árvores binárias, a única informação que o estimador de Knuth precisa é a profundidade das folhas. Alteramos a definição original do objeto $State$ para armazenar também um inteiro $depth$ que representa a profundidade do nó na árvore.

Apresentamos a definição de IMP_EST_KNUTH, um algoritmo que implementa o estimador de Knuth, em um nível mais alto de abstração. No entanto, a tradução dessa definição para a linguagem Python é trivial.

IMP_EST_KNUTH($depth$)

1: **return** $2 - (1/2^{depth})$ ▷ Equivalente a $(2^{depth+1} - 1) * (1/2^{depth})$

Em relação à IMP_EST_KNUTH, tivemos que fazer uma adaptação para que ele compute a estimativa de $|T|$ de forma online. Ao invés de termos a árvore inteira para visitar as folhas com uma determinada distribuição de probabilidade, o algoritmo visita as folhas em ordem cronológica à medida que o *branch-and-bound* gera essas folhas. Como vimos em sua definição, IMP_EST_KNUTH recebe um inteiro $depth$ que representa a profundidade da folha no caminho no qual o tamanho de T será estimado, e devolve uma estimativa de T.

Capítulo 3

O Estimador Wbe

Como visto na Seção 2.3, um dos maiores problemas na estimativa de Knuth é a sua necessidade de conhecer a árvore completa e a sua imprecisão no caso de árvores muito desbalanceadas. Para abordar estes problemas foi apresentado o Weighted Backtrack Estimator (WBE) [Kilby et al., 2006], que utiliza a parte da árvore já visitada pelo algoritmo de enumeração como um guia para estimar o que ainda não foi visto.

Nos limitamos aqui a estimar o tamanho de árvores de busca binárias. Isso faz com que nossa notação se mantenha mais simples, pois não precisamos nos referir a um nó em específico, mas apenas a sua profundidade. No entanto, a maioria das ideias mostradas aqui são facilmente generalizadas para qualquer árvore. Ainda mais, muitos algoritmos de busca geram árvores binárias para encontrar as soluções [Kilby et al., 2006].

O estimador WBE é inspirado pelo estimador de Knuth. Contudo, percorremos a árvore fazendo uma busca em profundidade "da esquerda para a direita". Assim as folhas são vistas em ordem cronológica e não aleatoriamente. Podemos fazer uma estimativa do tamanho da árvore em qualquer ponto durante a busca. Estimamos $|T|$, utilizando o estimador WBE, da seguinte forma:

$$\hat{O}_T = \frac{\sum_{i \in \psi} \mathbb{P}(i)\,(2^{i+1} - 1)}{\sum_{i \in \psi} \mathbb{P}(i)} \tag{3.1}$$

onde ψ é um multiconjunto, ou seja, um conjunto com repetição de elementos, que contém o tamanho dos ramos visitados até o momento pela busca, isto é, as profundidades das folhas, e $\mathbb{P}(i)$ é igual $\frac{1}{2^i}$, ou seja, a probabilidade de um nó, na profundidade i, ser escolhido em uma árvore binária. Como podemos observar, dada a profundidade de uma folha, a estimativa do tamanho da árvore é a mesma feita por Knuth, porém essa estimativa é ponderada pela probabilidade de encontrarmos uma folha naquela profundidade. É importante ressaltar que se escolhermos um caminho aleatoriamente em T, então $\mathbb{E}\left[\hat{O}_T\right] = |T|$ no final do primeiro ramo. Portanto o estimador WBE também é não-tendencioso.

Exemplo 3 *Estimativa do tamanho da árvore utilizando $\hat{O}_T$.*

Suponha que a árvore da Figura 3.1 é a árvore parcial da execução de um algoritmo de busca e que durante a execução do mesmo decidimos computar seu tamanho estimado no nó marcado com x. Sendo assim, temos o multiconjunto dos tamanhos dos ramos já visitados dado por $\psi = [2, 3, 3, 2, 4]$. Portanto, o tamanho estimado da árvore é:

$$\hat{O}_T = \frac{\frac{1}{4} \cdot 7 + \frac{1}{8} \cdot 15 + \frac{1}{8} \cdot 15 + \frac{1}{4} \cdot 7 + \frac{1}{16} \cdot 31}{\frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{4} + \frac{1}{16}} \approx 11.27$$

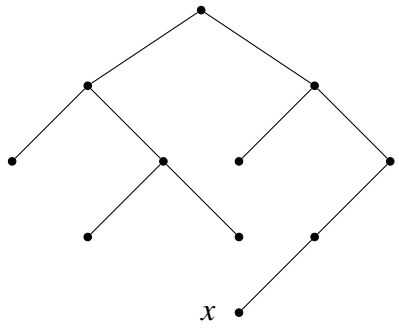

Figura 3.1: Árvore de estados

3.1 Análise do estimador

Como vimos na definição de $\hat{O}_T$ ele estima o tamanho árvore de busca de forma "on-line", isto é, durante a execução do algoritmo de enumeração. Além disso, ele se comporta bem estimando o tamanho de árvores desequilibradas. Nesse tipo de árvore o estimador de Knuth com baixa probabilidade visita os ramos mais profundos, no entanto quando visitados, o tamanho de T será superestimado. Por outro lado, com alta probabilidade o tamanho da árvore será subestimado.

Para exemplificar como os dois estimadores se comportam em uma árvore desequilibrada, vamos supor que temos uma árvore binária de profundidade n onde em cada nó, com exceção do último nível, tem um filho que é uma folha e o outro não.

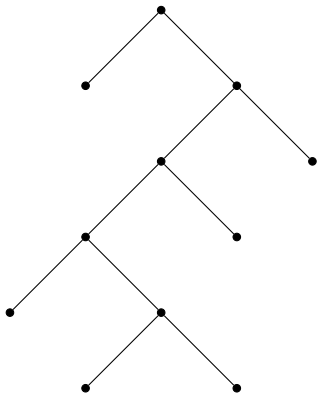

Figura 3.2: Árvore binária desbalanceada com $n = 5$

Nesse caso, como cada nível, menos a raiz, tem exatamente dois nós, o número total de nós na árvore é $2n + 1$. A esperança do estimador de Knuth após percorrer um caminho da raiz até uma folha é dado por:

$$\mathbb{E}\left[\hat{N}_T\right] = \left(\sum_{i=1}^{n} \frac{1}{2^i}(2^{i+1} - 1)\right) + \frac{1}{2^n}(2^{n+1} - 1) \tag{3.2}$$

isto é, temos, para todo $1 \leq i \leq n$, um caminho de tamanho i, e além disso temos mais um caminho de tamanho n, já que na profundidade n existem duas folhas. A Equação 3.2 é equivalente a:

$$\mathbb{E}\left[\hat{N}_T\right] = \left(\sum_{i=1}^{n} 2 - \frac{1}{2^i}\right) + 2 - \frac{1}{2^n}$$

e finalmente,

$$\mathbb{E}\left[\hat{N}_T\right] = 2n - \left(\sum_{i=1}^{n} \frac{1}{2^i}\right) + 2 - \frac{1}{2^n} = 2n - 1 + \frac{1}{2^n} + 2 - \frac{1}{2^n} = 2n + 1$$

e portanto é exatamente o tamanho da árvore. No entanto, a $Var(\hat{N}_T)$ é $\Theta(2^n)$ [Kilby et al., 2006]. Consequentemente, o estimador de Knuth precisa de muitas observações para diminuir essa variância [Kilby et al., 2006]. Por outro lado, o estimador WBE também retorna o valor esperado de $2n + 1$ após explorar um caminho, da seguinte forma:

$$\mathbb{E}\left[\hat{O}_T\right] = \frac{2n - \left(\sum_{i=1}^{n} \frac{1}{2^i}\right) + 2 - \frac{1}{2^n}}{\left(\sum_{i=1}^{n} \frac{1}{2^i}\right) + \frac{1}{2^n}}$$

isto é equivalente a dizer que:

$$\mathbb{E}\left[\hat{O}_T\right] = \frac{2n + 1}{1 - \frac{1}{2^n} + \frac{1}{2^n}} = 2n + 1$$

Portanto, $Var(\hat{O}_T)$ também é $\Theta(2^n)$ como no estimador de Knuth. Porém, após explorar apenas $n + 1$ caminhos a árvore terá sido percorrida completamente e o estimador WBE retorna o tamanho exato da árvore com variância zero. Ou seja, precisamos que o número de amostras no estimador de Knuth seja maior que o número de amostras do estimador WBE para que o erro em relação ao parâmetro se aproxime de zero. Em outras palavras, o estimador WBE é mais consistente que o estimador de Knuth.

3.2 Implementação

Nessa seção apresentamos a implementação do estimador WBE. O estimador WBE nos fornece uma estimativa do tamanho da árvore em cada folha da mesma.

Semelhante ao que vimos em *IMP_EST_KNUTH*, a única informação que o estimador WBE precisa é a profundidade das folhas já encontradas. O algoritmo *IMP_EST_WBE* recebe como parâmetro uma lista de inteiros $depths$ que representa a profundidade das folhas vistas até o momento da chamada do algoritmo pela busca, e retorna uma estimativa de $|T|$ baseado nessa lista. A variável $tree\_size$ é o numerador da Equação 3.1 e $prob$ é o denominador.

IMP_EST_WBE($depths$)

1: **for** $depth \in depths$ **do**
2: $\quad tree\_size \leftarrow tree\_size + 2 - (1/2^{depth})$
3: $\quad prob \leftarrow prob + (1/2^{depth})$
4: **return** $(tree\_size/prob)$

Neste caso, como a lista *depths* é incremental, podemos melhorar o algoritmo *IMP_EST_WBE* para que ele compute a estimativa de $|T|$ apenas para os novos elementos e então some o resultado à um acumulador dos valores antigos. No entanto, para facilitar a apresentação de *IMP_EST_WBE* deixamos que ele compute a estimativa de $|T|$ para todos os valores de *depths*.

Capítulo 4

O Estimador Recursivo

Assim como o estimador WBE, o estimador Recursivo, denotado por $\hat{P}_T$, utiliza como guia para estimar o tamanho da árvore a parte já visitada pela busca. Porém, esse estimador baseia-se em um esquema recursivo. Supondo que estamos percorrendo a árvore da esquerda para a direita e fazendo uma busca em profundidade, sabemos então exatamente o número de nós nessa árvore à esquerda da posição atual da nossa busca. Sendo assim, nossa estimativa da árvore completa é dada somando a parte já conhecida com a parte que ainda será visitada [Kilby et al., 2006].

Vamos denotar por $\mathcal{F}(X)$ uma função que estima o tamanho de uma subárvore com raiz de nó X. Podemos então construir um estimador que utiliza a parte conhecida da árvore e a estimativa dada por $\mathcal{F}(X)$.

EST_REC()

```
1:  global variables
2:      esq[] = "estimado"
3:
4:  procedure BUSCA(X, i)
5:      if folha(X) then
6:          return 1
7:      else
8:          esq[i] ← Busca(esq(X), i + 1)
9:          resultado ← Busca(dir(X), i + 1)
10:         return 1 + esq[i] + resultado
11:
12: procedure ESTIMADOR(X, i)
13:     if folha(X) then
14:         return 1
15:     else
16:         if esq[i] = "estimado" then
17:             tamEsq ← Estimador(esq(X), i + 1)
18:             tamDir ← F(dir(X))
19:         else
20:             tamEsq ← esq[i]
21:             tamDir ← Estimador(dir(X), i + 1)
22:         return 1 + tamEsq + tamDir
```

No algoritmo *EST_REC* o vetor $esq[i]$ guarda o tamanho da subárvore esquerda de um nó na profundidade i do ramo atual sendo explorado pela busca. Podemos obter uma estimativa do tamanho da árvore em qualquer folha chamando $Estimador(raiz, 0)$. Durante a execução de $Estimador(X, i)$, se $esq[i]$ for "$estimado$", isso significa que nossa busca está na subárvore à esquerda do nó na profundidade i, então chamamos $Estimador(esq(X), i + 1)$ recursivamente para contar o tamanho da subárvore à esquerda e usamos a função $\mathcal{F}(X)$ para estimar o tamanho da subárvore à direita. Por outro lado, se $esq[i]$ for conhecido, isso significa que a busca já percorreu toda a subárvore à esquerda do nó na profundidade i, então $esq[i]$ contém o tamanho da subárvore à esquerda e chamamos $Estimador(dir(X), i + 1)$ recursivamente para contar o tamanho da subárvore à direta.

Uma forma de estimar $\mathcal{F}$ é supor que o tamanho da subárvore direita tem mesmo tamanho da subárvore esquerda. Nesse caso, basta alterar a linha 18 do algoritmo *EST_REC* para $tamDir \leftarrow tamEsq$.

4.1 Análise do estimador

Como vimos na definição do estimador Recursivo ele também funciona de forma "on-line" e pode estimar o tamanho da árvore de busca em qualquer folha. Para iniciar uma busca basta fazer uma chamada de $Busca(raiz, 0)$ e quando atingirmos uma folha podemos fazer uma chamada de $Estimador(raiz, 0)$ para obter a estimativa.

Com relação à estimativa computada, fica claro que quando percorrermos a árvore completamente, o tamanho da árvore de busca retornada pelo estimador é exato. Além disso, se ramificarmos aleatoriamente para a direita ou para a esquerda ao percorrermos um caminho na árvore, no final do primeiro ramo $\mathbb{E}\left[\hat{P}_T\right]$ é exatamente $|T|$. [Kilby et al., 2006].

Assim como fizemos com o estimador WBE, vamos analisar como o estimador Recursivo se comporta computando a estimativa do tamanho de árvores pertencentes à uma família de árvores desequilibradas. Como exemplo, vamos supor uma árvore binária, onde a raiz tem um filho que é uma folha na esquerda e tem uma árvore binária completa de profundidade $n - 1$ na direita. Uma representação gráfica pode ser vista na Figura 4.1. Vamos supor que nosso estimador recursivo explora primeiro o filho à esquerda da raiz e depois a subárvore à direita.

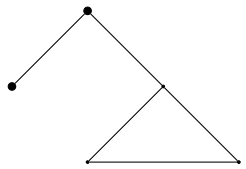

Figura 4.1: Árvore desequilibrada para o estimador recursivo

A esperança do estimador de Knuth novamente é o tamanho exato dessa árvore, e é computado da seguinte forma:

$$\mathbb{E}\left[\hat{N}_T\right] = \frac{1}{2} \cdot 3 + \left(\frac{1}{2^n}(2^{n+1} - 1)\right)2^{n-1} \tag{4.1}$$

Nesse tipo de árvore existe uma folha na profundidade 1, que é o filho à esquerda da raiz, e 2^{n-1} folhas de profundidade n, que são as folhas da árvore binária completa de profundidade $n - 1$ que é subárvore à direita da raiz. Por isso, o estimador de Knuth estima 3 para o tamanho de T com probabilidade de 50% e estima $2^{n+1} - 1$ também com probabilidade de 50%. Isso fica mais claro ao simplificarmos a Equação 4.1 da seguinte forma:

$$\mathbb{E}\left[\hat{N}_T\right] = \frac{1}{2} \cdot 3 + \frac{1}{2}(2^{n+1} - 1) = 2^n + 1$$

Nesse caso o estimador de Knuth tem 50% de chance de subestimar o tamanho da árvore. Em particular, a variância do tamanho da árvore é $\Theta(2^{2n})$ [Kilby et al., 2006]. Como consequência, para obter um bom resultado precisamos de uma grande quantidade de estimativas, já que temos essa chance de 50% escolhermos o caminho menor da esquerda. O estimador recursivo também estima tamanho 3 na primeira folha, porém ele compensa a estimativa já na segunda folha, quando estima tamanho $2^n + 1$, que é o tamanho correto, e mantêm essa estimativa até o final da busca. Isso ocorre porque ao chegar na folha mais a esquerda de uma subárvore de profundidade $n - 1$, o estimador, como definimos, estima o tamanho de uma árvore binária completa de profundidade $n - 1$. Portanto, assim como vimos na análise do estimador WBE, nesse caso o estimador Recursivo precisa de menos estimativas para diminuir o erro do que o estimador de Knuth. Ou seja, o estimador Recursivo é mais consistente que o estimador de Knuth.

4.2 Implementação

Nessa seção apresentamos a implementação do estimador Recursivo. Para este estimador também fizemos algumas adaptações para utilizarmos em *IMP_CM_BNB*. Como vimos na definição de *EST_REC*, *Busca* computa esq percorrendo a árvore de forma recursiva. No entanto, *IMP_CM_BNB* percorre a árvore de forma iterativa. Sabemos que *IMP_CM_BNB* empilha em S apenas o filho à direita de um dado nó. Então, com exceção da raiz, sempre que desempilhamos um nó $state$, sabemos que ele é um filho à direita do nó no nível $state.depth - 1$. Além disso, sabemos que mudamos de ramo, ou seja, terminamos de explorar a subárvore à esquerda no nó nível $state.depth - 1$ e vamos explorar seu filho à direita. Portanto, ao desempilhar $state$, para saber o tamanho da subárvore à esquerda do nó $state.depth - 1$ basta computá-lo utilizando o próprio $left$. Podemos ver um caso particular no Exemplo 4. Por outro lado, se chegarmos em uma folha $state$ que é filho à esquerda de $state.depth - 1$ apenas fazemos $left[state.depth - 1] \leftarrow 1$, já que o tamanho da subárvore à esquerda do nó na profundidade $state.depth - 1$ é apenas uma folha.

IMP_EST_REC($state, left$)

```
1:  if state.K = 0 then
2:      return 1
3:  else
4:      if left[state.depth] == −1 then
5:          state ← generate-left(state)
6:          left_size ← recursive(state, left)
7:          right_size ← left_size
8:      else
9:          left_size ← left[state.depth]
10:         state ← generate-right(state)
11:         right_size ← recursive(state, left)
12: return (1 + left_size + right_size)
```

O algoritmo *IMP_EST_REC* implementa o estimador Recursivo. Ele recebe como parâmetro um nó da árvore, $state$, e $left$, uma lista com os tamanhos das subárvores à esquerda do ramo atual na busca. Devolve uma estimativa de $|T|$.

Exemplo 4 *Exemplo na computação de $left$ em um caso particular.*

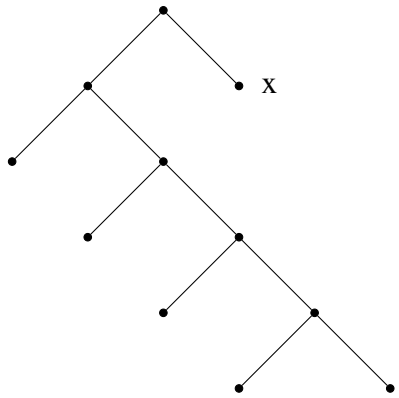

Figura 4.2: Árvore binária

Ao desempilharmos o nó marcado por um x, denotado simplesmente por $state$, sabemos que $state.depth$ é 1, e que já percorremos toda a subárvore à esquerda de seu pai, o nó na profundidade $state.depth - 1$ que é 0. Para computarmos o tamanho da subárvore à esquerda do nó na profundidade $state.depth - 1$ basta fazermos a soma $1 + left[1] + 1 + left[2] + 1 + left[3] + 1 + left[4] + 1$.

A chamada inicial de *IMP_EST_REC* deve ser feita passando como parâmetro a raiz da árvore T. Os métodos *generate-left(state)* e *generate-right(state)* geram os estados que representam os filhos à esquerda e à direita de $state$, respectivamente.

Capítulo 5

O Estimador de Cornuéjols

O estimador de Cornuéjols, denotado por $\hat{Q}_T$, foi proposto para ser implementado como parte de um resolvedor de problemas de programação inteira mista (*mixed integer programming, MIP*), cuja principal característica é apresentar pelo menos uma variável inteira. O intuito do estimador é analisar a árvore parcial resultante de uma curta execução do algoritmo de *branch-and-bound*, e estimar o formato da árvore completa.

Para o seu correto funcionamento, o autor de [Cornuéjols et al., 2006] indica alguns requisitos:

- Funcionar como parte de um solucionador MIP de propósito geral.
- Fornecer estimativas sem controlar ou direcionar o processo de solução.
- Poder produzir uma estimativa com precisão satisfatória, medida por um fator de erro, após um curto período de tempo.
- Poder atualizar a estimativa conforme for sendo executado.
- Não pode tornar o processo de execução mais lento. Os cálculos adicionais necessários para as previsões devem consumir uma quantidade insignificante de tempo, comparado ao algoritmo de enumeração que gera a árvore.

Apesar das indicações apontadas em [Cornuéjols et al., 2006], neste trabalho não consideramos o estimador de Cornuéjols apenas como resolvedor MIP. Nosso intuito é fazer com que ele também resolva outros problemas, independente do algoritmo utilizado para gerar a árvore.

Um dos objetivos desse estimador é produzir uma previsão com "precisão satisfatória". Essa precisão pode ser medida por um fator de erro pelo qual a previsão subestima ou superestima o tamanho real da árvore. Em [Cornuéjols et al., 2006] um fator de erro igual a 5, obtido após 5 segundos de execução é considerado satisfatório. Por exemplo, se estimamos 1 hora para o tempo de execução de um problema, é satisfatório que ele seja resolvido entre 12 minutos e 5 horas, isto é, entre 1 hora dividido pelo fator 5 e 1 hora multiplicado pelo fator 5.

5.1 Sequência-γ

Um elemento importante para o funcionamento do estimador de Cornuéjols é a sequência-γ, que representa a forma e o tamanho da árvore estimada. Definimos γ_i como a razão entre a largura do nível $i+1$ e a largura do nível i. A sequência-γ da árvore é a sequência de γ_i, de todos

os níveis i. Temos então que a sequência-γ é formada por $\gamma_0, \gamma_1, \ldots, \gamma_{d_T}$, onde $\gamma_i = \frac{w_T(i+1)}{w_T(i)}$, para $0 \leq i \leq d_T$. Essa sequência de razões nos permite reproduzir o número de nós de cada nível. Por exemplo, dado a sequência-γ = (2,1), sabemos que o valor 2 é a razão entre a largura do nível 1 e do nível 0, e o valor 1 da sequência-γ é a razão entre a largura do nível 2 e do nível 1. É importante lembrar que o primeiro nível apresenta apenas a raiz e por isso sempre tem largura igual a 1. Concluímos então que o nível 0 tem largura 1 e o nível 1 tem largura 2. Em sequência, notamos que o nível 2 tem largura igual do nível 1, pois é única forma de obter a razão 1. Observe que a sequência-γ não nos dá a árvore exata, e sim um perfil dela, que é justamente o que precisaremos. Na Figura 5.1 podemos ver as possíveis árvores para a sequência-γ = (2,1) usada como exemplo.

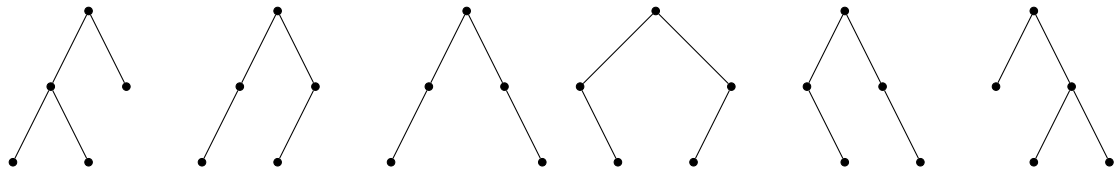

Figura 5.1: Possíveis árvores para a sequência-γ = (2,1)

Pode parecer interessante utilizar essa sequência, obtida de uma árvore parcial, como base para estimar o formato da árvore completa. Apesar disso, testes citados em [Cornuéjols et al., 2006] mostraram que essa estratégia não leva a bons resultados. Ao invés disso, usaremos a árvore parcial para estimar três parâmetros chave da árvore completa, formalmente definidos na próxima seção: A "profundidade" (d_t), que guarda o valor do nível mais profundo da árvore; O "último nível completo" (l_t), que guarda o valor do último nível onde a árvore binária é completa; E a "cintura" (b_t), que guarda o nível mais largo da árvore. Finalmente, esses três parâmetros serão utilizados para modelar sequência-γ da árvore.

Vejamos agora como podemos utilizar um modelo linear para estimar a sequência-γ. Essa estratégia será importante para construirmos a árvore de medição M usada na próxima seção. A árvore M representa um perfil da árvore, um formato. A proposta é modelar esse perfil da árvore completa através dos três parâmetros chave: d_t, l_t e b_t. Dada a sequência-γ de uma árvore T, a largura de um nível particular i é $w_i = \prod_{j=0}^{i-1} \gamma_j$. O tamanho da árvore é $n(T) = 1 + \sum_{i=1}^{d_T} \prod_{j=0}^{i-1} \gamma_j$. Como já citado na seção destinada ao estimador de Knuth, a maioria das árvores de busca geradas por algoritmos de enumeração tem o maior número de nós concentrados em seus níveis intermediários, resultando numa curva em formato de sino ("bell-shaped"), observado na Figura 2.2. A observação apoia o uso do modelo linear apresentado em [Cornuéjols et al., 2006], pela Fórmula 5.1. Do início até l_T, o valor é 2, pois em uma árvore binária completa o próximo nível possui o dobro de vértices se comparado ao anterior. De l_T até b_T, o valor é aproximadamente 1, uma vez que os níveis anterior e posterior à cintura tendem a ter larguras próximas a ela. Finalmente, de b_T até d_T, temos um valor que se aproxima à 0, já que os valores subsequentes à cintura tendem a ser cada vez menor.

$$\gamma_i = \begin{cases} 2, & \text{se } 0 \leq i \leq l_T - 1 \\ 2 - \frac{i - l_T + 1}{b_T - l_T + 1}, & \text{se } l_T - 1 \leq i \leq b_T - 1 \\ 1 - \frac{i - b_T + 1}{d_T - b_T + 1}, & \text{se } b_T \leq i \leq d_T \end{cases} \tag{5.1}$$

5.2 O método

Assim como realizado para os demais estimadores, iremos limitar-nos a estimar o tamanho de árvores de busca binárias, facilitando a notação empregada. Se redefinirmos o último nível completo (l_t), o procedimento de $\hat{Q}_T$ pode acomodar uma árvore com qualquer quantidade de filhos. O fator adicional, como já mencionado, para foco em árvores binárias é que muitos algoritmos de busca geram árvores desse tipo para as suas soluções.

Antes de partirmos para o funcionamento do estimador, vamos observar algumas definições. Em uma árvore T, a variável $w_T(i)$ é a largura do nível i, isto é, o número de nós do nível i. A variável $d_T = max\{i : w_T(i) > 0\}$ é a profundidade da árvore, ou seja, o maior i tal que o nível de i seja maior que zero. O nível $l_T = min\{i : \frac{w_T(i+1)}{w_T(i)} < 2, 0 < i < d_T\}$ é chamado de último nível completo da árvore, acima desse ponto a árvore é binária completa. A cintura da árvore é o nível com a maior largura, $b_T = max\{i : w_T(i) = m\}$ e $m = max\{w_T(i) : 0 \leq i \leq d_T\}$. Quando mais de um nível apresenta a mesma largura máxima, temos $b_T = \lceil \frac{b_1+b_2}{2} \rceil$, onde $b_1 = min\{i : w_T(i) = m\}$, $b_2 = max\{i : w_T(i) = m\}$, ou seja, b_T é o centro do menor intervalo contendo todos os níveis com largura máxima. Seja $n(T) = \sum_{i=0}^{d_T} w_T(i)$ o número de nós em T. A sequência $\{w_T(i) : 0 \leq i \leq d_T\}$ é chamada de perfil da árvore T.

Vejamos agora como $\hat{Q}_T$ estima o tamanho de uma árvore T:

EST_CORNUEJOLS(G, Q, K)

1: $w(i) = 0$, para todo $i = 0, \ldots, j$, onde j é um número grande o suficiente
2: Executa o algoritmo de enumeração
3: Em cada nó da árvore, incrementa o contador correspondente em 1
4: Para a execução quando um evento pré-especificado ocorrer. Por exemplo, atingir um determinado tempo ou encontrar um determinado nó. Seja então t a subárvore resultante de T. Temos $w_t(i) \leftarrow w(i)$, para todo i
5: Encontra l_t, b_t e d_t
6: Constrói a árvore de medição M
7: Se o problema não for resolvido, **retorna** $n(M)$ que corresponde a uma estimativa de $n(T)$, e continua o algoritmo de enumeração, voltando para a Linha 1. Caso contrário, finaliza

Conforme observado, na Linha 4 a execução para de acordo com um evento pré-especificado. Na Linha 5 são encontrados os três parâmetros chave da árvore parcial: o último nível completo, a cintura e a profundidade, e estes serão utilizados para estimar os parâmetros da árvore T. Na Linha 6 é construída uma árvore de medição M. Ela não é uma árvore de fato, é apenas um perfil de uma árvore concebida para replicar o perfil da árvore estimada. A árvore M é construída de acordo com um modelo discutido na seção anterior. O número de nós de M é usado como estimativa para o número total de nós da árvore gerada pelo algoritmo de enumeração. O procedimento é aplicado repetidamente com intuito de produzir estimativas periódicas até o algoritmo de enumeração terminar. Em resumo, temos algo como: a partir da árvore parcial t; encontramos os três parâmetros chave l_t, b_t, d_t; eles são usados para construir a árvore de medição M; a partir dela estimamos l_T, b_T, d_T; com os três parâmetros chave de T, estimamos o tamanho da árvore T.

O processo é dividido em duas fases: A fase 1 termina com a produção da primeira predição. Na fase 2, previsões periódicas vão sendo produzidas. Para que a fase 1 termine, é necessário que duas condições sejam satisfeitas: o tempo de solução deve ser pelo menos 5 segundos e o número de nós na árvore parcial deve ser pelo menos 20 vezes a largura da árvore

parcial ($n(t) \geq 20dt$). A razão entre o número de nós e a largura da árvore é importante para obter uma melhor aproximação dos parâmetros da árvore completa. Segundo [Cornuéjols et al., 2006] o fator 20 foi estabelecido empiricamente e ele pode ser alterado, refletindo no balanceamento entre velocidade e precisão da primeira previsão.

5.3 Análise do estimador

O estimador de Cornuéjols mostrou bons resultados para a maioria dos problemas testados em [Cornuéjols et al., 2006], porém existem imprecisões para alguns casos em específico, que serão discutidas na sequência. O modelo linear proposto, apesar de funcional, apresenta problemas para árvores com profundidade alta e largura baixa. O problema decorre do fato do estimador utilizar apenas três parâmetros, o último nível completo, a cintura e a profundidade. A questão é que o estimador não analisa a largura máxima da árvore, por exemplo. A variável b_T usada para a cintura guarda o nível da largura máxima, e não a largura em si. Se pegarmos duas árvores, uma delas mais estreita e a outra mais larga, e elas apresentarem o mesmo último nível completo, a mesma cintura e a mesma profundidade, o modelo vai estimar o mesmo tamanho para ambas as árvores. Uma possibilidade de melhorar o comportamento de γ observado em árvores mais estreitas é usar um modelo não linear, também proposto em [Cornuéjols et al., 2006], para a sequência-γ. Por exemplo:

$$\hat{\gamma}_i = \lambda(\bar{\gamma}_i - 1)^3 + (1 - \lambda)(\bar{\gamma}_i - 1) + 1, \text{para } 0 \leq i \leq d_T \tag{5.2}$$

para $\lambda \in [0, 1]$ e onde $\bar{\gamma}_i$ é a sequência-γ obtida pelo modelo linear. Essa estratégia tem o modelo linear como um caso especial, quando $\lambda = 0$. Aumentando o valor de λ, podemos obter boas estimativas para sequências-γ de árvores estreitas, mas segundo os autores de [Cornuéjols et al., 2006] esse procedimento não oferece bons resultados para perfis de árvores mais comuns. Ele tem um melhor desempenho em problemas MIP. Em [Cornuéjols et al., 2006] são mostrados testes mais aprofundados que confirmam estes resultados.

Outra questão importante é a preocupação com a estimativa dos três parâmetros da árvore. Testando o comportamento do modelo linear ao realizar mudanças, é possível observar que a cintura se mostra o mais importante dos parâmetros, pois é o que causa maior variação no número de nós da árvore de medição. Podemos pensar então numa outra estratégia para obter a cintura, de modo a melhorar a estimativa da árvore. Da forma em que foi implementado consideramos apenas o nível da maior largura. Nos casos em que existe um empate, pegamos o nível médio. Ao invés de pensarmos apenas no nível com a maior largura, podemos considerar outros níveis quase tão largos quanto a maior largura. Vamos então considerar um valor intermediário, como uma média entre o menor e o maior nível que possuem mais de 50% da largura máxima. Isto é, $b_T = \lceil \frac{b_1 + b_2}{2} \rceil$, onde $b_1 = min\{i : w_T(i) \geq 0,5t\}$, $b_2 = max\{i : w_T(i) \geq 0,5t\}$ e $t = maxw_T(i) : 0 \leq i \leq d_T$. Apesar da mudança não ter sido estudada a fundo, alguns testes realizados em [Cornuéjols et al., 2006], mostraram uma redução de um terço das previsões incorretas.

5.4 Implementação

Nessa seção apresentamos a implementação do estimador de Cornuéjols. O algoritmo *IMP_EST_CORNUEJOLS* implementa o estimador de Cornuéjols.

O algoritmo *IMP_EST_CORNUEJOLS* recebe como parâmetro uma lista de inteiros w que armazena a largura de cada nível em T até o momento da chamada do estimador. O

IMP_EST_CORNUEJOLS(w)

```
1:  d, l, b ← get-parameters(w)
2:  while i ≤ d + 1 do
3:      if i ≤ l − 1 then
4:          γ-sequence[i] ← 2
5:      else if l ≤ i e i ≤ b − 1 then
6:          γ-sequence[i] ← 2 − ((i − l + 1)/(b − l + 1))
7:      else
8:          γ-sequence[i] ← 1 − ((i − b + 1)/(d − b + 1))
9:  while i ≤ d + 1 do
10:     lvl_width ← 1
11:     while j <= i do
12:         lvl_width ← lvl_width ∗ γ-sequence[j]
13:     tree_size ← tree_size + lvl_width
14: return (1 + tree_size)
```

método *get-parameters(w)* computa os parâmetros d, l e b como descrito na Seção 5.1. A lista γ-*sequence* modela o perfil de T. A variável lvl_width é o número de nós em um nível de T. Por sua vez, $tree\_size$ é a soma do número de nós em cada nível de T, mais a raiz.

Capítulo 6

Resultados

Nesse capítulo apresentamos os resultados referente às implementações dos estimadores. Testamos nossa implementação dos estimadores com diversas instâncias. A Tabela 6.1 apresenta as estimativas dos quatro estimadores para o grafo da Figura 6.1. Para este exemplo executamos o *branch-and-bound* sem nenhuma poda na árvore, ou seja, na prática o algoritmo executado foi o *backtracking*. A primeira coluna representa o número de nós que já foram efetivamente encontrados pelo algoritmo de enumeração. Para cada estimador apresentamos uma coluna com número de nós estimados pelo mesmo e uma coluna com a razão entre o número de nós já encontrados e o número de nós estimados, isto é, a porcentagem da árvore já percorrida.

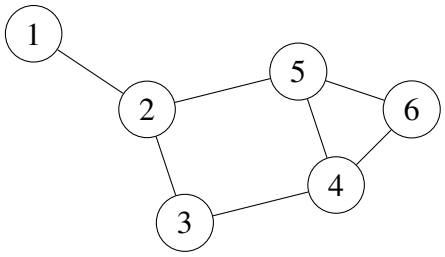

Figura 6.1: Grafo com clique máxima de tamanho 3

Atual	$\hat{N}_T$	Razão	$\hat{O}_T$	Razão	$\hat{P}_T$	Razão	$\hat{Q}_T$	Razão
4	1.875	2.133	15.000	0.267	15.000	0.267	2.833	1.412
5	3.750	1.333	15.000	0.333	15.000	0.333	7.656	0.653
7	5.625	1.244	15.000	0.467	15.000	0.467	7.656	0.914
8	7.500	1.067	15.000	0.533	15.000	0.533	7.656	1.045
11	9.375	1.173	15.000	0.733	19.000	0.579	9.778	1.125
13	11.312	1.149	16.455	0.790	19.000	0.684	12.000	1.083
14	13.250	1.057	17.667	0.792	19.000	0.737	12.000	1.167
17	15.188	1.119	18.692	0.909	21.000	0.810	15.000	1.133
18	17.125	1.051	19.571	0.920	21.000	0.857	15.000	1.200
21	19.094	1.100	21.069	0.997	25.000	0.840	21.111	0.995
22	21.062	1.045	22.467	0.979	25.000	0.880	21.111	1.042
25	23.047	1.085	24.180	1.034	29.000	0.862	23.750	1.053
26	25.031	1.039	25.839	1.006	29.000	0.897	23.750	1.095
28	27.016	1.036	27.444	1.020	29.000	0.966	23.750	1.179
29	29.000	1.000	29.000	1.000	29.000	1.000	23.750	1.221

Tabela 6.1: Estimativa de $|T|$ para o grafo da Figura 6.1

Para facilitar a visualização dos dados apresentamos um gráfico na Figura 6.2 referente aos dados da Tabela 6.1. Nele vemos o número de vezes que os estimadores foram executados, para esse exemplo em particular, pelo valor estimado por cada estimador. A linha pontilhada é o tamanho exato da árvore. Portanto, queremos que a linha representada por algum estimador se aproxime de linha pontilhada com o menor número possível de chamadas. Nesse caso o estimador que melhor cumpre esse objetivo é o estimador Recursivo.

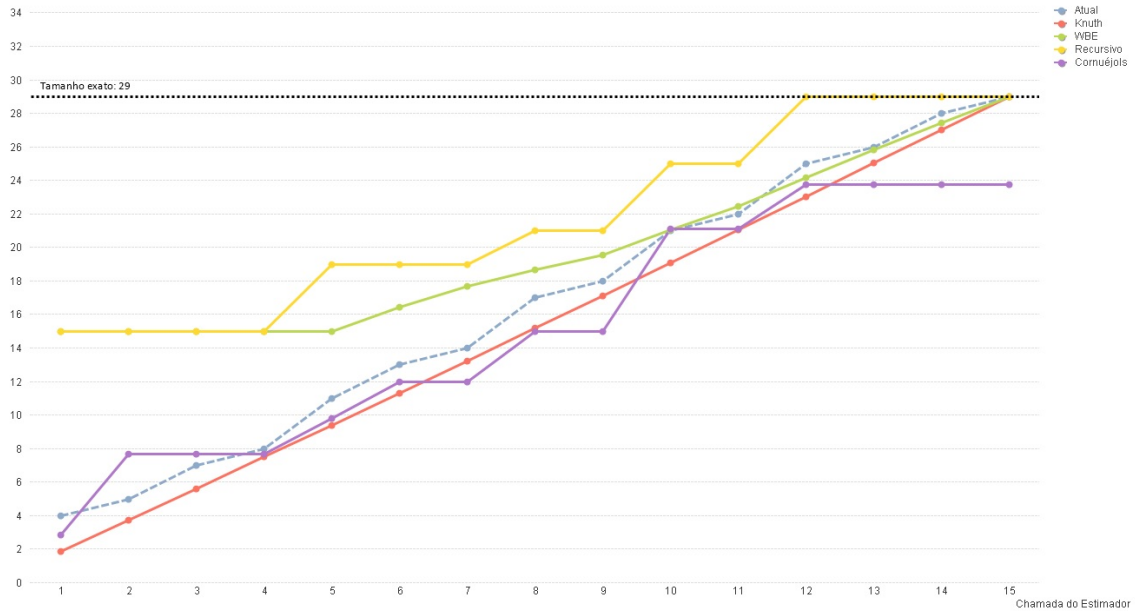

Figura 6.2: Gráfico da estimativa de $|T|$ para o grafo da Figura 6.1

Para observar melhor as características dos estimadores fizemos testes com instâncias maiores. Um grafo aleatório no modelo $\mathcal{G}_{n,p}$ é um grafo G gerando aleatoriamente, onde o parâmetro n é o número de vértices em G e p é a probabilidade de existir uma aresta entre dois vértices, para todo par de vértices em G. Na Tabela 6.2 utilizamos como instância o grafo

$\mathcal{G}_{50,\frac{1}{2}}$. Para diminuir o tamanho do espaço de busca utilizamos como limitante do algoritmo de *branch-and-bound* uma coloração para o grafo. Uma explicação detalhada do algoritmo de coloração utilizado, denotado por DF, pode ser encontrado em [Carmo e Züge, 2011]. Coloração é um limitante bastante apertado, porém encontrar o número mínimo de cores é um problema tão difícil quanto encontrar a clique máxima. Por isso heurísticas são usadas por algoritmos que empregam coloração como um limitante superior. Colocamos na tabela apenas algumas estimativas geradas ao longo da execução para diminuir seu tamanho.

Atual	$\hat{N}_T$	Razão	$\hat{O}_T$	Razão	$\hat{P}_T$	Razão	$\hat{Q}_T$	Razão
5	1.938	2.581	31.000	0.161	15.000	0.333	5.864	0.853
18	15.688	1.147	50.200	0.359	43.000	0.419	19.447	0.926
33	29.514	1.118	60.687	0.544	87.000	0.379	22.137	1.491
47	43.505	1.080	87.824	0.535	87.000	0.540	26.422	1.779
59	57.500	1.026	115.057	0.513	119.000	0.496	60.458	0.976
73	71.281	1.024	99.174	0.736	93.000	0.785	34.770	2.100
86	85.250	1.009	113.667	0.757	109.000	0.789	79.438	1.083
100	99.062	1.009	105.667	0.946	105.000	0.952	134.182	0.745
115	113.020	1.018	115.271	0.998	121.000	0.950	134.182	0.857
128	127.008	1.008	128.008	1.000	131.000	0.977	134.182	0.954
143	141.004	1.014	141.566	1.010	159.000	0.899	139.781	1.023
157	155.002	1.013	155.324	1.011	169.000	0.929	145.081	1.082
171	169.001	1.012	169.104	1.011	173.000	0.988	217.558	0.786
184	183.000	1.005	183.011	1.005	187.000	0.984	323.753	0.568
198	197.000	1.005	197.003	1.005	207.000	0.957	378.927	0.523
212	211.000	1.005	211.000	1.005	213.000	0.995	388.605	0.546
223	223.000	1.000	223.000	1.000	223.000	1.000	432.533	0.516

Tabela 6.2: Estimativa de $|T|$ para o grafo $\mathcal{G}_{50,\frac{1}{2}}$

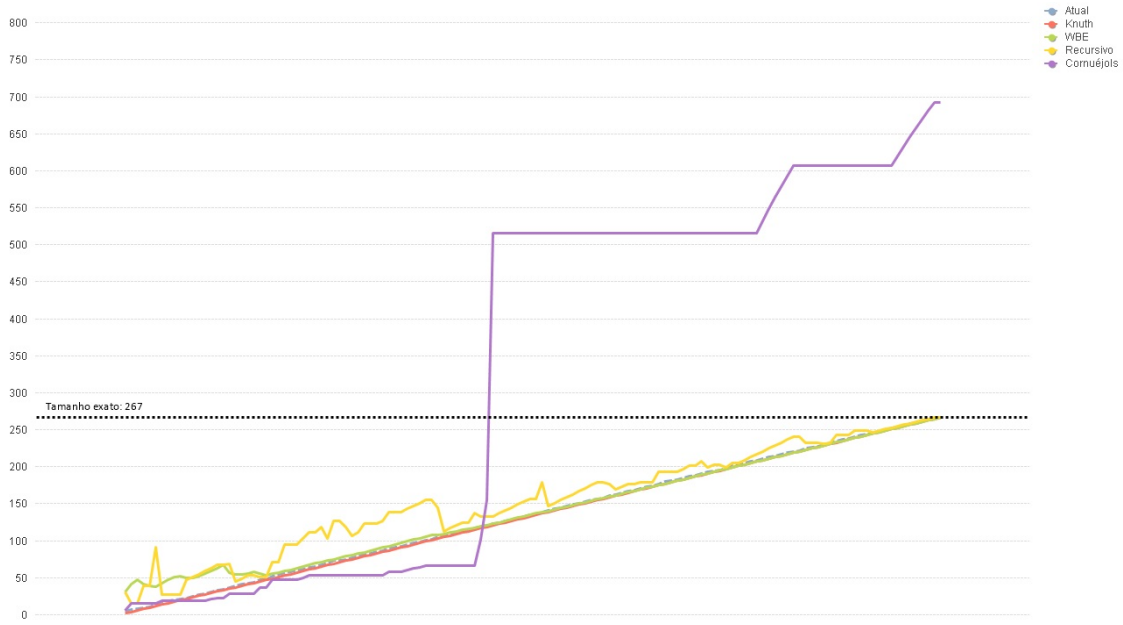

Figura 6.3: Gráfico da estimativa de $|T|$ para o grafo $\mathcal{G}_{50,\frac{1}{2}}$

Na Figura 6.3 vemos um gráfico referente aos dados apresentados na Tabela 6.2. Podemos observar que para a instância $\mathcal{G}_{50,\frac{1}{2}}$ todos os estimadores se comportaram de forma similar, com exceção do estimador de Cornuéjols que superestimou o tamanho da árvore.

Outro tipo de grafo que usamos como instância são os grafos de Moon-Moser. Esse é um grafo multipartido completo formado pelo particionamento de n vértices em $\lceil n/3 \rceil$ subconjuntos. Uma aresta existe entre dois vértices se e somente se eles pertencem a subconjuntos distintos. Em nosso teste o grafo tem 15 vértices. O limitante superior utilizado no *branch-and-bound* é o mesmo descrito no exemplo da Subseção 1.2.1, o tamanho do clique atual somado com seu número de vizinhos deve ser maior que o tamanho do maior clique já encontrado. Na Tabela 6.3 podemos observar as estimativas dos quatro estimadores, apesar de apenas algumas linhas serem apresentadas na tabela.

Atual	$\hat{N}_T$	Razão	$\hat{O}_T$	Razão	$\hat{P}_T$	Razão	$\hat{Q}_T$	Razão
6	1.969	3.048	63.000	0.095	63.000	0.095	6.597	0.909
46	43.812	1.050	233.667	0.197	263.000	0.175	72.442	0.635
89	85.719	1.038	304.778	0.292	383.000	0.232	77.543	1.148
132	127.650	1.034	365.123	0.362	463.000	0.285	112.395	1.174
174	169.604	1.026	427.768	0.407	511.000	0.341	119.906	1.451
216	211.573	1.021	495.769	0.436	639.000	0.338	119.906	1.801
255	253.531	1.006	540.867	0.471	535.000	0.477	126.547	2.015
300	295.461	1.015	548.101	0.547	659.000	0.455	166.107	1.806
341	337.398	1.011	560.870	0.608	583.000	0.585	166.107	2.053
383	379.352	1.010	585.024	0.655	655.000	0.585	166.107	2.306
426	421.323	1.011	622.112	0.685	767.000	0.555	166.107	2.565
468	463.299	1.010	661.210	0.708	775.000	0.604	241.813	1.935
510	505.286	1.009	707.573	0.721	975.000	0.523	241.813	2.109
551	547.261	1.007	740.284	0.744	843.000	0.654	255.095	2.160
594	589.225	1.008	759.907	0.782	959.000	0.619	334.214	1.777
635	631.197	1.006	786.309	0.808	895.000	0.709	334.214	1.900
677	673.174	1.006	814.811	0.831	919.000	0.737	334.214	2.026
720	715.161	1.007	852.160	0.845	1103.000	0.653	334.214	2.154
760	757.148	1.004	889.128	0.855	1095.000	0.694	334.214	2.274
803	799.143	1.005	932.162	0.861	1247.000	0.644	350.563	2.291
844	841.129	1.003	966.143	0.874	1147.000	0.736	350.563	2.408
887	883.084	1.004	964.049	0.920	939.000	0.945	458.151	1.936
927	925.047	1.002	970.541	0.955	971.000	0.955	595.854	1.556
970	967.025	1.003	992.218	0.978	1035.000	0.937	770.386	1.259
1011	1009.012	1.002	1020.976	0.990	1075.000	0.940	770.386	1.312
1029	1029.000	1.000	1029.000	1.000	1029.000	1.000	770.386	1.336

Tabela 6.3: Estimativa de $|T|$ para o grafo Moon-Moser com 15 vértices

Na Figura 6.4 temos o gráfico referente aos dados apresentados na Tabela 6.3. Nesse caso observamos que o estimador de Cornuéjols subestimou o tamanho da árvore. Já os estimadores Recursivo e WBE se comportaram melhor, se aproximando do número real de nós da árvore mais rápido.

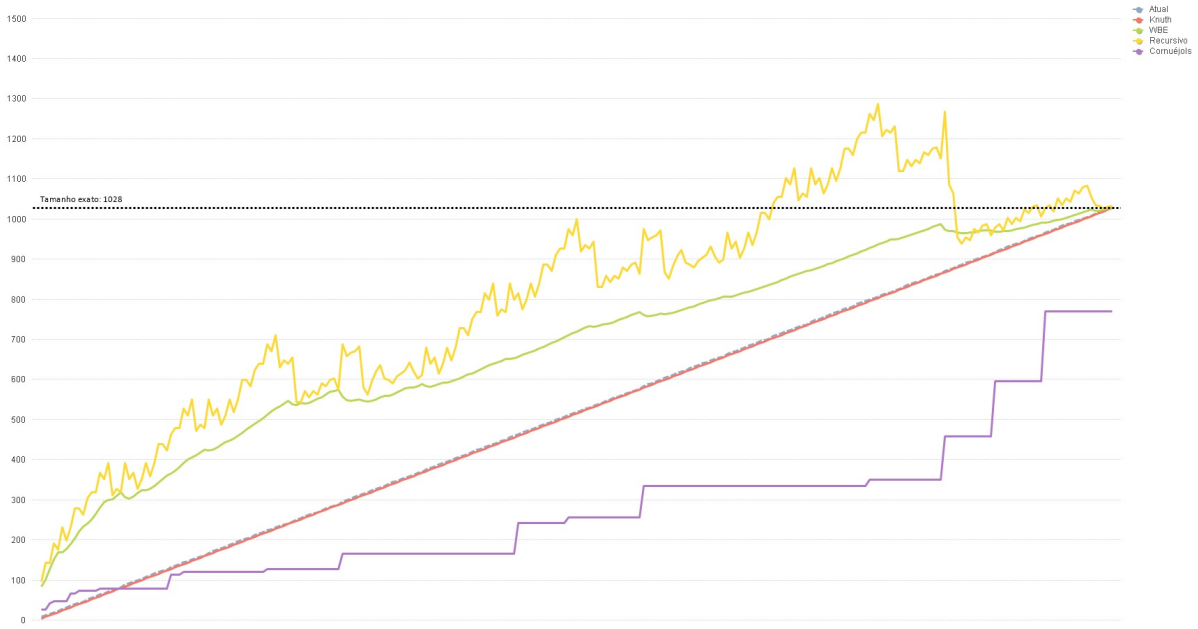

Figura 6.4: Gráfico da estimativa de $|T|$ para o grafo Moon-Moser com 15 vértices

Capítulo 7

Conclusão

Muitos desafios estão envolvidos em saber estimar o tamanho da árvore gerada pelos algoritmos de enumeração. É difícil construir um único estimador que funcione muito bem com todos os tipos de árvores. Cada estimador discutido neste trabalho apresenta vantagens e desvantagens que refletem diretamente na estimativa gerada. O estimador de Knuth, por percorrer ramos de forma aleatória, tem dificuldades em estimar o tamanho de uma árvore desbalanceada. O estimador WBE, proposto por Kilby, apresenta uma melhoria em relação ao de Knuth, fazendo uma busca em profundidade e estimando o número de nós através de uma média ponderada. Sua principal vantagem sobre o método original é o fato dele precisar de menos amostras para computar uma solução exata para a busca. O estimador Recursivo, também proposto por Kilby, é realizado da esquerda para a direita, e parte do ponto que a subárvore esquerda é semelhante à da direita. Sua desvantagem é a necessidade de percorrer novamente a estrutura já visitada pela busca para computar a estimativa. O estimador de Cornuéjols propõe estimar o tamanho da árvore nos primeiros estágios de execução, tomando como base três parâmetros chaves para a árvore parcial. O grande problema do estimador é que os três parâmetros não guardam com precisão as características da árvore, isso faz com que árvores bem diferentes acabem sendo vistas como iguais.

Neste trabalho, implementamos os estimadores estudados e os colocamos lado a lado, comparando os seus resultados em nossa implementação. Para que isso fosse possível, foram necessárias algumas mudanças, como por exemplo, a alteração no funcionamento do método original, proposto por Knuth, que passou a funcionar de modo online. Diferente do que é proposto para os estimadores, a implementação foi feita de modo que eles pudessem funcionar para árvores genéricas, geradas tanto por algoritmos de *backtracking* como de *branch-and-bound*. Também foram feitas alterações no projeto original, para que ele pudesse comportar as necessidades dos novos estimadores.

A construção dos algoritmos realizada no presente trabalho, nos permitiu analisar mais de perto as dificuldades e os méritos na utilização de estimadores que dizem o tamanho da árvore de estados, conhecer o tamanho delas nos permite visualizar uma estimativa do tempo necessário para se obter a resposta em questão.

Referências Bibliográficas

[Bohm e Zech, 2010] Bohm, G. e Zech, G. (2010). *Introduction to statistics and data analysis for physicists*. DESY.

[Carmo e Züge, 2011] Carmo, R. e Züge, A. (2011). Branch and bound algorithms for the maximum clique problem under a unified framework. *Journal of the Brazilian Computer Society*, páginas 1–15.

[Cornuéjols et al., 2006] Cornuéjols, G., Karamanov, M. e Li, Y. (2006). Early estimates of the size of branch-and-bound trees. *INFORMS Journal on Computing*, 18(1):86–96.

[Kilby et al., 2006] Kilby, P., Slaney, J., Thiébaux, S. e Walsh, T. (2006). Estimating search tree size. *ipi*, 1:0.

[Knuth, 1975] Knuth, D. E. (1975). Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29:121–136.

[Kreher e Stinson, 1999] Kreher, D. L. e Stinson, D. R. (1999). *Combinatorial algorithms: generation, enumeration, and search*. CRC Press LTC.