UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Matemática, Estatística e Computação Científica

ESTHER SOFÍA MAMIÁN LÓPEZ

Método de pontos interiores para estimar os parâmetros de uma gramática probabilística livre de contexto

Campinas

2018

Esther Sofía Mamián López

Método de pontos interiores para estimar os parâmetros de uma gramática probabilística livre de contexto

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutora em Matemática Aplicada.

Orientador: Aurelio Ribeiro Leite de Oliveira

Este exemplar corresponde à versão final da Tese defendida pela aluna Esther Sofía Mamián López e orientada pelo Prof. Dr. Aurelio Ribeiro Leite de Oliveira.

Campinas

2018

Agência(s) de fomento e nº(s) de processo(s): CAPES

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

M31m

Mamián López, Esther Sofía, 1985-
Método de pontos interiores para estimar os parâmetros de uma gramática probabilística livre de contexto / Esther Sofía Mamián López. – Campinas, SP : [s.n.], 2018.

Orientador: Aurelio Ribeiro Leite de Oliveira.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Métodos de pontos interiores. 2. Processamento de linguagem natural (Computação). 3. Gramáticas probabilísticas livres de contexto. I. Oliveira, Aurelio Ribeiro Leite de, 1962-. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Interior point method for estimating of parameters for stochastic context-free grammar
Palavras-chave em inglês:
Interior-point methods
Natural language processing (Computer science)
Probabilistic context-free grammars
Área de concentração: Matemática Aplicada
Titulação: Doutora em Matemática Aplicada
Banca examinadora:
Aurelio Ribeiro Leite de Oliveira [Orientador]
Glaucia Maria Bressan
Antonio Augusto Chaves
Marcia Aparecida Gomes Ruggiero
Estevão Esmi Laureano
Data de defesa: 30-01-2018
Programa de Pós-Graduação: Matemática Aplicada

Tese de Doutorado defendida em 30 de janeiro de 2018 e aprovada pela banca examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). AURELIO RIBEIRO LEITE DE OLIVEIRA

Prof(a). Dr(a). GLAUCIA MARIA BRESSAN

Prof(a). Dr(a). ANTONIO AUGUSTO CHAVES

Prof(a). Dr(a). MARCIA APARECIDA GOMES RUGGIERO

Prof(a). Dr(a). ESTEVÃO ESMI LAUREANO

As respectivas assinaturas dos membros encontram-se na Ata de defesa

Agradecimentos

Este trabalho não teria sido possível sem a convergência de pessoas que confluem na nossa vida e que contribuem gratamente no desenvolvimento do trabalho. Um especial agradecimento:

- com todo meu amor, para Emiro Mamián e Aydée López pelo incondicionável apoio e fortaleza carregada de amor,
- à minhas irmãs e irmão pela parceria e apoio de sempre,
- para Juan Carlos, por estes anos de parceria e amor, por seu apoio infinito que fortalece o espirito, toda minha admiração e meu amor,
- para meu orientador professor Dr. Aurelio Ribeiro Leite de Oliveira pelas contribuições acadêmicas, pela formação, mas também por todo seu apoio em outros aspectos da vida,
- ao professor Dr. Fredy Angel Amaya, sempre com o entusiasmo de colaborar e contribuir,
- aos professores e comunidade em geral do Instituto de Matemática, Estatística e Computação Científica pelo importante trabalho que realizam na nossa formação como pesquisadores,
- aos amigos e amigas brasileiros pelo carinho e a amizade que me ofereceram,
- aos amigos e amigas colombianas por ser uma segunda família,
- à banca examinadora, Profa. Dra. Glaucia Maria Bressan, Prof. Dr. Antonio Augusto Chaves, Profa. Dra. Marcia Aparecida Gomes Ruggiero, Prof. Dr. Estevão Esmi Laureano, pela disposição de contribuir e as sugestões dadas,
- à PEC-PG/CAPES pelo suporte financeiro,
- ao Brasil e à UNICAMP todo meu carinho pela oportunidade de me formar como pesquisadora, por ser um país de braços abertos e me acolher estes anos.

Resumo

No marco do Processamento da Linguagem Natural (PLN), nosso objetivo é modelar uma linguagem natural, por exemplo: Inglês, Português, etc. O modelo probabilístico mais simples para modelar linguagens naturais, são as chamadas Gramáticas Probabilísticas Livres de Contexto (GPLC), isto é, uma gramática livre de contexto na qual cada regra tem associado um valor de probabilidade. Para o processo de treino de uma GPLC (determinar as probabilidades associadas às regras), supomos que a parte estrutural da GPLC é dada, quer dizer, os símbolos não terminais, símbolos terminais, regras da gramática e símbolo inicial são conhecidos. Treinar a gramática pode ser entendido como o processo de encontrar os valores de probabilidade ótimos para cada una das regras da GPLC. A estratégia está baseada em um treino a partir de um *corpus*, onde este além de conter a parte estrutural da GPLC, contém exemplos de sentenças da linguagem. Assim determinamos uma função critério da amostra e um método para otimizá-la. O método clássico para estimar os parâmetros de um GPLC é o método *Inside-Outside*, este demanda uma grande quantidade de tempo tornando-o inviável para aplicações complexas, pois é de ordem de $O(m^3n^3)$ onde m é o comprimento da sentença, e n é o número de símbolos não terminais da gramática. Nesse contexto, nossa proposta é estudar a aplicação de métodos de pontos interiores primal-dual para o processo de treinamento de uma gramática probabilística livre de contexto. A escolha de este método é em razão que eles são bem-sucedidos para problemas de programação não linear de grande porte.

Palavras-chave: Métodos de pontos interiores. Processamento de linguagem natural. Gramáticas probabilísticas livres de contexto.

Abstract

In many Natural Language Processing tasks, one aim is modeling Natural Language (NL), ie, English, Portuguese, etc. The simplest probabilistic model for NL is a Probabilistic Context Free Grammar (PCFG), which is a context free grammar with probabilities added to the rules. For training a PCFG (to calculate probabilities), we assume that the structure of the grammar in terms of the number of terminals and nonterminals symbols, the rules, and the start symbol are given in advance. Training the grammar comprises simply a process that tries to find the optimal probabilities to assign for different grammar rules within this grammar structure. The approach, for this purpose, is corpus-based work, meaning we have a corpus (sample of sentences) of a natural language and we will training the grammar with it. From corpus we recover structural part of grammar and a sample of sentences, so we need defined a criterion function and optimization method for probabilities assign to grammar rules. The Inside-Outside algorithm allows us to train the parameters of a PCFG on sentences of a natural language. However, for each sentence, each iteration of training is $O(m^3n^3)$, where m is the length of sentence, and n is the number of nonterminal symbols in the grammar. For this, our aim is studying primal-dual Interior Point Methods for training probabilistic context free grammar, because computational experiments and the theoretical development, have show that primal-dual algorithms perform well for large-scale nonlinear programming problems.

Keywords: Interior point method. Natural language processing. Probabilistic context-free grammar.

Lista de ilustrações

Lista de tabelas

Lista de Métodos

Sumário

Introdução

A origem da teoria da linguagem formal ocorreu na metade da década de 50 com o desenvolvimento de modelos matemáticos para gramáticas relacionados com o trabalho baseado em linguagem natural feito por Noam Chomsky [Cho53]. Um dos objetivos nesta área foi desenvolver modelos computacionais de gramáticas capazes de descrever linguagens naturais, tais como Inglês, Português, etc. Fazendo isso, tinha-se a esperança de poder ensinar às máquinas a interpretar linguagens naturais.

Entre os anos 60 e 90 a linguística, a psicologia, a inteligência artificial e o Processamento de Linguagem Natural (PLN) foram dominadas por teorias e métodos baseados em abordagens *racionalistas*, justificadas principalmente pela grande aceitação dos argumentos e teorias de Noam Chomsky [CDM03]. As crenças racionalistas na inteligência artificial caracterizavam-se pela criação de sistemas inteligentes com grande quantidade de conhecimento inicial codificado à mão.

A linha de pesquisa *empiricista* para o processamento da linguagem natural ressurge no final da década dos 80, depois de mais de 25 anos nas sombras das crenças da linha racionalista. Segundo o empiricismo uma máquina poderia aprender a estrutura complexa e extensa de uma linguagem apenas observando uma grande quantidade de exemplos e usando procedimentos estatísticos; métodos de associação e generalização indutiva, tal como o aprendizado de regras [Bri93].

Na atualidade, existe um grande entusiasmo pelos métodos estatísticos no processamento de linguagem natural, dado que em diversos sub-problemas de PLN, tais métodos proporcionaram soluções práticas que não foram obtidas usando métodos tradicionais do processamento da linguagem natural. As Gramáticas Probabilísticas Livres de Contexto (GPLC) pertencem ao grupo dos modelos probabilísticos que têm sido usados com grande sucesso em várias tarefas do processamento de linguagem natural. Em anos recentes, os pesquisadores colocaram grande ênfase em soluções práticas de engenharia, desenvolvendo métodos que trabalham com textos, que não necessariamente têm sentido gramatical e comparam os diferentes métodos. Esta nova ênfase é chamada de tecnologia da linguagem ou engenharia da linguagem. Neste tópico usam-se bastante os métodos estatísticos porque, no aprendizado automático e em problemas de desambiguação, os bons resultados assim o sugerem.

Uma gramática probabilística livre de contexto é uma extensão natural da uma gramática livre de contexto. Uma GPLC está constituída basicamente de duas partes: sua parte estrutural, entre os quais está o conjunto de regras da gramática e sua parte estocástica: um conjunto de valores de probabilidade associadas às regras. O

enfoque neste trabalho é o aprendizado das regras de uma GPLC, ou seja, atribuir valores de probabilidade às regras da gramática. Deste modo determinaremos uma GPLC que representa uma linguagem.

Um dos algoritmos clássicos para estimar os valores de probabilidade de uma gramática probabilística livre de contexto é o algoritmo Inside-Outside [Bak79]. Este foi proposto por James K. Baker no ano 1979 como uma generalização do algoritmo *forward-backward* para estimar os parâmetros do modelo oculto de Markov para gramáticas probabilísticas livres de contexto. Embora seja um dos métodos mais usados, comparado com outros métodos, o algoritmo Inside-Outside é muito lento; para cada sentença e em cada iteração em que for necessário realizar o treino, seu tempo é $O(m^3n^3)$, onde m é o número de símbolos não terminais e n o comprimento da sentença. Detalhes do método Inside-Outside podem ser lidos em [Bak79].

Neste trabalho propomos explorar um método de pontos interiores barreira logarítmica primal-dual, como alternativa para estimar os valores de uma gramática probabilística livre de contexto. Implementamos o método, que será utilizado para o aprendizado de regras a partir do *corpus*[1] *Tycho Brahe* [MSM93] e fazemos o estudo pertinente para determinar a viabilidade do método aplicado ao treino de uma GPLCs. Além disso, para poder definir o problema de otimização a ser resolvido, desenvolvemos um método, que é uma variação do método Cocke-Younger-Kassami [CDM03].

Esta monografia está organizada da seguinte maneira; no Capítulo 1 fazemos uma revisão teórica dos conceitos de autômatos, gramáticas e linguagens; no Capítulo 2 discutimos o método Inside-Outside e como ele é usado para o problema de aprendizado da gramática (estimação dos parâmetros da GPLC). Apresentamos o conceito de *perplexidade por palavra* como uma medida para determinar quão bom é o modelo obtido. O Método de Cocke-Younger-Kassami e uma variação desenvolvida para as condições do trabalho são apresentados no Capítulo 3. O Capítulo 4 foi reservado para uma revisão do método de pontos interiores que será usado para estimar os parâmetros da gramática probabilística livre de contexto. No Capítulo 5 apresentamos os resultados obtidos os quais são submetidos a discussão ao final do mesmo. Finalizamos o documento apresentando as conclusões e as perspectivas futuras.

[1] O corpo de um texto é chamado de *corpus* (latim para corpo). O plural de *corpus* é *corpora*. Para nosso contexto, a partir do *corpus* pode-se extrair as informações estruturais da gramática, assim como as sentenças da linguagem que representa, pelo qual, ao falar de *corpus* fazemos referência à amostra da linguagem que queremos representar.

1 Autômatos, Gramáticas e Linguagens

1.1 Introdução

As pessoas falam e escrevem de diferentes maneiras, e, aquilo que se fala tem sempre alguma estrutura e regularidade. Na linguística, a sintaxe tem como propósito identificar tal estrutura que permite analisar como os grupos de palavras trabalham e relacionam-se umas com outras, tanto num contexto como de forma individual. As linguagens têm uma estrutura recursiva complexa e os modelos baseados em árvores sintáticas permitem capturar essa informação. O modelo mais simples para capturar esse propósito é uma gramática probabilística livre de contexto [CDM03], a qual é uma extensão das gramáticas livres de contexto que tem adicionalmente uma função que atribui a cada uma das regras da gramática um valor de probabilidade.

Embora e por múltiplas razões, as gramáticas livres de contexto não são consideradas adequadas para descrever linguagens naturais tais como o Inglês, elas jogam um papel importante na computação linguística e outras áreas [HMU01]. Uma importante aplicação das gramáticas livres de contexto acontece na compilação e especificação de linguagens de programação. A gramática de uma linguagem de programação é uma referência para quem quer aprender a sintaxes da linguagem. Projetistas de compiladores e interpretadores para linguagens de programação geralmente começam obtendo uma gramática da linguagem. Vários compiladores, interpretadores e outras importantes aplicações como os sistemas de Extração da Informação, Tradução Automática ou Reconhecimento de Fala contêm uma componente chamada de *parser* ou analisador sintático que extrai o significado formal ou estrutura sintática de uma sentença.

Em sua forma mais simples, o problema de análise sintática consiste na criação de um algoritmo onde seus dados de entrada são sentenças de uma linguagem, que associa a cada sentença sua correspondente árvore sintática [Col99]. Um dos maiores problemas na análise sintática são os diferentes tipos de ambiguidade, um deles acontece quando uma sentença tem associada mais de uma árvore sintática. Uma das abordagens utilizadas para sortear estes problemas são os métodos estatísticos, basicamente, porque os problemas de ambiguidades podem ser vistos como problemas de eleição.

As abordagens estatísticas para o processamento de linguagem natural, tentam resolver o problema de desambiguização com um aprendizado automático do léxico e das preferências estruturais a partir de um *corpus*. Para abordar este problema se define uma função critério que depende do *corpus* e de um método para otimizá-la. A função critério para otimizar que geralmente se define é a função de verossimilhança do corpus [Bak79],

[Col99]. O método para otimizar a função de verossimilhança é discutido em posteriores capítulos desta monografia.

No desenvolvimento deste capítulo apresentamos conceitos e exemplos que ajudam esclarecer como as árvores sintáticas estão relacionadas às gramáticas probabilísticas livres de contexto. Assim como as definições formais dos conceitos de gramática, gramática livre de contexto, gramática probabilística livre de contexto. No final deste capítulo apresentamos como obter a função critério a partir do corpus, chamada de função de verossimilhança.

1.2 Teoria dos Autômatos

Na teoria da computação existem três áreas centrais: autômatos, computabilidade e complexidade. Elas estão fortemente relacionadas pela questão:

Quais são as capacidades e limitações dos computadores?

Num esforço para responder esta pergunta, a teoria da computação examina quais problemas computacionais podem ser resolvidos em modelos teóricos de cômputo, assim como os custos espaciais e temporais associados às diferentes abordagens utilizadas ao resolver inumeráveis problemas computacionais.

A teoria dos autômatos lida com as definições e propriedades de modelos matemáticos de computação. Na ciência da computação, esses modelos desempenham um papel em diversas áreas aplicadas. Um modelo chamado de autômato finito é usado em processamento de texto, compiladores e projeto de *hardware*. Outro modelo, denominado gramática livre de contexto, é utilizado em linguagens de programação e inteligência artificial [Sip06].

Um autômato finito é uma lista de cinco objetos: conjunto de estados, um alfabeto de entrada, regras de movimentação, estado inicial e estado final ou de aceitação. Com frequência existe um ou mais estados como estados finais ou estados de aceitação. A chegada em um desses estados de aceitação, após uma sequência de entradas, indica que a sequência de entrada é boa em algum aspecto, em caso de não chegar ao estado final, dizemos que a sequência de entrada é rejeitada.

Formalmente, um autômato finito é uma 5-tupla definida como:

Definição 1.2.1. *[HMU01] Um autômato finito é 5-tupla* $(Q, \Sigma, \delta, q_0, F)$ *onde*

1. Q *é um conjunto finito conhecido como* ***estados****.*
2. Σ *é um conjunto finito conhecido como* ***alfabeto*** *ou* ***símbolos de entrada****.*

3. $\delta : Q \times \Sigma \to Q$ *é a* ***função de transição****.*

4. $q_0 \in Q$ *é o* ***estado inicial****.*

5. $F \subseteq Q$ *é o conjunto de* ***estados finais*** *ou* ***estados de aceitação****.*

Note que a função δ na Definição 1.2.1, define as regras de movimentação. Se o autômato finito têm uma seta de um estado x para um estado y rotulada como o símbolo de entrada 1, significa que, se o autômato está no estado x quando lê 1 então ele se move para o estado y. Usando a função de transição, o anterior é reescrito como $\delta(x, 1) = y$.

Usando a notação formal da Definição 1.2.1 podemos descrever um exemplo de autômato finito especificando cada uma de suas cinco partes:

Exemplo 1.2.1. *Seja o seguinte autômato* $M1 = (Q, \Sigma, \delta, q_0, F)$ *onde:*

1. $Q = \{q_1, q_2, q_3\}$

2. $\Sigma = \{0, 1\}$

3. δ *é descrito como*

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 *é o estado inicial.*

5. $F = \{q_2\}$

Por exemplo, quando a máquina $M1$ lê a entrada 1101 procede da seguinte forma:

1. Começa no estado q_1.

2. Lê 1 e vai do estado q_1 para q_2.

3. Lê 1 e vai do estado q_2 para q_2.

4. Lê 0 e vai do estado q_2 para q_3.

5. Lê 1 e vai do estado q_3 para q_2.

6. *Aceita* porque $M1$ tem como estado final o q_2.

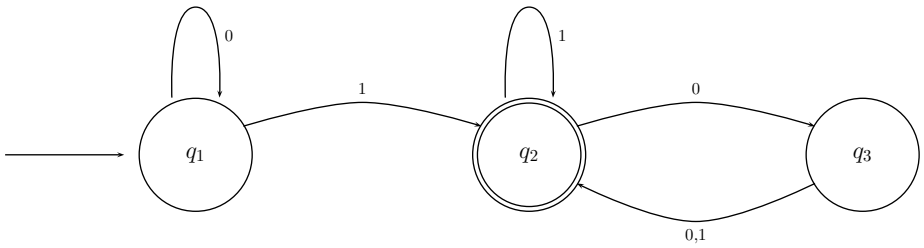

Figura 1 – Autômato finito $M1$ que possui três estados. É usual designar os estados de aceitação por um círculo duplo.

Quando o autômata recebe uma entrada como 1101, ele processa essa entrada e devolve uma saída que pode ser: *aceita* ou *rejeita*. Dizemos que o conjunto A de entradas que foram aceitas pelo autômata é a *linguagem* do autômata $M1$, ou que $M1$ *reconhece* A.

Definição 1.2.2. *[HMU01] Uma linguagem é dita Linguagem Regular se existe algum autômato finito que a reconhece.*

1.2.1 Operações Regulares de Linguagens

Uma vez definido o conceito de autômato finito e linguagem regular é importante estudar algumas de suas propriedades.

Assim como na aritmética temos objetos básicos (os números) e as operações ($+$, $-$, $\times$ e $\div$) para manipulá-los, analogamente na teoria da computação, os objetos são as linguagens e as operações para manipulá-los são as chamadas *operações regulares*:

Definição 1.2.3. *Seja A e B linguagens. As operações regulares são definidas como* ***união****,* ***concatenação*** *e* ***estrela*** *[Sip06].*

- *União* $A \cup B = \{x : x \in A \text{ ou } x \in B\}$.
- *Concatenação* $A \circ B = \{xy : x \in A \text{ e } y \in B\}$.
- *Estrela* $A^* = \{x_1 x_2 ... x_k : k \geqslant 0 \text{ e } x_i \in A, i = 1...k\}$.

Observe que a operação estrela é uma operação unitária e não binária como no caso das outras duas operações.

É possível provar que a coleção de linguagens regulares é fechada com as três operações regulares [Sip06].

1.2.2 Autômato Finito não Determinístico

Ao invés dos Autômatos Finitos Determinísticos (AFD), denominados até agora como autômatos finitos, os Autômatos Finitos Não Determinísticos (AFND) são aqueles que possuem a propriedade de estar em vários estados ao mesmo tempo. Analogamente aos autômatos finitos determinísticos, um autômato finito não determinístico possui um conjunto finito de estados, um conjunto finito de símbolos de entrada, um estado inicial e um conjunto de estados de aceitação. No entanto, eles diferem no tipo da função δ. No caso do autômato finito não determinístico, a função δ retorna um conjunto de estados. E, no caso em que a função δ esteja associada a um um autômato finito determinístico, retorna exatamente um estado [HMU01].

As diferenças entre um autômato finito determinístico e um autômato finito não determinístico são:

- cada estado de um AFD possui n saídas de transição associadas aos n elementos do alfabeto. Na Figura 2, por exemplo, vemos como isso não é satisfeito, pois o estado q_1 possui uma saída para o símbolo 0, mas possui duas para o símbolo 1; o estado q_2 têm uma seta para 0, mas nenhuma até 1. Em um AFN uma saída de transição pode estar associada a zero, um ou múltiplos símbolos do alfabeto;
- no caso de um AFD as saídas de transição são associadas a símbolos do alfabeto, no caso de um AFND pode ser associadas a elementos do alfabeto o símbolo ε (vazio). Podem existir zero, uma, ou muitas saídas de transição associadas ao ε.

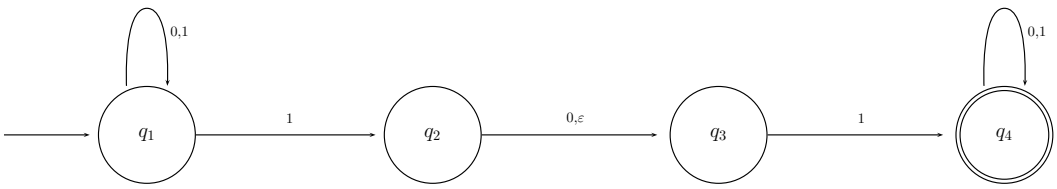

Figura 2 – Autômato finito não determinístico.

Informalmente, um AFND funciona da seguinte forma: vamos supor que o autômato da Figura 2 esteja no estado q_1 e que o seguinte símbolo de uma cadeia de entrada é 1, temos então várias (duas para nosso exemplo) opções para proceder. Neste caso o autômato faz um certo número de cópias dele mesmo (no exemplo, duas) e segue todas as possibilidades em paralelo. E, repete o anterior para cada cópia. Se o próximo símbolo de entrada não está associado a nenhuma saída de transição de estado, então

a cópia não tem mais sentido e é desconsiderada junto com o caminho levado até ela. Finalmente, se qualquer uma dessas cópias está num estado de aceitação no final da entrada, o AFND aceita a cadeia de entrada [Sip06].

Assim, como no caso de AFD, dizemos que o conjunto A de cadeias de entradas que foram aceitas pelo AFND é sua linguagem ou que o AFND reconhece a linguagem A.

Segue a definição formal de um autômato finito não determinístico:

Definição 1.2.4. *Um autômato finito não determinístico é 5-tupla* $(Q, \Sigma, \delta, q_0, F)$ *onde [HMU01]*

1. Q *é um conjunto finito conhecido como* ***estados****.*
2. Σ *é um conjunto finito conhecido como* ***alfabeto*** *ou* ***símbolos de entrada****.*
3. $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ [1] *é a* ***função de transição****. Ela retorna um conjunto de estados ao invés de um único estado como no caso do autômato finito determinístico.*
4. $q_0 \in Q$ *é o* ***estado inicial****.*
5. $F \subseteq Q$ *é o conjunto de* ***estados de aceitação****.*

Não determinístico é uma generalização de determinístico, logo cada autômato finito determinístico é automaticamente um autômato finito não determinístico, e dizemos que dois autômatos são equivalentes se reconhecem a mesma linguagem.

Teorema 1.2.1. *Todo autômato finito não determinístico tem um autômato finito determinístico equivalente [Sip06].*

Demostração 1.2.1. *A demostração pode ser estudada em [Sip06], página 76.*

O Teorema 1.2.1 é uma alternativa para caracterizar as linguagens regulares. O seguinte corolário estabelece este fato.

Corolário 1.2.1. *Uma linguagem é dita regular se, e somente se, existe um autômato finito não determinístico que a gere [Sip06].*

1.2.3 Expressões Regulares

As expressões regulares são outro tipo de notação para a definição de linguagens, também podem ser consideradas uma linguagem de programação na qual expressamos algumas aplicações importantes, como aplicações de pesquisa em textos ou componentes de

[1] Notando que $\mathcal{P}(Q)$ é o conjunto *power* de Q, ou seja, ele contém todos os sub-conjuntos de Q.

compiladores. As expressões regulares estão intimamente relacionadas com os autômatos finitos não determinísticos [HMU01].

As expressões regulares têm um papel importante em aplicações da ciência da computação. Em aplicações envolvendo texto, os usuários querem poder fazer busca por cadeias que satisfaçam certos padrões. Expressões regulares proveem um método poderoso para descrever tais padrões. Utilitários tais como AWK [2] e GREP [3] no UNIX, linguagens de programação modernas tais como PERL (www.perl.org) e editores de texto, todos eles proveem mecanismos para descrição de padrões usando expressões regulares [Sip06].

Um exemplo de uma expressão regular que descreve uma linguagem é:

$$(0 \cup 1)0^*,$$

onde a linguagem que representa consiste em todas as cadeias que começam com 0 ou 1 e segue com um número n de zeros. Essa expressão regular pode ser entendida em partes: primeiro, os símbolos 0 e 1 são abreviações para os conjuntos $\{0\}$ e $\{1\}$. Dessa forma, $(0 \cup 1)$, significa $(\{0\} \cup \{1\})$. O valor dessa parte é a linguagem $\{0, 1\}$. A parte 0^* significa $\{0\}^*$ e, seu valor é a linguagem de todas a cadeias que têm qualquer número de zeros. Finalmente, o símbolo de concatenação $\circ$, geralmente está implícito em expressões regulares, assim $(0 \cup 1)0^*$ é uma abreviação de $(0 \cup 1) \circ 0^*$. A concatenação junta as cadeias das duas partes para obter o valor da expressão inteira.

Definição 1.2.5. *Seja Σ um alfabeto, dizemos que R é uma expressão sobre Σ, e o conjunto que assim o denota está definido recursivamente como segue [Sip06]:*

1. *a é uma expressão regular e denota o conjunto $\{a\}$,*
2. *ε é uma expressão regular e denota o conjunto $\{\varepsilon\}$,*
3. *$\varnothing$ vazio,*
4. *$(R_1 \cup R_2)$, donde R_1 e R_2 são expressões regulares,*
5. *$(R_1 \circ R_2)$, donde R_1 e R_2 são expressões regulares,*
6. *(R_1^*), donde R_1 é expressão regular.*

Nos itens 4, 5 e 6 as expressões representam as linguagens obtidas tomando-se a união ou concatenação das linguagens R_1 e R_2, ou a estrela da linguagem R_1, respectivamente.

[2] AWK é uma linguagem de programação desenvolvido para processar dados baseados em texto: fluxo de dados ou ficheiros [AKW87].

[3] GREP é um utilitário para busca de texto em arquivos, é usado por vários pacotes ou outros utilitários no sistema UNIX.

Expressões regulares e os autômatos finitos são equivalentes no poder descritivo. Qualquer expressão regular pode ser convertida para um autômato finito que reconhece sua linguagem e vice-versa.

Teorema 1.2.2. *Uma linguagem é dita regular se, e somente se, existe uma expressão regular que a descreva.*

Demostração 1.2.2. *A demonstração encontra-se no Capítulo 1, na página 66, em [Sip06].*

1.3 Linguagens e gramáticas livres de contexto

Nesta seção vamos introduzir a notação de uma gramática livre de contexto e mostramos como as gramáticas definem linguagens. Discutimos como uma árvore de análise sintática representa a estrutura gramatical das sentenças de uma linguagem. Esta árvore é gerada pelo analisador sintático ou *parsing*, em inglês.

A coleção de linguagens associadas às gramáticas livres de contexto são chamados de linguagens livres de contexto. As linguagens livres de contexto formam uma classe maior das linguagens regulares, ver Figura 3.

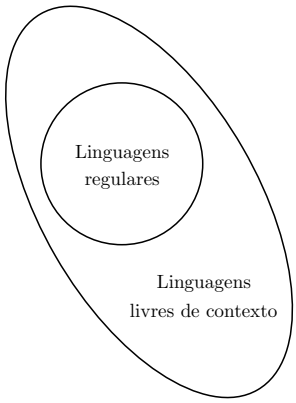

Figura 3 – Relação entre linguagens regulares e linguagens livres de contexto.

1.3.1 Definição das Gramáticas Livres de Contexto

Nas seções anteriores foram introduzidos informalmente os principais conceitos da teoria de linguagem: alfabetos, cadeias e linguagem. Formalmente eles são definidos como:

Definição 1.3.1. *[CDM03]*

(a) Um alfabeto ou vocabulário, denotado por Σ, é um conjunto finito de símbolos.

(b) Uma cadeia ou sentença é uma sequência finita de símbolos, pertencentes a um alfabeto Σ.

(c) O comprimento de uma cadeia χ *é dado pelo número de símbolos que a compõem, denotado por* $|\chi|$.

(d) Uma cadeia ϵ *é dita vazia quando está constituída por nenhum símbolo, neste caso* $|\epsilon| = 0$.

(e) Σ^* *apresenta o conjunto de todas as cadeias de um alfabeto* Σ.

(f) Σ^+ *apresenta o conjunto de todas as cadeias de* Σ*, tal que seu tamanho é maior ou igual a um.*

(g) Uma forma sentencial é uma sequência finita de símbolos pertencentes ao conjunto $(N \cup \Sigma)^*$.

(h) Uma linguagem L *sobre* Σ *é definida como um sub-conjunto do conjunto* Σ^*.

Existem quatro componentes importantes na descrição gramatical de uma linguagem:

- um alfabeto Σ cujos símbolos são denominados símbolos terminais. Vamos usar letras minúsculas como a, b, c, d para representá-los,
- um conjunto finito de variáveis ou símbolos não terminais, denotado por N com $N \cap \Sigma = \varnothing$. As letras maiúsculas A, B, C, D representam os símbolos não terminais,
- uma variável S, denominada variável de partida. $S \in N$,
- um conjunto finito P de regras de derivação. Cada derivação têm a forma $\alpha \longrightarrow \gamma$, onde α e γ são sequências finitas de símbolos de $(N \cup \Sigma)^*$. A expressão $\alpha \longrightarrow \gamma$ significa que α é substituída por γ. Onde α e γ são as denominadas formas sentenciais.

Definição 1.3.2. *[Sip06] Uma gramática formal é uma 4-tupla* $G = (N, \Sigma, P, S)$*, onde* N, Σ, P *e* S *estão acordo com os itens anteriores.*

Definição 1.3.3. *[HMU01] Sejam* $\gamma_1, \gamma_2 \in (N \cup \Sigma)^*$*, suponha que existe uma sequência de regras de derivação* $q_1, q_2, ..., q_{m-1} \in P$ *e formas sentenciais* $\alpha_1, \alpha_2, ..., \alpha_m \in (N \cup \Sigma)^*$*,* $m \geqslant 1$ *tal que*

$$\gamma_1 = \alpha_1 \overset{q_1}{\Rightarrow} \alpha_2 \overset{q_2}{\Rightarrow} ... \overset{q_{m-1}}{\Rightarrow} \alpha_m = \gamma_2,$$

onde $\alpha_i \overset{q_i}{\Rightarrow} \alpha_{i+1}$ *significa que* α_{i+1} *é derivado de* α_i *usando a regra* q_i *uma única vez. Se diz que há uma derivação de* γ_1 *em* γ_2*, denotada por* $\gamma_1 \overset{*}{\Rightarrow} \gamma_2$.

Definição 1.3.4. *[HMU01] Uma derivação à esquerda de uma cadeia $\chi \in \Sigma^+$ em G é uma sucessão de regras de derivação de $d_x = (q_1, q_2, \ldots, q_m)$, $m \geqslant 1$, tal que: $S = \alpha_1 \overset{q_1}{\Rightarrow} \alpha_2 \ldots \overset{q_{m-1}}{\Rightarrow} \alpha_m = \chi$, onde $\alpha_i \in (N \cup \Sigma)^+$, com $1 \leqslant i \leqslant m$ e q_i substitui o símbolo não terminal mais à esquerda de α_i.*

A definição de derivação à direita de uma cadeia é equivalente. Neste trabalho vamos usar derivação à esquerda, somente. Logo, de agora em diante por derivação entenda-se como derivação à esquerda.

Definição 1.3.5. *[HMU01] Uma gramática G na qual o conjunto P de regras de derivação está constituído por regras da forma $A \longrightarrow \alpha$, onde $A \in N$ e $\alpha \in (N \cup \Sigma)^+$, denomina-se gramática livre de contexto. Ver gramática G_1 da Figura 4.*

Definição 1.3.6. *[HMU01] Uma linguagem livre de contexto $L(G)$ gerada pela gramática livre de contexto $G = (N, \Sigma, P, S)$, consiste no conjunto*

$$L(G) = \{\chi \in \Sigma^+ : S \overset{*}{\Rightarrow} \chi\}.$$

1.3.2 Exemplo

Seja G_1 na Figura 4 uma gramática livre de contexto. Ela consiste em uma coleção de regras de derivação onde cada regra está composta de um símbolo não terminal no lado esquerdo e uma forma sentencial do lado direito, separados por uma seta. Lembrando que os símbolos escritos com maiúsculas são os símbolos não terminais e as formas sentenciais estão constituídas tanto por símbolos não terminais, como por símbolos terminais os quais estão denotados com letras minúsculas. Numa gramática, um símbolo não terminal é designado como símbolo inicial. Por exemplo, a gramática G_1 contém três regras de derivação, possui dois símbolos não terminais A e B, onde dizemos que A é o símbolo inicial e os símbolos terminais são a,b e c.

$$\begin{array}{lcl} A & \rightarrow & aAb \\ A & \rightarrow & B \\ B & \rightarrow & c \end{array}$$

Figura 4 – Gramática G_1.

A Gramática G_1 na Figura 4, gera uma linguagem onde cada cadeia dessa linguagem pode ser obtida da seguinte forma:

1. escrevemos o símbolo inicial A,
2. procuramos uma regra onde o lado esquerdo esteja o símbolo A e substituímos o símbolo não terminal pelo lado direito (forma sentencial) dessa regra,

3. da esquerda a direita tomamos o símbolo não terminal na forma sentencial e procuramos uma regra onde o lado esquerdo é o símbolo não terminal tomado, e o substituímos pelo lado direito,

4. repetimos o passo 3 até que não existam símbolos não terminais na forma sentencial que está sendo atualizada.

Assim, a gramática G_1 gera a cadeia *aaacbbb* e a sequência de substituições para obter a cadeia é chamada de derivação. Por exemplo a derivação da cadeia *aaacbbb* é:

$$A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aaaAbbb \Rightarrow aaaBbbb \Rightarrow aaacbbb.$$

É possível representar essa mesma derivação usando uma árvore de análise sintática, como na Figura 5.

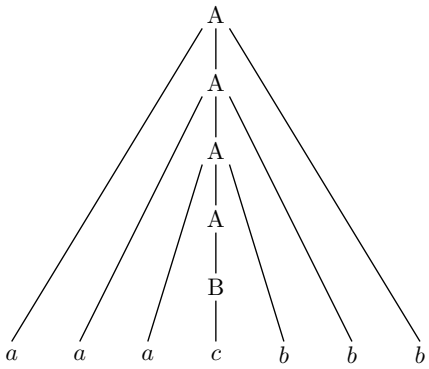

Figura 5 – Árvore sintática para *aaacbbb* na gramática G1.

O conjunto de todas as cadeias geradas desta maneira é chamado de linguagem. Denotamos como $L(G_1)$ a linguagem da gramática G_1.

1.3.3 Árvore sintática

Como pode ser observado no final do último exemplo apresentado, um jeito bastante útil para escrever as derivações através de árvores, são as chamadas árvores sintáticas. Assim abrimos um breve parênteses para estabelecer formalmente as árvores sintáticas. Seja $G = (\Sigma, N, S, P)$ uma gramática livre de contexto, uma árvore é uma árvore sintática para G se satisfaz o seguinte:

- cada vértice da árvore têm uma etiqueta que pode ser um símbolo de $(\Sigma \cup N)$,

- a etiqueta da raiz é S, por convenção,

- se um vértice é interior e está etiquetada com o símbolo A, necessariamente $A \in N$,
- se um vértice tem etiqueta A e existem k vértices com etiquetas $X_1, X_2, ..., X_k$ tal que são filhas do vértice A, então diremos que

$$A \to X_1 X_2 ... X_k, \tag{1.1}$$

  é uma regra de produção que pertence a P,
- se um vértice possui a etiqueta ϵ diremos que ele é uma folha[4] sendo a única filha do pai.

Observe que no caso da árvore sintática apresentada na Figura 5, os vértices com as etiquetas A e B são os vértices interiores.

1.3.4 Forma Normal de Chomsky

Em geral, quando trabalhamos com gramáticas livres de contexto é conveniente reduzi-la numa forma bem mais simples, sem perder o poder generativo[5] da gramática. A forma mais simples e mais utilizada é a Forma Normal de Chomsky (FNC) [Sip06].

Vamos supor L uma linguagem livre de contexto diferente de vazio, logo ela pode ser gerada por uma gramática livre de contexto G com as seguintes propriedades:

- cada símbolo terminal e cada símbolo não terminal de G deve aparecer na derivação de alguma sentença de L,
- não existem regras de derivação da forma $a \to b$, com a e b símbolos não terminais.

Além disso, se ϵ não pertence a L, então não existem regras de produção da forma $A \to \epsilon$, de fato se ϵ não pertence à L, então necessariamente cada regra de produção deve ser da forma $A \to BC$ ou $A \to a$, com $A, B, C \in N$ e $a \in \Sigma$, esta forma especial é chamada de Forma Normal de Chomsky (FNC). Existe uma outra importante forma denominada Forma Normal de Greibach que pode ser estudada em [HMU01].

Definição 1.3.7. *[HMU01] Uma gramática livre de contexto G está na forma normal de Chomsky, quando as regras de derivação estão na forma: $A \longrightarrow BC$ e $A \longrightarrow a$, onde $A, B, C \in N$ e $a \in \Sigma$.*

Para obter uma gramática em FNC a partir de uma gramática livre de contexto, devemos efetuar diversas simplificações preliminares:

[4] Lembrando que uma folha é um vértice de grau 1.

[5] Poder generativo no sentido da sua capacidade de gerar sentenças da linguagem que está gerando ou modelando

- eliminar símbolos inúteis, as variáveis ou terminais que não aparecem em nenhuma derivação de uma cadeia de terminais a partir do símbolo de início,
- eliminar as ε - produções, aquelas que têm a forma $A \rightarrow \varepsilon$ para alguma variável A,
- eliminar as produções unitárias, as que têm a forma $A \rightarrow B$ para variáveis A e B.

O estudo sobre o procedimento para transformar ou reduzir uma gramática livre de contexto em forma normal de Chomsky pode ser visto em [Sip06, HMU01].

Em virtude do seguinte teorema, neste trabalho usamos gramáticas na forma normal de Chomsky.

Teorema 1.3.1. *[Sip06] Qualquer linguagem livre de contexto, tal que $\epsilon \notin L$, pode ser gerada por uma gramática livre de contexto na forma normal de Chomsky.*

Demostração 1.3.1. *A prova pode ser estudada no Capítulo 2, página 107 em [Sip06]*

Embora as GLC em FNC reconhecem a mesmas linguagens que as GLC, em geral produzem diferentes árvores sintácticas.

1.3.5 Exemplo de uma GLC em FNC

Um segundo exemplo de uma gramática livre de contexto que descreve um fragmento do Inglês e sendo que está na forma normal de Chomsky, é dado na Figura 6 e vamos chamá-la de G_2.

$$
\begin{array}{lcl}
\langle SENTENCE \rangle & \rightarrow & \langle NOUN-PHRASE \rangle \langle VERB-PHRASE \rangle \\
\langle PREPOSITION-PHRASE \rangle & \rightarrow & \langle PREPOSITION \rangle \langle NOUN-PHRASE \rangle \\
\langle VERB-PHRASE \rangle & \rightarrow & \langle VERB \rangle \langle NOUN-PHRASE \rangle \\
\langle VERB-PHRASE \rangle & \rightarrow & \langle VERB-PHRASE \rangle \langle PREPOSITION-PHRASE \rangle \\
\langle NOUN-PHRASE \rangle & \rightarrow & \langle NOUN-PHRASE \rangle \langle PREPOSITION-PHRASE \rangle \\
\langle PREPOSITION \rangle & \rightarrow & \langle with \rangle \\
\langle VERB \rangle & \rightarrow & \langle saw \rangle \\
\langle NOUN-PHRASE \rangle & \rightarrow & \langle astronomers \rangle \\
\langle NOUN-PHRASE \rangle & \rightarrow & \langle ears \rangle \\
\langle NOUN-PHRASE \rangle & \rightarrow & \langle saw \rangle \\
\langle NOUN-PHRASE \rangle & \rightarrow & \langle stars \rangle \\
\langle NOUN-PHRASE \rangle & \rightarrow & \langle telescopes \rangle
\end{array}
$$

Figura 6 – Gramática G_2 na forma normal de Chomsky.

A Gramática G_2 contém seis símbolos não terminais, o símbolo inicial S, sete símbolos terminais os quais estão em letra itálica, e doze regras. Uma sentença ω da linguagem $L(G_2)$ é: $\omega = astronomers\ saw\ start\ with\ ears$.

$S \Rightarrow NP\ \ VP \Rightarrow astronomers\ \ V\ \ NP \Rightarrow astronomers\ \ saw\ \ NP\ \ PP \Rightarrow astronomers\ saw\ stars\ P\ PP \Rightarrow astronomers\ saw\ stars\ with\ ears.$

Figura 7 – Derivação da sentença ω na Gramática G_2.

Na Figura 7 temos a derivação na Gramática G_2 da sentença ω. Note que, usamos as siglas em Inglês dos símbolos não terminais, para abreviar. Logo os símbolos não terminais ficam: S, NP, PP, VP, P e V. A respectiva árvore sintática t_1 da sentença ω pode ser vista na Figura 8.

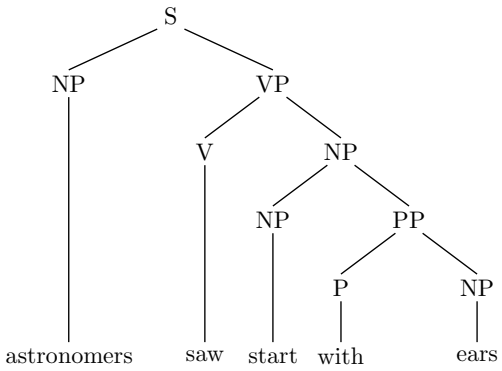

Figura 8 – Árvore sintática t_1 da sentença ω na Gramática G_2.

Observe que, outra maneira de descrever a linguagem de uma gramática é como o conjunto das derivações das árvores da análise sintática, que têm na raiz o símbolo inicial e uma cadeia de terminais como derivação, quer dizer: a sentença.

1.3.6 Ambiguidade em gramáticas e linguagens

Considere a Gramática G_2 descrita na Figura 6 e a sentença ω. Observe que, além da árvore sintática t_1 da Figura 8, essa sentença possui uma segunda árvore de derivação t_2 que está ilustrada na Figura 9.

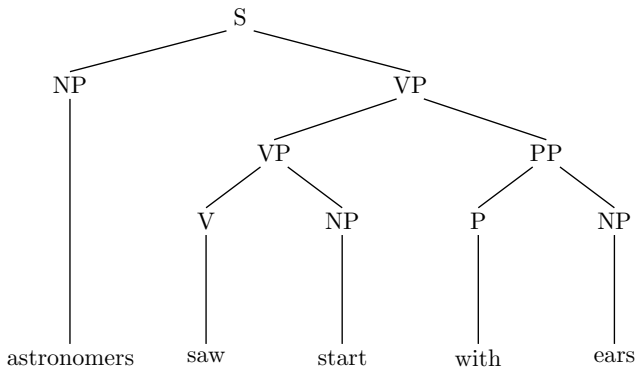

Figura 9 – Árvore sintática t_2 da sentença ω na Gramática G_2.

Esse é um exemplo de como às vezes uma gramática gera uma mesma sentença de diferentes maneiras, tendo então uma sentença associada a diferentes árvores sintáti-

cas e portanto diferentes significados. Essa ambiguidade é um problema para algumas aplicações, tal como em linguagens de programação, onde um programa deve ter uma única interpretação. Outros problemas de ambiguidade, específicos da análise sintática, são revisados em [Col99].

É importante observar que não é uma multiplicidade de derivações[6] que causa ambiguidade, mas sim, a existência de duas o mais árvores de análise sintática. Desse modo, dizemos que uma gramática livre de contexto $G = (\Sigma, N, P, S)$ é ambígua se existe pelo menos uma sentença ω em Σ^* para a qual podemos encontrar pelo menos duas árvores de análise sintática diferentes. Se cada sentença tiver no máximo uma árvore de análise sintática na gramática, a gramática é não-ambígua.

No caso particular da análise sintática, um grande problema é quando a ambiguidade sintática gera uma explosão exponencial de análise sintática para uma sentença. O problema com estes e outros tipos de ambiguidade[7] é ainda agravado quando se trata de sentenças longas, tornando necessária uma gramática bastante ampla [Col99]. Em resposta a estas dificuldades, vários pesquisadores começaram a estudar abordagens baseadas em aprendizado de máquina, principalmente através de métodos estatísticos (com algumas notáveis exceções, tais como métodos de aprendizagem baseados em regras, como em [Bri93] e [HM97]), porque o problema de desambiguação pode ser visto como um problema de eleição.

O aprendizado de máquina se insere num contexto cuja linha de pesquisa é chamada de empiricista, uma vez que se baseia em exemplos já prontos (obtidos a partir de um *corpus*) e aprende como lidar com aqueles ainda não vistos. Um destes modelos estatísticos são as gramáticas probabilísticas livres de contexto. Abordamos este último tópico no próximo capítulo.

1.4 Gramáticas probabilísticas livres de contexto

As gramáticas probabilísticas livres de contexto é o modelo probabilístico mais simples e natural baseado em árvores sintáticas. O uso de modelos estatísticos oferece uma boa solução para o problema de ambiguidade; são robustos; generalizam bem e têm bom comportamento na presença de erros e novos dados. Em problemas reais, têm proporcionado bons resultados práticos que não tinham sido obtidos usando métodos tradicionais. No entanto, alguns pesquisadores sugerem que, em vários aspectos, as GPLCs são modelos fracos para modelar as linguagens. Dado que essa discussão está fora do contexto do nosso trabalho compartilhamos ao leitor algumas referências a respeito [Col99],[Cha96] e [Cha97].

[6] Lembrando que, segundo a ordem em que as substituições das regras sejam efetuadas, diversas derivações podem ser obtidas.

[7] Outros tipos de ambiguidade podem ser estudados e analisados em [HMU01], [CDM03]

Segue definições e conceitos formais deste tópico.

1.4.1 Definição das gramáticas probabilísticas livres de contexto

Uma gramática probabilística livre de contexto é uma gramática livre de contexto que tem valores de probabilidade atribuídos a cada uma de suas regras:

Definição 1.4.1. *[Pei99] Uma gramática probabilística livre de contexto G_p é um par (G, p) tal que G é uma gramática livre de contexto e $p : P \longrightarrow (0, 1]$ é um mapeamento um a um. E, que possui a seguinte propriedade:*

$$\forall A \in N, \sum_{(A \longrightarrow \alpha) \in \Gamma_A} p(A \longrightarrow \alpha) = 1 \tag{1.2}$$

onde $\Gamma_A \in P$ representa o conjunto de regras de derivação cujo antecedente é A.

A Definição 1.4.1 deriva-se naturalmente da definição de linguagem probabilística:

Definição 1.4.2. *[Pei99] Uma linguagem probabilística em um alfabeto Σ é um par (L, ϕ), com L uma linguagem formal e $\phi : \Sigma^* \longrightarrow \Re$ é uma função real nas cadeias de Σ^*. A função da probabilidade satisfaz:*

1. *$\chi \notin L$, então $\phi(\chi) = 0$ para todo $\chi \in \Sigma^*$.*
2. *$\chi \in L$, então $0 < \phi(\chi) \leqslant 1$ para todo $\chi \in \Sigma^*$.*
3. *$\sum_{\chi \in L} \phi(\chi) = 1$.*

Uma vez atribuídos os valores de probabilidade às regras da gramática, é possível definir as probabilidades das sentenças de uma linguagem, e, desta maneira saber por exemplo, se dada uma sentença ela pertence ou não à linguagem. A seguir apresentamos algumas definições necessárias para este objetivo.

Seja G_p uma gramática probabilística livre de contexto, para cada $\chi \in L(G_p)$ denomina-se D_χ o conjunto formado por todas as derivações d_χ da cadeia χ. Por $Nr(q_i, d_\chi)$ designa-se o número de vezes em que a regra q_i foi usada na derivação d_χ.

Definição 1.4.3. *[Pei99] Dada uma gramática probabilística livre de contexto G_p, a probabilidade de uma derivação d_χ da cadeia $\chi \in \Sigma^*$ define-se como:*

$$Pr(\chi, d_\chi | G_p) = \prod_{i=1}^{|P|} p(q_i)^{Nr(q_i, d_\chi)}. \tag{1.3}$$

Definição 1.4.4. *[Pei99] Dada uma gramática probabilística livre de contexto G_p, a probabilidade de uma cadeia $\chi \in \Sigma^*$ define-se como:*

$$Pr(\chi|G_p) = \sum_{d_\chi \in D_\chi} Pr(\chi, d_\chi|G_p). \tag{1.4}$$

Definição 1.4.5. *[Pei99] A linguagem gerada por uma gramática probabilística livre de contexto G_p define-se como $L(G_p) = \{\chi \in L(G)|Pr(\chi|G_p) > 0\}$.*

Definição 1.4.6. *[BT73] Uma gramática probabilística livre de contexto G_p denomina-se consistente se a linguagem gerada por G_p é estocástica, ou seja:*

$$\sum_{\chi \in L(G_p)} \phi(x) = 1. \tag{1.5}$$

No seguinte exemplo a meta é atribuir valores de probabilidade às sentenças de uma linguagem, desta maneira decidir quais sentenças são corretas/incorretas.

1.4.2 Exemplo de como calcular a probabilidade de uma sentença ω dada uma gramática G_p

Suponha a Gramática G_2 descrita na Figura 6, vamos atribuir um valor de probabilidade para cada uma das regras da gramática G_2 (Ver 10).

S	$\rightarrow$	$NP\ VP$	1,0
PP	$\rightarrow$	$P\ NP$	1,0
VP	$\rightarrow$	$V\ NP$	0,7
VP	$\rightarrow$	$VP\ PP$	0,3
P	$\rightarrow$	$with$	1,0
V	$\rightarrow$	saw	1,0
NP	$\rightarrow$	$NP\ PP$	0,4
NP	$\rightarrow$	$astronomers$	0,1
NP	$\rightarrow$	$ears$	0,18
NP	$\rightarrow$	saw	0,04
NP	$\rightarrow$	$stars$	0,18
NP	$\rightarrow$	$telescopes$	0,1

Figura 10 – Gramática probabilística livre de contexto na forma normal de Chomsky, onde cada regra tem associada um valor de probabilidade.

Onde os símbolos não terminais são S, NP, PP, VP, P e V. O símbolo inicial é S e os símbolos terminais são as palavras escritas em itálico, assim como antes.

Informalmente, a probabilidade associada a uma sentença χ é a soma das probabilidades associadas à cada uma de suas árvores sintáticas. Assim, no exemplo, a probabilidade da sentença $\omega =$ *astronomers saw start with ears* é dada pela probabilidade da árvore sintática t_1 na Figura 8 mais a probabilidade da árvore sintática t_2 da Figura 9, assumindo que ω não possui mais árvores de derivação.

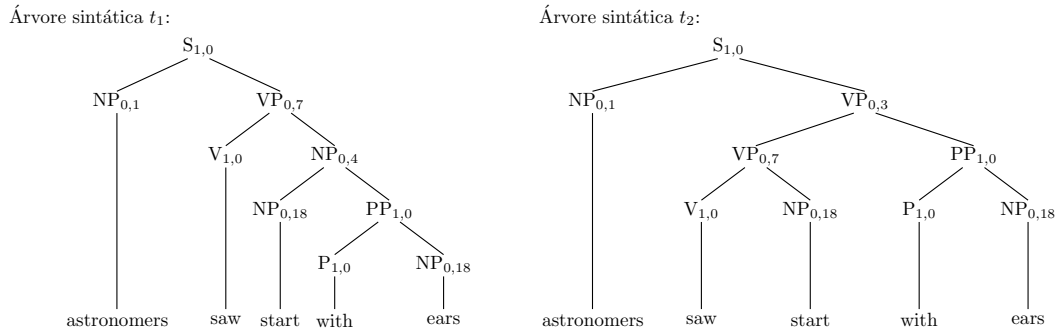

Figura 11 – Árvores sintáticas da sentença ω com valores de probabilidade associadas às regras.

Para determinar a probabilidade de uma árvore sintática, vamos supor que a gramática satisfaz o seguinte:

- *place invariance*: a probabilidade de uma sub-árvore não depende da sequência de cadeias que ele gera.
- *livre de contexto*: a probabilidade de uma sub-árvore não depende das sequências de cadeias que ela não gera,
- *ancestral-livre*: a probabilidade de uma sub-árvore não depende dos outros nós fora da sub-árvore.

Com estas suposições, podemos dizer que a probabilidade de uma árvore sintática é o produto das probabilidades associadas àquelas regras utilizadas para construir a árvore. Assim, com ajuda da Figura 11, calculamos as probabilidades das árvores sintáticas t_1 e t_2:

$$P(t_1) = 1,0 \times 0,1 \times 0,7 \times 1,0 \times 0,4 \times 0,18 \times 1,0 \times 1,0 \times 0,18 = 0,0009072 \quad (1.6)$$

$$P(t_2) = 1,0 \times 0,1 \times 0,3 \times 0,7 \times 1,0 \times 1,0 \times 0,18 \times 1,0 \times 0,18 = 0,0006804 \quad (1.7)$$

E portanto, estabelecemos que a probabilidade da sentença ω esta dada por:

$$P(\omega) = P(t_1) + P(t_2) = 0,0015876. \quad (1.8)$$

Observe que, neste exemplo foi assumido que as probabilidades das regras estavam dadas e o propósito era determinar a probabilidade de uma sentença.

Por outro lado, lembremos que a abordagem que estudaremos para estimar os parâmetros (as probabilidades das regras) do modelo probabilístico de uma linguagem, é o aprendizado da gramática, ou seja, as probabilidades são estimadas a partir de um corpus [MSM93].

1.4.3 Função de verossimilhança

Para o processo de aprendizagem da gramáticas probabilísticas livres de contexto, alguns pesquisadores propõem decompor o processo: na aprendizagem da gramática característica (as regras de derivação) e aprendizagem das probabilidades das regras [Vid94].

Nós vamos trabalhar com o problema de aprendizagem das probabilidades das regras, somente. Portanto vamos supor que a parte estrutural (símbolos e regras) da gramática está dada. Usando a teoria da inferência estatística usamos a técnica da máxima verossimilhança para estimar os parâmetros das probabilidades das regras. Assim, devemos primeiro determinar a função de verossimilhança a partir do corpus e depois escolher um método para otimizá-la.

Nas gramáticas probabilísticas livres de contexto a função de verossimilhança de uma amostra Ω de $L(G)$ é definida assim:

$$Pr(\Omega|G_p) = \prod_{\chi \in \Omega} Pr(\chi|G_p). \tag{1.9}$$

Note que, quando ordenamos as regras e definimos a probabilidade da i-ésima regra como uma variável $x_i = p(q_i)$, $i = 1, 2, \ldots |P|$, obtemos um polinômio de várias variáveis em $x \in \Re^{|P|}$.

No exemplo a seguir observamos como construir a função de verossimilhança para uma gramática dada:

1.4.4 Exemplo para calcular a função de verossimilhança

Seja $G = (N, \Sigma, P, S)$, onde $N = \{A, B, C, S\}$, $\Sigma = \{a, b\}$, S representa o símbolo inicial e P é o conjunto das regras assim definidas:

$$\begin{array}{lll} 1)\ q_1 = S \to AB & 4)\ q_4 = A \to a & 7)\ q_7 = C \to AB \\ 2)\ q_2 = S \to BC & 5)\ q_5 = B \to CC & 8)\ q_8 = C \to a. \\ 3)\ q_3 = A \to BA & 6)\ q_6 = B \to b & \end{array}$$

A cada regra associamos uma probabilidade:

$$\begin{array}{lll} x_1 = p(S \to AB) & x_4 = p(A \to a) & x_7 = p(C \to AB) \\ x_2 = p(S \to BC) & x_5 = p(B \to CC) & x_8 = p(C \to a). \\ x_3 = p(A \to BA) & x_6 = p(B \to b) & \end{array}$$

Seja $\Omega = \{baaba, baaa\} \subset L(G)$ uma amostra da linguagem L. Sejam $\chi = baaba$ e $\eta = baaa$. As árvores de derivação para as cadeias da amostra estão dadas pela Figura

12 e a Tabela 1 relaciona o número de vezes $Nr(q_i, d_j)$ que a regra q_i está sendo usada em cada derivação d_j.

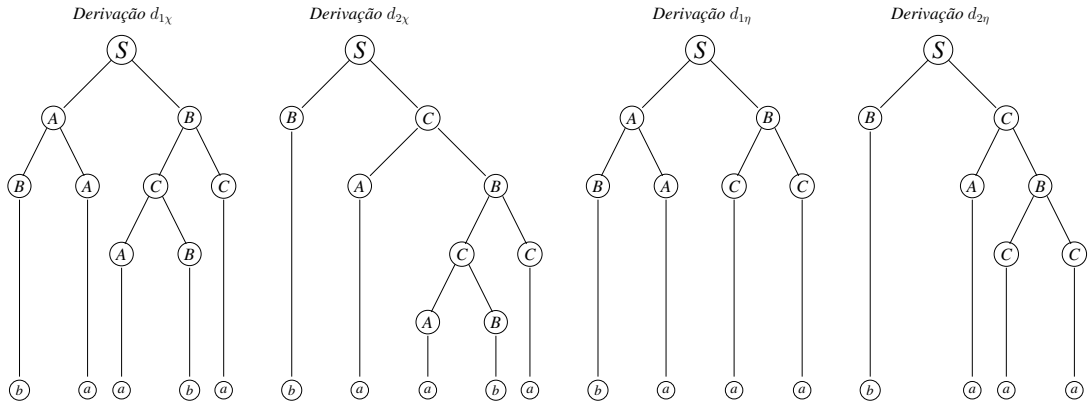

Figura 12 – Árvores de derivação dos elementos de Ω.

Regra	D_χ		D_η		Probabilidade associada
	$Nr(q_i, d_{1\chi})$	$Nr(q_i, d_{2\chi})$	$Nr(q_i, d_{1\eta})$	$Nr(q_i, d_{2\eta})$	
$q_1 = S \to AB$	1	0	1	0	x_1
$q_2 = S \to BC$	0	1	0	1	x_2
$q_3 = A \to BA$	1	0	1	0	x_3
$q_4 = A \to a$	2	2	1	1	x_4
$q_5 = B \to CC$	1	1	1	1	x_5
$q_6 = B \to b$	2	2	1	1	x_6
$q_7 = C \to AB$	1	2	0	1	x_7
$q_8 = C \to a$	1	1	2	2	x_8

Tabela 1 – Número de vezes que cada regra está sendo usada nas derivações.

Da Equação (1.3) temos que:

$$\begin{aligned}\Pr(\chi, d_{1\chi}|G_p) &= x_1x_3x_4^2x_5x_6^2x_7x_8 \qquad \Pr(\eta, d_{1\eta}|G_p) = x_1x_3x_4x_5x_6x_8^2\\ \Pr(\chi, d_{2\chi}|G_p) &= x_2x_4^2x_5x_6^2x_7^2x_8 \qquad \Pr(\eta, d_{2\eta}|G_p) = x_2x_4x_5x_6x_7x_8^2,\end{aligned}$$

agora, usando (1.4), obtemos que:

$$\begin{aligned}\Pr(\chi|G_p) &= \Pr(\chi, d_{1\chi}|G_p) + \Pr(\chi, d_{2\chi}|G_p) = x_1x_3x_4^2x_5x_6^2x_7x_8 + x_2x_4^2x_5x_6^2x_7^2x_8\\ \Pr(\eta|G_p) &= \Pr(\eta, d_{1\eta}|G_p) + \Pr(\eta, d_{2\eta}|G_p) = x_1x_3x_4x_5x_6x_8^2 + x_2x_4x_5x_6x_7x_8^2.\end{aligned}$$

Finalmente, da Equação (1.9) temos a função de verossimilhança da amostra Ω:

$$\begin{aligned}\Pr(\Omega|G_p) &= \Pr(\chi|G_p) * \Pr(\eta|G_p)\\ &= (x_1x_3x_4^2x_5x_6^2x_7x_8 + x_2x_4^2x_5x_6^2x_7^2x_8)(x_1x_3x_4x_5x_6x_8^2 + x_2x_4x_5x_6x_7x_8^2)\\ &= x_1^2x_3^2x_4^3x_5^2x_6^3x_7x_8^3 + x_1x_2x_3x_4^3x_5^2x_6^3x_7^2x_8^3\\ &\quad + x_1x_2x_3x_4^3x_5^2x_6^3x_7^2x_8^3 + x_2^2x_4^3x_5^2x_6^3x_7^3x_8^3.\end{aligned}$$

Assim, conforme visto anteriormente e lembrando a Definição 1.9, podemos concluir que a função $Pr(\Omega|G_p)$ corresponde a um polinômio em várias variáveis, e que o problema de estimação dos parâmetros é equivalente a otimizar um polinômio sujeito a restrições lineares canalizadas:

$$\begin{aligned}\text{maximizar} \quad & Pr(\Omega|G_p)\\ \text{sujeito a} \quad & \sum_{x_i \in \Psi_A} x_i = 1, \quad \forall\ \Psi_A : A \in \Sigma\\ & 0 \leqslant x_i \leqslant 1, \quad i = 1, ..., |P|,\end{aligned} \tag{1.10}$$

onde Ψ_A representa o conjunto de todas as probabilidades de regras cujo antecedente é A.

Observe que o Problema (1.10) é não linear com variáveis canalizadas, e que as características da restrição

$$\sum_{x \in \Psi_A} x_i = 1, \forall\ \Psi_A : A \in \Sigma, \tag{1.11}$$

já garantem que a restrição canalizada $x_i \leqslant 1$, $i = 1, ..., |P|$ seja satisfeita. Portanto, podemos reescrever o Problema (1.10) como:

$$\begin{aligned}\text{maximizar} \quad & Pr(\Omega|G_p)\\ \text{sujeito a} \quad & \sum_{x \in \Psi_A} x_i = 1, \quad \forall\ \Psi_A : A \in \Sigma\\ & x_i \geqslant 0, \qquad i = 1, ..., |P|.\end{aligned} \tag{1.12}$$

1.5 Conjunto de treino (*corpus*)

1.5.1 Introdução

Como já foi mencionado antes, nossa abordagem para o aprendizado da gramática é basicamente proporcionar alguns exemplos da linguagem, ou seja, a amostra, esta é extraída de um *corpus* da linguagem e a partir dela a gramática aprende como são construídas as sentenças válidas da linguagem que desejamos gerar ou modelar. Lembrando que o conceito de sentença válida está associado diretamente à existência de pelo menos uma árvore de derivação. Além disso, é necessário avaliar a eficiência dos algoritmos e estudar seus problemas numa aplicação real. É por isso que utilizamos uma parte do *corpus* histórico do Português *Tycho Brahe*.

A decisão de trabalhar com este *corpus* é por ser um *corpus* que da língua Portuguesa, e estar disponível para pesquisa livre.

1.6 *Corpus Tycho Brahe*

O *corpus* histórico do Português *Tycho Brahe* é um *corpus* eletrônico anotado, composto de textos em português escritos por autores nascidos entre 1380 e 1881.

```
((CP-EXL (CONJ Mas)
  (WNP (WD que) (Q muito))
  (, ,)
  (NP-VOC (NPR Senhor))
  (, ,)
  (CP-THT (C que)
    (IP-SUB (NP-SBJ (D-F-P as) (N-P vaidades))
      (ET-SP estejam)
      (PP (FP só)
        (P a@)
        (NP (D-P @os)
          (N-P pés)
        (PP (P de)
        (NP (PRO$-F Vossa) (NPR Majestade)))))))
  (, ,)
```

Figura 13 – Exemplo de uma árvore em formato de anotação marcado de brackets. *Corpus Tycho Brahe*.

Atualmente, 76 textos (3.303.196 palavras) estão disponíveis para pesquisa livre, com um sistema de anotação linguística em duas etapas: anotação morfológica (aplicada

em 44 textos, num total de 1.956.460 palavras); e anotação sintática (aplicada em 20 textos, num total de 877.247 palavras).

É de interesse neste trabalho, só trabalhar com aquela parte que está em anotação sintática, onde as árvores são representadas com uma *anotação marcado de brackets* dentro de um arquivo de texto. A Figura 13 mostra um exemplo de uma árvore, escrita nesta anotação, do *corpus Tycho Brahe*.

O *Tycho Brahe* foi desenvolvido como parte do projeto temático *Rhythmic patterns, parameter setting*, and *language change*, com financiamento da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e coordenação de Charlotte Galves. Apoio suplementar foi obtido junto ao Conselho Nacional de Pesquisa (CNPq), com o projeto *PRONEX Critical phenomena in probability and stochastic processes*.

2 Algoritmo *Inside-Outside*

2.1 Introdução

A ideia global do trabalho é treinar uma GPLC. Assim como no final do capítulo anterior, supomos que a parte estrutural da gramática é dada, ou seja, os seguintes conjuntos são conhecidos: o conjunto finito de símbolos não terminais N, o conjunto finito de símbolos terminais Σ, o conjunto finito de regras P e o símbolo inicial S. Treinar uma gramática pode ser entendido como o processo de determinar as probabilidades ótimas para cada uma das regras da gramática a partir de dados de treinamento. Surge então o algoritmo *Inside-Outside* que treina os parâmetros de uma GPLC a partir de um *corpus* de treinamento e das árvores sintáticas de treinamento. A suposição básica é que uma boa gramática é aquela que consegue gerar as sentenças contidas no *corpus*, portanto procuramos aquela gramática que maximiza a função de verossimilhança dos dados pertencentes ao *corpus*, assim como no Exemplo 1.4.4. Neste capítulo apresentamos um dos métodos clássicos para o treino das GPLC: o algoritmo *Inside-Outside*, este está baseado no teorema de transformações crescentes [BE66] e nos algoritmo *Inside* e *Outside*, descritos na Seção 2.3 e na Seção 2.4.

2.2 Transformações crescentes

Vamos começar falando do teorema das transformações crescentes que, como foi dito, é a base principal do algoritmo *Inside-Outside*.

Teorema 2.2.1 (Teorema das transformações crescentes)**.** *Seja $F(x) = (\{x_{ij}\})$ um polinômio homogêneo com coeficientes não negativos e de grau d em suas variáveis $\{x_{ij}\}$. Seja $x = \{x_{ij}\}$ um ponto do domínio $D = \{x_{ij} \geqslant 0 \Big| \sum_{j=1}^{q_i} x_{ij} = 1, i = 1, \ldots, p, j = 1, \ldots, q_i\}$. Para um dado $x = \{x_{ij}\} \in D$, o ponto $J(x) = J(\{x_{ij}\})$ em D, está definido em cada i, j como:*

$$J(x)_{ij} = \frac{(x_{ij} \frac{\partial F}{\partial x_{ij}}\Big|_{(x)})}{\sum_{j=1}^{q_i} x_{ij} \frac{\partial F}{\partial x_{ij}}\Big|_{(x)}}. \tag{2.1}$$

Então $F(J(x)) > F(x)$, exceto se $J(x) = x$.

Se $F(x)$ possui um valor máximo finito, então é possível mostrar que toda sequência de pontos gerados pela transformação dada na Equação (2.1), converge para um máximo local de $F(x)$. Este resultado e a demostração deste teorema podem ser vistos em [BE66].

Usando o Teorema 2.2.1 em um polinômio F cujo domínio seja D e usando um algoritmo de máxima subida, é possível obter um máximo local de F.

2.3 Algoritmo *Inside*

O algoritmo *Inside* é usado em problemas de análise sintática probabilística de sentenças, ou seja, dada uma χ determina se $Pr(\chi|G_p) > 0$. Para resolver este problema é suficiente encontrar uma derivação d_χ cuja probabilidade seja maior que zero e que usando d_χ seja possível derivar a sentença χ a partir do símbolo inicial S, isto é $S \stackrel{*}{\Rightarrow} \chi$.

Este algoritmo determina se $Pr(\chi|G_p) > 0$ calculando a probabilidade da sentença a partir de todas as possíveis derivações. Para isso, define: $e(A < i, j >) = Pr(A \stackrel{*}{\Rightarrow} \chi_i, \ldots, \chi_j|G_p)$, que é a probabilidade de que a sub-cadeia $\chi_i, \ldots, \chi_j$ seja gerada a partir de $A \in N$. Para calcular $e(A < i, j >)$ são usadas as fórmulas a seguir:

$$
\begin{aligned}
e(A < i, i >) = \quad & p(A \longrightarrow \chi_i), \quad 1 \leqslant i \leqslant |\chi| \\
e(A < i, j >) = \quad & \sum_{B,C \in N} p(A \longrightarrow BC) \sum_{k=i}^{j-1} e(B < i, k >) e(C < k+1, j >), \\
& \qquad\qquad\qquad\qquad\qquad\qquad 1 \leqslant i < j \leqslant |\chi|,
\end{aligned}
$$

onde $A \longrightarrow \chi_i$, $A \longrightarrow BC \in P$.

Finalmente, $Pr(\chi|G_p) = e(S < 1, |\chi| >)$. O algoritmo tem complexidade $O(|\chi|^3|P|)$.

2.4 Algoritmo Outside

Da forma semelhante ao algoritmo *Inside*, este algoritmo determina se una sentença χ pertence a uma linguagem, calculando $Pr(\chi|G_p) > 0$ a partir de todas as possíveis derivações.

Define-se $f(A < i, j >) = Pr(S \stackrel{*}{\Rightarrow} \chi_1 \ldots \chi_{i-1} A \chi_{j+1} \ldots \chi_{|\chi|}|G_p)$, como a probabilidade de que a partir do símbolo inicial seja gerada a sub-cadeia $\chi_1 \ldots \chi_{i-1}$, depois o símbolo não terminal A e em seguida a sub-cadeia $\chi_{j+1} \ldots \chi_{|\chi|}$. Note que o símbolo não terminal A tem que gerar a sub-cadeia $\chi_i \ldots \chi_j$. Para calcular $f(A < i, j >)$, são usadas a

seguintes fórmulas:

$$\forall A \in N,$$

$$f(A < 1, |\chi| >) = \begin{cases} 1 & \text{se } A = S \\ 0 & \text{se } A \neq S \end{cases}$$

$$f(A < i, j >) = \sum_{B,C \in N} \left(p(B \longrightarrow CA) \sum_{k=1}^{i-1} f(B < k, j >) e(C < k, i-1 >) + p(B \longrightarrow AC) \sum_{k=j+1}^{|\chi|} f(B < i, k >) e(C < j+1, k >) \right)$$

$$1 \leqslant i < j \leqslant |\chi|.$$

Logo, $Pr(\chi|G_p) = \sum_{A \in N} f(A < i, i >) p(A \longrightarrow \chi_i)$, $1 \leqslant i \leqslant |\chi|$. O cálculo das fórmulas anteriores tem complexidade $O(|\chi|^3 |P|)$.

2.5 Algoritmo *Inside-Outside*

O algoritmo *Inside-Outside* maximiza a função de verossimilhança definida na Equação (1.9). Vamos iniciar o desenvolvimento do algoritmo notando que as probabilidades das regras de uma gramática probabilística livre de contexto, são pontos do domínio D do Teorema 2.2.1. Para uma gramática probabilística livre de contexto G_p dada (não necessariamente na forma normal de Chomsky), temos:

$$p(A_i \longrightarrow \alpha_j) = \{x_{ij}\}, \quad i = 1, \ldots, |N| \quad j = 1, \ldots, |\Gamma_A|,$$

onde Γ_{A_i} representa o conjunto de regras da gramática G_p cujo antecedente é A_i. Portanto, a função de verossimilhança que está definida em termos destas probabilidades pode ser maximizada usando o Teorema 2.2.1. É possível definir uma transformação para cada $(A \longrightarrow \alpha) \in P$ como:

$$\overline{p}(A \longrightarrow \alpha) = \frac{p(A \longrightarrow \alpha) \left(\frac{\partial ln Pr(\Omega|G_p)}{\partial p(A \longrightarrow \alpha)} \right)}{\sum_{(A \longrightarrow \alpha) \in \Gamma_A} p(A \longrightarrow \alpha) \left(\frac{\partial ln Pr(\Omega|G_p)}{\partial p(A \longrightarrow \alpha)} \right)}. \tag{2.2}$$

Por outro lado, note que dada uma GPLC o logaritmo da função de verossimilhança de uma amostra Ω é definida como:

$$ln Pr(\Omega|G_p) = ln \prod_{x \in \Omega} Pr(\chi|G_p).$$

Assim, desenvolvendo algebricamente a Equação (2.2) temos que para cada $(A \longrightarrow \alpha) \in P$:

$$\overline{p}(A \longrightarrow \alpha) = \frac{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi|G_p)} p(A \longrightarrow \alpha) \left(\frac{\partial Pr(\chi|G_p)}{\partial p(A \longrightarrow \alpha)} \right)}{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi|G_p)} \sum_{(A \longrightarrow \alpha) \in \Gamma_A} p(A \longrightarrow \alpha) \left(\frac{\partial Pr(\chi|G_p)}{\partial Pr(A \longrightarrow \alpha)} \right)}. \tag{2.3}$$

Vamos desenvolver primeiramente o numerador da Equação (2.3) fazendo uso das equações (1.3) e (1.4), obtendo:

$$\begin{aligned} p(A \longrightarrow \alpha) \left(\frac{\partial Pr(\chi|G_p)}{\partial Pr(A \longrightarrow \alpha)} \right) &= p(A \longrightarrow \alpha) \sum_{d_\chi \in D_\chi} \left(\frac{\partial Pr(\chi, d_\chi|G_p)}{\partial Pr(A \longrightarrow \alpha)} \right) \\ &= \sum_{d_\chi \in D_\chi} Nr(A \longrightarrow \alpha, d_\chi) Pr(\chi, d_\chi|G_p). \end{aligned} \tag{2.4}$$

Para resolver o denominador de (2.3) usamos a expressão (2.4) e como o número de vezes que o símbolo não terminal A faz parte de uma regra de derivação d_χ está dado por

$$Nr(A, d_\chi) = \sum_{(A \longrightarrow \alpha) \in \Gamma_A} Nr(A \longrightarrow \alpha, d_\chi),$$

portanto,

$$\begin{aligned} \sum_{(A \longrightarrow \alpha) \in \Gamma_A} p(A \longrightarrow \alpha) \left(\frac{\partial Pr(\chi|G_p)}{\partial Pr(A \longrightarrow \alpha)} \right) &= \sum_{(A \longrightarrow \alpha) \in \Gamma_A} \sum_{d_\chi \in D_\chi} Nr(A \longrightarrow \alpha, d_\chi) Pr(\chi, d_\chi|G_p) \\ &= \sum_{d_\chi \in D_\chi} \sum_{(A \longrightarrow \alpha) \in \Gamma_A} Nr(A \longrightarrow \alpha, d_\chi) Pr(\chi, d_\chi|G_p) \\ &= \sum_{d_\chi \in D_\chi} Nr(A, d_\chi) Pr(\chi, d_\chi|G_p). \end{aligned}$$

Finalmente, a Expressão (2.3) fica:

$$\overline{p}(A \longrightarrow \alpha) = \frac{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi|G_p)} \sum_{d_\chi \in D_\chi} Nr(A \longrightarrow \alpha, d_\chi) Pr(\chi, d_\chi|G_p)}{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi|G_p)} \sum_{d_\chi \in D_\chi} Nr(A, d_\chi) Pr(\chi, d_\chi|G_p)}, \tag{2.5}$$

para cada $(A \longrightarrow \alpha) \in P$.

O algoritmo *Inside-Outside* aplica iterativamente essa transformação até maximizar localmente a função de verossimilhança. O ótimo atingido depende dos valores iniciais. A convergência do algoritmo está garantida pelo Teorema 2.2.1 e acontece quando as probabilidades da gramática não mudam de iteração para iteração.

Para fazer os cálculos de uma forma eficiente, vamos considerar uma gramática G na forma normal de Chomsky e $A \longrightarrow BC$ uma regra de G, onde $A, B, C \in N$. Vamos supor que d_χ é uma derivação da cadeia χ e que a regra $A \longrightarrow BC$ pertence à sequência das regras da derivação d_χ (Ver Figure 14).

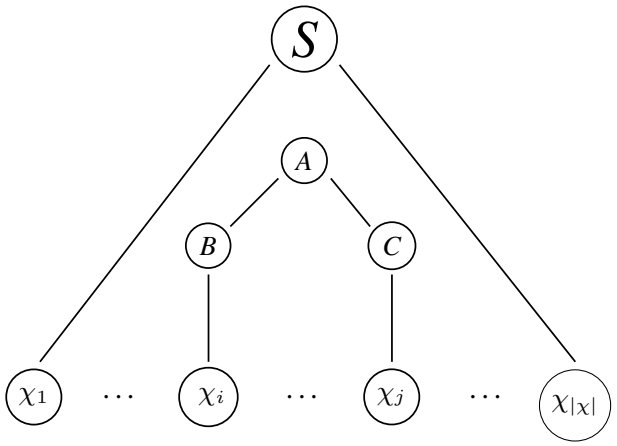

Figura 14 – Árvore de derivação d_χ.

Assim, a partir do símbolo inicial S a cadeia $\chi_1, \ldots, \chi_{i-1} A \chi_{j+1}, \ldots, \chi_{|\chi|}$ é gerada, o símbolo não terminal A gera BC, deste modo B tem que gerar $\chi_i, \ldots, \chi_k$ e C deve gerar $\chi_{k+1}, \ldots, \chi_j$, para algum $i < k < j$.

Somando a probabilidade de todas as derivações em que a regra $A \longrightarrow BC$ está limitada para valores de i e j obtemos:

$$\begin{aligned} Pr(S \overset{*}{\Rightarrow} \chi_1 \ldots \chi_{i-1} A \chi_{j+1} \ldots \chi_{|\chi|}) p(A \longrightarrow BC) \times \\ Pr(B \overset{*}{\Rightarrow} \chi_i \ldots \chi_k | G_p) Pr(C \overset{*}{\Rightarrow} \chi_{k+1} \ldots \chi_j | G_p) \quad &= \quad f(A < i, j >) p(A \longrightarrow BC) \times \\ & \quad e(B < i, k >) e(C < k+1, j >). \end{aligned} \tag{2.6}$$

Considerando todos os valores possíveis dos limites (i, j e k) e raciocinando da mesma forma para o denominador de (2.5) e para as regras do tipo ($A \longrightarrow a$) com $A \in N$ e $a \in \Sigma$, obtemos:

$$\overline{p}(A \longrightarrow a) = \frac{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi | G_p)} \sum_{i=1}^{|\chi|} f(A < i, i >) p(A \longrightarrow \chi_i)}{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi | G_p)} \sum_{i=1}^{|\chi|} \sum_{j=i}^{|\chi|} f(A < i, j >) e(A < i, j >)}. \tag{2.7}$$

E, para toda regra da forma $(A \longrightarrow BC) \in P$:

$$\overline{p}(A \longrightarrow BC) =$$

$$\frac{\sum_{\chi \in \Omega} \frac{P(A \longrightarrow BC)}{Pr(\chi|G_p)} \sum_{1 \leqslant i \leqslant k < j \leqslant |\chi|} f(A < i, j >) e(B < i, k >) e(C < k+1, j >)}{\sum_{\chi \in \Omega} \frac{1}{Pr(\chi|G_p)} \sum_{i=1}^{|\chi|} \sum_{j=i}^{|\chi|} f(A < i, j >) e(A < i, j >)}. \tag{2.8}$$

Observamos que estas expressões necessitam os mesmos cálculos que os algoritmos *Inside* e *Outside* utilizam.

Assim, para alguns valores iniciais das probabilidades das regras da gramática probabilística livre de contexto, aplicamos as fórmulas (2.8) e (2.7) repetidamente, até que, os valores de duas iterações consecutivas alcancem uma tolerância determinada. Os valores iniciais das regras condicionam o ótimo local atingido [BE66].

Para cada regra de derivação, o algoritmo *Inside-Outside* necessita usar a estratégia *Inside*, depois a estratégia *Outside* e finalmente utiliza a transformação (2.1) até convergir. A complexidade em cada iteração é $O(3|\Omega|l_m^3|P|)$, onde $l_m = max_{x \in \Omega}|x|$ [Pei99]. Em outras palavras, para cada sentença do *corpus*, cada iteração usada no seu treino é de $O(m^3 n^3)$, onde m é o comprimento da sentença e n o número de símbolos não terminais de G.

2.6 Qualidade do modelo obtido

Uma vez encontrada uma solução, ou seja, estimadas as probabilidades das regras da gramática probabilística livre de contexto, é preciso verificar a qualidade do modelo probabilístico obtido. A perplexidade é uma medida usada para medir e comparar os modelos obtidos independentemente do método usado. Esta medida é calculada usando um conjunto de dados que não foram usados no processo de aprendizagem, vamos denominá-lo conjunto de teste Ts. Quando o modelo é uma gramática probabilística livre de contexto, a perplexidade por palavra (PP) é definida usando a teoria da entropia [Rou96]:

$$PP(Ts, G_p) = exp\left\{ -\frac{\sum_{\chi \in Ts} ln(Pr(\chi|G_p))}{\sum_{\chi \in Ts} |\chi|} \right\}. \tag{2.9}$$

Intuitivamente o conceito de perplexidade é uma medida do tamanho do conjunto de palavras a partir do qual a próxima palavra é escolhida levando em consideração as anteriores. Quanto mais próximo de zero o valor da perplexidade, melhor a qualidade do modelo [Pei99].

Com isto, finalizamos as generalidades sobre o método *Inside-Outside*, lembrando que não é o método a ser usado neste trabalho para maximizar os parâmetros de uma GPLC. Mas se usarmos a Equação (2.9) para determinar as perplexidades dos modelos obtidos usando o método de pontos interiores.

3 Algoritmo Cocke-Younger-Kassami (CYK)

3.1 Introdução

Como visto no Capítulo 1, as gramáticas probabilísticas livres de contexto possuem um importante componente chamado de *parser*[1] ou analisador sintático, este é indispensável para obter a função de verossimilhança da amostra. Neste capítulo apresentamos um método eficiente para abordar um dos problemas de análise sintática: dada uma sentença χ e uma GPLC G_p, estabelecer se $\chi \in L(G)$. A solução para este problema consiste em encontrar uma sequência de derivações que permitam derivar χ a partir do símbolo inicial de G_p, fazendo uso das regras da gramática. Normalmente é necessário determinar todas as possíveis árvores sintáticas da sentença χ de acordo com uma gramática G.

Para resolver os problemas da análise sintática, vamos apresentar o bem sucedido algoritmo Cocke-Younger-Kasami que com um tempo cúbico reconhece uma linguagem L dada. Ou seja, dada uma cadeia χ do vocabulário da linguagem L, o algoritmo CYK é capaz de aceitar o rejeitar a cadeia χ como uma sentença válida da linguagem L [You66] [Kas65]. Já para o segundo problema e baseado no algoritmo CYK, podemos encontrar todas as possíveis árvores sintáticas da sentença χ numa gramática G dada.

Para este trabalho e uma vez estudado o algoritmo CYK, desenvolvemos uma variação do mesmo para determinar o valor da função de verossimilhança (ver Exemplo 1.4.4) num ponto x, isto porque baseados no método CYK podemos calcular as probabilidades de todas as possíveis árvores de derivação de uma sentença χ, dada a gramática G.

3.2 O algoritmo CYK

Seja $G = (N, \Sigma, P, S)$ uma gramática que vamos supor na forma normal de Chomsky e seja χ uma cadeia de símbolos de Σ e com comprimento $n \geqslant 1$. Determine para cada par i, j e para cada símbolo não terminal $A \in N$, uma derivação qualquer $A \overset{*}{\Rightarrow} \chi_{i,j}$, onde $\chi_{i,j}$ é uma sub-cadeia de χ de comprimento j e que inicia na posição i, com $i, j = 1, ..., |P|$.

Usando indução em j e com $j = 1$, então $A \overset{*}{\Rightarrow} \chi_{i,j}$ se, e somente se, $A \to \chi_{i,j} \in P$. Continuando para valores de $n > 1$; $A \overset{*}{\Rightarrow} \chi_{i,j}$ se, e somente se, existe alguma regra $A \to BC \in P$ e algum inteiro k com $1 \leqslant k < j$, tal que B gera os primeiros k símbolos da sub-cadeia χ_{ij} e C gera os últimos $j - k$ símbolos de χ_{ij}, ou seja $B \overset{*}{\Rightarrow} \chi_{i,k}$ e $C \overset{*}{\Rightarrow} \chi_{i+k,j-k}$.

[1] O *parser* é o gerador das árvores sintáticas.

Como $k < j$ e $j - k < j$, pela indução sabemos que existem as duas derivações $B \overset{*}{\Rightarrow} \chi_{i,k}$ e $C \overset{*}{\Rightarrow} \chi_{i+k,j-k}$, portanto podemos estabelecer que existe uma derivação $A \to \chi_{i,j}$. Assim, quando $j = n$ podemos determinar que de fato existe $S \overset{*}{\Rightarrow} \chi_{1,n}$. Mas $\chi_{1,n} = \chi$, pelo qual $\chi \in L(G)$ se, e somente se, $S \overset{*}{\Rightarrow} \chi_{1,n}$.

Antes de escrever formalmente o algoritmo vamos definir N_{ij} como o conjunto de símbolos não terminais A tal que $A \overset{*}{\Rightarrow} \chi_{ij}$. Note que podemos supor que $1 \leqslant i \leqslant n-j+1$ pois não existem cadeias de comprimento maior que $n - i + 1$ e que começam na posição i. O algoritmo CYK é descrito a seguir e imediatamente depois apresentamos o Exemplo 3.2.1 onde cada conjunto $N_{i,j}$ está sendo representado pela caixa (i, j) da Tabela 2 e que será preenchida usando o algoritmo CYK [HMU01].

Método 1 – Algoritmo Cocke-Younger-Kasami

entrada : Uma cadeia χ de elementos do conjunto Σ.
salida : Aceita ou rejeita χ como sentença da linguagem L(G)

para $i \leftarrow 1$ ***até*** n **faça**
  $N_{i,1}$:= $\{A \mid A \to a$ é uma regra de produção e a é o i-ésimo símbolo de χ é $a\}$;
fim
para $j \leftarrow 2$ ***até*** n **faça**
  para $i \leftarrow 1$ ***até*** $n - j + 1$ **faça**
    $N_{i,1}$:= $\varnothing$ **para** $i \leftarrow 1$ ***até*** $j - 1$ **faça**
      $N_{i,1}$:= $N_{i,1} \cup \{A \mid A \to BC$ é uma regra de produção, $B \in N_{i,k}$ e $C \in N_{i+k,j-k}\}$
    fim
  fim
fim

3.2.1 Exemplo

Considere uma gramática livre de contexto G em FNC:

$$\begin{aligned} S &\to AB \mid BC \\ A &\to BA \mid a \\ B &\to CC \mid b \\ C &\to AB \mid a \end{aligned}$$

e seja $\chi = baaba$ a cadeia de entrada para o Método 1. A primeira linha da Tabela 2 é preenchida no primeiro *loop*: "para" na linha 1, que corresponde ao conjuntos $N_{i,1}$, com $1 \leqslant i \leqslant n$ e para os quais $N_{1,1} = N_{4,1} = \{B\}$ pois B é o único símbolo não terminal que deriva em b. De igual maneira notamos que $N_{2,1} = N_{3,1} = N_{5,1} = \{A, C\}$, pois A e C são os únicos que derivam em a.

Para determinar os elementos de $N_{i,j}$ com $j > 1$, realizamos o *loop*: "para" da linha 4 no Método 1. Neste fragmento do método analisamos em pares elementos

dos conjuntos $N_{i,k}$ e $N_{i+k,j-k}$ para $k = 1, 2...j-1$. Adicionamos o símbolo não inicial A em $N_{i,j}$ sempre que $A \to BC \in P$, com $B \in N_{i,k}$ e $C \in N_{i+k,j-k}$. No exemplo, para adicionar elementos ao conjunto $N_{2,3}$: primeiro tomamos os pares $(N_{2,1},N_{3,2}) = \{(A,S),(A,C),(C,S),(C,C)\}$ e procuramos coincidência de cada par no lado direito das regras de derivação, naqueles que há coincidência, o símbolo do lado esquerdo é acrescentado no conjunto $N_{2,3}$. De forma análoga fazemos para o par $(N_{2,2},N_{4,1})$. Com isso, terminamos de adicionar todos os elementos possíveis em $N_{2,3}$.

Se, finalizado o Método 1, o conjunto $N_{1,n}$ contém o símbolo inicial S, dizemos que existe um conjunto de regras em P tal que $S \overset{*}{\Rightarrow} \chi$, ou seja, que χ é uma sentença da linguagem $L(G)$.

		b	a	a	b	a
				$i \to$		
		1	2	3	4	5
	1	B	A, C	A, C	B	A, C
	2	S, A	B	S, C	S, A	
$j \downarrow$	3	$\varnothing$	B	B		
	4	$\varnothing$	S, A, C			
	5	S, A, C				

Tabela 2 – Tabela dos elementos em N_{ij}.

		b	a	a	b	a
				$i \to$		
		1	2	3	4	5
	1	B	A, C	A, C	B	A, C
	2	A	B	C	A	
$j \downarrow$	3	$\varnothing$	B	B		
	4	$\varnothing$	A, C			
	5	S				

Tabela 3 – Tabela dos elementos em N_{ij} atualizada.

Assim, utilizando o Método 1 resolvemos a primeira questão apresentada no início deste capítulo. A segunda questão é resolvida com uma extensão do Método 1. Observe os conjuntos $N_{i,j}$, mais detalhadamente nas entradas (i,j) da Tabela 2, a partir deles podemos extrair a estrutura gramatical da sentença: primeiramente eliminamos o símbolo S de todos os conjuntos $N_{i,j}$, exceto para $N_{1,5}$, dado que as regras de derivação de nossa gramática não contém o símbolo S no lado direito delas. Adicionalmente, tiramos todos os elementos de $N_{1,5}$ exceto o símbolo S pois as sentenças pertencentes à linguagem são aquelas que derivam do símbolo inicial S. Podemos ver na Tabela 3 como essas mudanças alteram a Tabela 2. Agora, analisando as entradas $\{(1,1),(2,4)\}$ junto com as regras de derivação podemos atualizar a entrada $(2,4) = \{C\}$. Note que isto, é uma

análise inversa ao feito no Método 1 e que de forma recursiva determinamos que a sentença $\chi = baaba$ possui duas árvores de derivação, as quais estão descritas na Figura 15.

Baseados no Método 1 e lembrando que a função de verossimilhança é um polinômio obtido a partir de uma amostra Ω de uma linguagem L, vamos apresentar uma variação do Método 1 que foi desenvolvida para obter a função de verossimilhança.

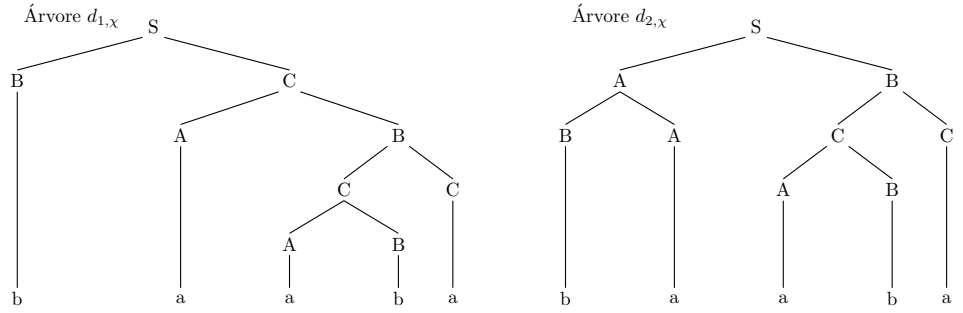

Figura 15 – Árvores sintáticas da sentença χ.

3.3 Variação do algoritmo CYK para calcular a função de verossimilhança num ponto x

Vamos supor que queremos modelar e/ou gerar uma linguagem L a partir de um *corpus*.

Dado um *corpus* que contém a anotação sintática de sentenças pertencentes à linguagem que queremos modelar, podemos extrair os seguintes dados:

- a amostra $\Omega = \{\chi, \eta\}$, vamos supor que contém duas sentenças da linguagem $L(G)$, onde $\chi = aac$, $\eta = deab$,
- os símbolos não terminais da linguagem $N = \{S, B, C\}$,
- os símbolos terminais $\Sigma = \{a, b, c, d, e\}$,
- dado que não temos informação explícita acerca do conjunto P, temos que considerar todas as possíveis regras que possam ser construídas a partir dos conjuntos N e Σ. Antes de prosseguir, vamos supor que $G = (N, \Sigma, P, S)$ é uma gramática livre de contexto que está na forma normal de Chomsky. Portanto, toda derivação pertencente a P é da forma:

$$\begin{aligned} &A \to BC, && \text{onde } A, B, C \in N, \\ &A \to a, && \text{onde } A \in N \text{ e } a \in \Sigma. \end{aligned}$$

Com essa consideração, obtemos um conjunto de 27 regras de derivação e usando uma função, atribuímos a cada regra um valor de probabilidade x_i, com $1 \leqslant i \leqslant 27$; apresentadas na Tabela 4.

x_1	S	$\rightarrow$	BB	x_9	C	$\rightarrow$	SS	x_{17}	S	$\rightarrow$	e	x_{25}	C	$\rightarrow$	c
x_2	S	$\rightarrow$	BC	x_{10}	C	$\rightarrow$	SB	x_{18}	B	$\rightarrow$	a	x_{26}	C	$\rightarrow$	d
x_3	S	$\rightarrow$	CB	x_{11}	C	$\rightarrow$	BS	x_{19}	B	$\rightarrow$	b	x_{27}	C	$\rightarrow$	e
x_4	S	$\rightarrow$	CC	x_{12}	C	$\rightarrow$	BB	x_{20}	B	$\rightarrow$	c				
x_5	B	$\rightarrow$	SS	x_{13}	S	$\rightarrow$	a	x_{21}	B	$\rightarrow$	d				
x_6	B	$\rightarrow$	SC	x_{14}	S	$\rightarrow$	b	x_{22}	B	$\rightarrow$	e				
x_7	B	$\rightarrow$	CS	x_{15}	S	$\rightarrow$	c	x_{23}	C	$\rightarrow$	a				
x_8	B	$\rightarrow$	CC	x_{16}	S	$\rightarrow$	d	x_{24}	C	$\rightarrow$	b				

Tabela 4 – Regras de derivação associadas à um valor de probabilidade x_i, com $1 \leqslant i \leqslant 27$.

Observe que o lado esquerdo não compartilha elementos iguais com o lado direito, em nenhuma das regras. Isto porque devemos evitar regras que gerem ciclos [HMU01].

Com isso, temos definido uma gramática livre de contexto probabilística $G_p = (N, \Sigma, P, S)$, e dado que, χ e η são duas sentenças válidas da linguagem L podemos garantir que existe pelo menos uma sequência de regras de derivação em P tal que $S \overset{*}{\Rightarrow} \chi$ e $S \overset{*}{\Rightarrow} \eta$. Lembrando a Equação (1.3), junto com a Equação (1.4) e a Equação (1.9), notamos que é preciso encontrar todas as derivações possíveis de cada uma das sentenças. Vamos dividir o processo em dois passos:

P1. Dada uma sentença χ de comprimento $\|\chi\| = n$, determinar a quantidade de estruturas possíveis de árvores sintáticas. Por exemplo, para uma sentença χ de comprimento $n = 3$ as duas estruturas possíveis estão na Figura 16.

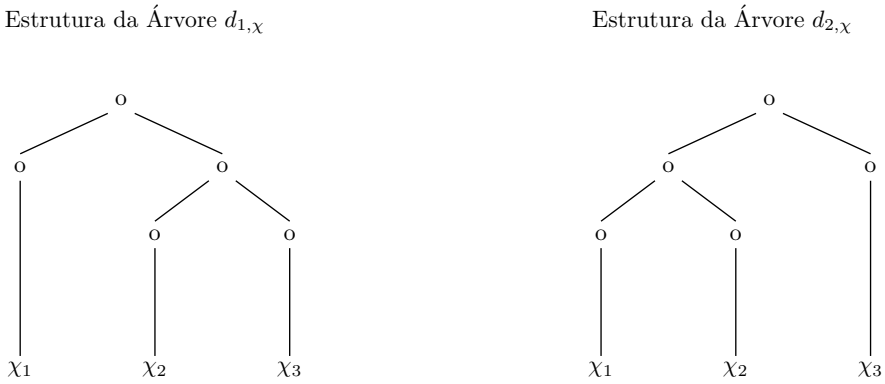

Figura 16 – Estrutura de árvores possíveis da sentença χ com $n = 3$.

Para uma sentença χ de comprimento $n = 4$, temos cinco estruturas possíveis apresentadas na Figura 17

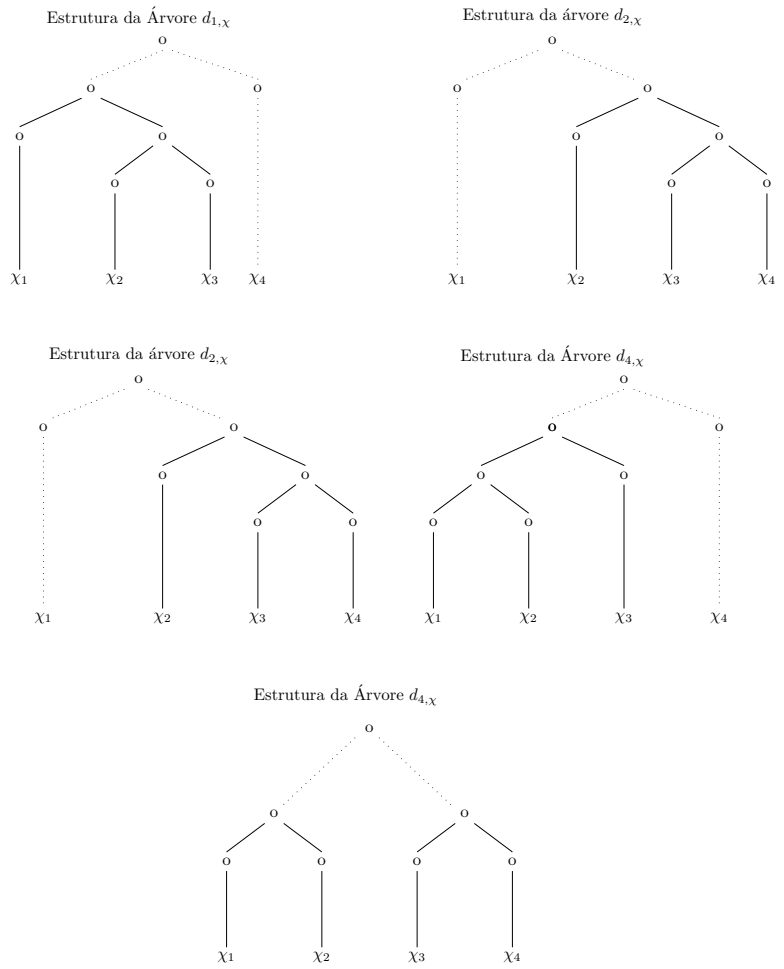

Figura 17 – Estrutura de árvores possíveis da sentença χ com $n = 4$.

Continuando com uma análise semelhante à realizada, obtemos uma fórmula que determina, dado o valor de n, o número de estruturas possíveis:

Proposição 3.3.1. *Seja χ, tal que $\|\chi\| = n$, então o número de estruturas possíveis OP_n é dado por*

$$OP_n = \sum_{j=1}^{n-1} (OP_j \times OP_{n-j}). \tag{3.1}$$

Continuando com a análise, elaboramos um algoritmo que tem como dado de entrada o comprimento da sentença n e como saída todas as possíveis estruturas de árvores de qualquer sentença de comprimento n. Estas estruturas são entregues em formato matricial da seguinte forma: por exemplo, no caso de $n = 3$ o algoritmo retorna a matriz D (ver Figura 18), onde cada linha da matriz D representa uma estrutura.

$$D = \begin{array}{|c|c|c|c|c|} \hline 4 & 1 & 5 & 3 & 6 \\ \hline 4 & 2 & 1 & 3 & 6 \\ \hline \end{array}$$

Figura 18 – Matriz D.

Para o desenvolvimento da variação do algoritmo CYK, é importante notar que a estrutura armazenada nas linhas da matriz D pode ser representada numa estrutura matricial, e que na sua vez é naturalmente representada no formato de árvore de derivação. Estas ideias estão resumidas na Figura 19.

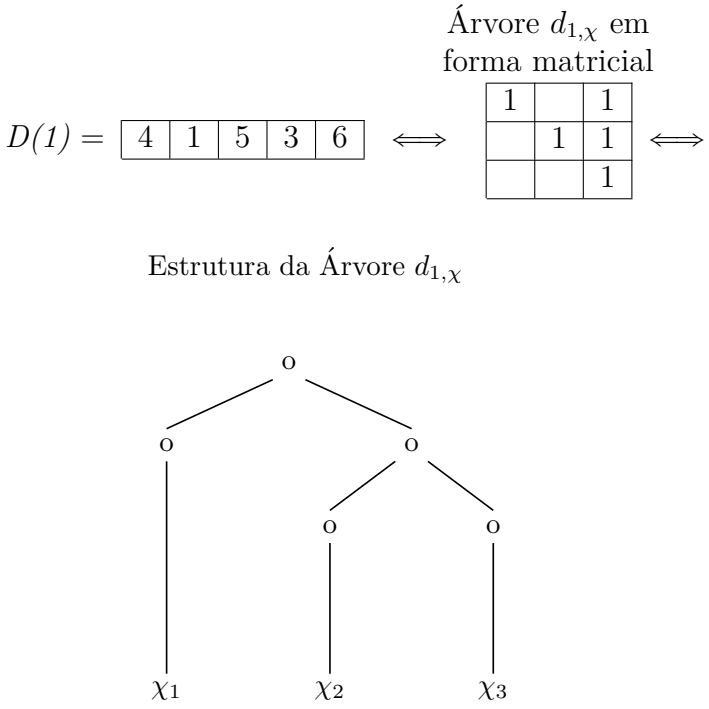

Figura 19 – Representação da estrutura da árvore sintática em forma matricial como no algoritmo CYK e forma vetorial como no caso do algoritmo CYK modificado.

P2. Uma vez obtidas todas as estruturas possíveis para uma sentença χ, com $\|\chi\| = n$, vamos construir todas as árvores sintáticas associadas a cada estrutura, por exemplo: a estrutura armazenada na primeira linha da matriz D do exemplo anterior, pode ser representada em formato de tabela, assim como no Método 1 (ver Tabela 5),

	χ_1	χ_2	χ_3
	$i \rightarrow$		
$j \downarrow$	o	o	o
	$\varnothing$	o	
	o		

Tabela 5 – Tabela formato algoritmo CYK.

onde as caixas diferentes de vazio têm que ser preenchidas com símbolos não terminais, assim como no Método CYK. Note que para $\chi = aac$ a Tabela 5 pode ser facilmente atualizada para a Tabela 6 da seguinte forma:

		a	a	c
			$i \rightarrow$	
		1	2	3
	1	S, B, C	S, B, C	S, B, C
$j \downarrow$	2	$\varnothing$	S, B, C	
	3	S		

Tabela 6 – Tabela atualizada para $\chi = aac$.

Vamos inicialmente considerar o símbolo inicial S na entrada (2,2) da Tabela 6, e notamos que existem quatro regras de derivação que possuem o símbolo S no lado esquerdo, a saber: 1, 2, 3 e 4 da Tabela 4. Criamos então o vetor:

$$v_{S_{2,2}}^T = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}, \tag{3.2}$$

onde x_i, $i = 1, ..., 4$ são os valores de probabilidade associados às regras que possuem no lado esquerdo o símbolo S.

A partir do vetor (3.2) obtemos o seguinte vetor

$$v_{S_{2,1}}^T = \begin{pmatrix} x_{18} & x_{18} & x_{23} & x_{23} \end{pmatrix}, \tag{3.3}$$

pois o primeiro símbolo não terminal do lado direito das regras 1 e 2 é B, e a regra que gera a sub-sentença $\chi_2 = a$ a partir de B é a 18. A mesma análise é feita para as regras 3 e 4 do vetor 3.2. Por outro lado, observe que o segundo símbolo não terminal do lado direito da regra 1 é B, o mesmo acontece com uma análise semelhante para a regra 3. Repare que agora estamos posicionados na entrada (3,1) da Tabela 6, pelo qual este símbolo B deve gerar, por meio de uma regra, a sub-sentença $\chi_3 = c$. Com esta informação preenchemos então um vetor de dimensão (4×1):

$$v_{S_{3,1}}^T = \begin{pmatrix} x_{20} & x_{25} & x_{20} & x_{25} \end{pmatrix}. \tag{3.4}$$

E com a Equação (3.5) atualizamos o vetor $vs_{2,2}$:

$$v_{S_{2,2}}^T = (vs_{2,2} * vs_{2,1} * vs_{3,1})^T, \tag{3.5}$$

onde, abusando da linguagem, dizemos que $(vs_{2,2} * vs_{2,1})$ é o produto componente a componente. Portanto o resultado é um vetor de dimensão (4×1):

$$v_{S_{2,2}}^T = \begin{pmatrix} x_1x_{18}x_{20} & x_2x_{18}x_{25} & x_3x_{23}x_{20} & x_4x_{23}x_{25} \end{pmatrix}, \tag{3.6}$$

onde cada entrada é o produto de todas as regras envolvidas em cada sub-árvore que gera a sub-sentença $\chi_{2,3} = ac$. Na Figura 20 desenhamos cada entrada do vetor $v_{S_{2,2}}$. Da mesma maneira, vamos obter valores para os vetores $v_{B_{22}}^T$ e $v_{C_{22}}^T$.

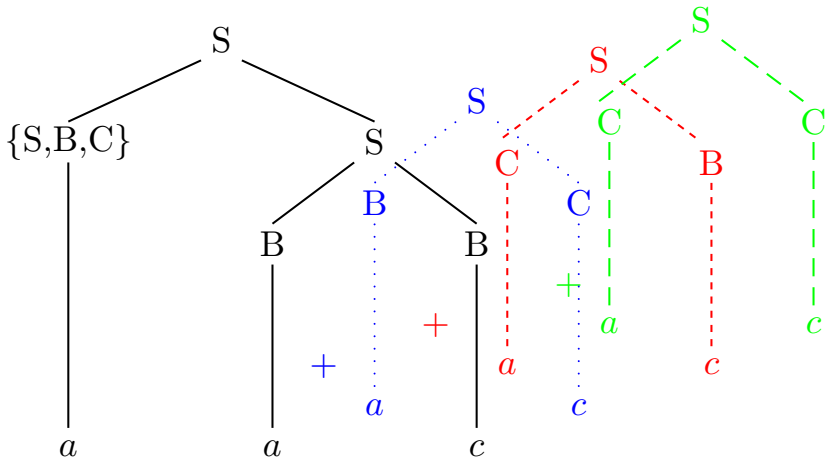

Figura 20 – Entradas do vetor $v_{S_{2,2}}$ na Equação (3.6).

Vamos agora nos posicionar na entrada (1,3) na Tabela 6; como antes podemos criar o vetor

$$v_{S_{3,3}}^T = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}, \tag{3.7}$$

e o seguinte vetor a partir do $v_{S_{3,3}}^T$:

$$v_{S_{1,1}}^T = \begin{pmatrix} x_{18} & x_{18} & x_{23} & x_{23} \end{pmatrix}. \tag{3.8}$$

Agora, analisando o lado direito das regras 1, 2, 3 e 4, e reparando também no segundo símbolo não terminal, podemos estabelecer que as possíveis regras que podem conectar na estrutura são, no caso da regra 1 as regras 5 até a 8. Para a regra 2, as regras desde a 9 até a 12, e assim por diante para as regras 3 e 4. Esta informação será armazenada em uma matriz que chamaremos de $VS_{2,2}$:

$$VS_{2,2} = \begin{pmatrix} x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \end{pmatrix}. \tag{3.9}$$

Agora, temos que juntar esta nova informação com as informações que já tínhamos

obtido anteriormente, isso é feito atualizando a matriz $VS_{2,2}$ para:

$$VS_{2,2} = \begin{pmatrix} v_{B_{2,2}}^T \\ v_{C_{2,2}}^T \\ v_{B_{2,2}}^T \\ v_{C_{2,2}}^T \end{pmatrix}. \tag{3.10}$$

Logo, o vetor $v_{S_{3,3}}^T$ é atualizado com a Equação (3.11):

$$v_{S_{3,3}}^T = (vs_{3,1} * vs_{2,2})^T VS_{2,2}. \tag{3.11}$$

Com isto, obtemos a informação de todas as possíveis árvores da primeira estrutura que está armazenada na primeira linha de D. Com um processo análogo obtemos para a segunda estrutura que está armazenada na segunda linha de D o vetor $\overline{v}_{S_{2,2}}^T$. Temos então, o valor da probabilidade para χ na gramática G_p da seguinte forma:

$$\Pr(\chi|G_p) = \sum_{i=1}^{\|v_{S_{2,2}}\|} v_{S_{2,2}}(i) + \sum_{i=1}^{\|\overline{v}_{S_{2,2}}\|} \overline{v}_{S_{2,2}}(i). \tag{3.12}$$

Com isto, finalizamos o processo de dois passos para obter a probabilidade de uma sentença dada.

Para finalizar o exemplo, fazemos os mesmos dois passos apresentados para a sentença η, obtendo assim o valor de $\Pr(\eta|G_p)$. Para finalmente obter a função de verossimilhança da amostra Ω:

$$\Pr(\Omega|G_p) = \Pr(\chi|G_p) * \Pr(\eta|G_p). \tag{3.13}$$

Assim, uma vez obtida a função de verossimilhança, determinamos um método para otimizá-la: o método de pontos interiores barreira logarítmica primal-dual.

4 Método de pontos interiores barreira logarítmica primal-dual

4.1 Introdução

Karmarkar no ano 1984 expõe com grande sucesso um algoritmo para programação linear [Kar84], sendo provavelmente, depois do desenvolvimento do método simplex, um dos mais importantes descobrimentos na área da programação linear. As propriedades teóricas do algoritmo de Karmarkar -complexidade polinomial- junto com seu bom desempenho em problemas práticos a grande escala, geraram uma explosão de pesquisas na área, tornando-se um dos métodos base para o desenvolvimento dos métodos de pontos interiores. Pesquisas mais focadas na teoria, se concentraram no método pontos interiores primal-dual cujos experimentos computacionais, que aconteceram em paralelo ao desenvolvimento teórico, mostraram que os métodos primal-dual em problemas práticos apresentavam melhores resultados em comparação a outros métodos de pontos interiores. No ano 1968, Fiacco e McCormick em [FM68] realizaram algumas observações que incluíam o fato que as condições suficientes para minimização sem restrições da função barreira logarítmica estavam localmente implícitas pelas condições KKT relaxadas e as condições suficientes de segundo ordem padrão. Em 1987, inspirados no trabalho de Meggido sobre métodos barreira logarítmica, Kojima, Mizuno, e Yoshise [KMY89] propuseram o método de pontos interiores primal-dual para programação linear, este método consiste basicamente em aplicar o método de Newton modificado às condições KKT relaxadas.

Em 1996, El-Bakry, Tapia, Tsuchiya and Zhang [EBTTZ96] motivados pelo grande sucesso computacional dos métodos de pontos interiores para programação linear, voltaram seu interesse para problemas, em geral, mais complexos da área: a programação não linear. Sendo assim apresentam em [EBTTZ96] uma formulação de métodos de pontos interiores para programação não convexa com restrições lineares e esclareceram a relação entre as condições KKT relaxadas e a função barreria logarítmica.

Neste capítulo apresentamos a formulação do método de pontos interiores barreira logarítmica primal-dual para problemas não convexos com restrições lineares. E, assim como no artigo [EBTTZ96] começamos apresentando a formulação linear para depois estabelecer a formulação para problemas de programação não linear.

4.2 Formulação para programação linear

Considere um problema de programação linear na forma padrão

$$\begin{aligned} \text{minimizar} \quad & c^t x \\ \text{sujeito a} \quad & Ax = b, \\ & x \geqslant 0, \end{aligned} \tag{4.1}$$

onde $c, x \in \Re^n$, $b \in \Re^m$ e $A \in \Re^{m \times n}$. Vamos supor que $m \leqslant n$ e posto$(A) = m$. Da teoria da dualidade sabemos que associado ao Problema (4.1) existe um outro problema denominado o dual [BJS10]:

$$\begin{aligned} \text{maximizar} \quad & b^t y \\ \text{sujeito a} \quad & A^T y + z = c, \\ & z \geqslant 0, \end{aligned} \tag{4.2}$$

onde $z \in \Re^n$ é denominada variável de folga dual.

Lembre-se que todo problema de otimização linear pode ser transformado na forma padrão [BJS10].

Um ponto $x \in \Re^n$ se diz estritamente factível para o Problema (4.1) se ele é factível e positivo. Um ponto $z \in \Re^n$ é factível para o Problema (4.2) se existe um $y \in \Re^m$ tal que (y, z) é factível para o Problema (4.2) e será estritamente factível se (y, z) é factível e $z > 0$.

As condições de otimalidade que devem satisfazer as soluções de problemas lineares, são derivados desde as propriedades intrínsecas da teoria da dualidade ou podem ser estabelecidas como um caso especial das condições de otimalidade de problemas de otimização gerais com restrições, as chamadas condições de Karush-kuhn-Tucker (KKT). Antes de definir o Teorema KKT (Teorema 4.2.2), garantimos que a condição de qualificação do mesmo seja satisfeita.

Definição 4.2.1. *[NW06] O conjunto das restrições ativas $\Lambda(x)$ em um ponto factível x, é o conjunto de índices das restrições de igualdade que estão sendo satisfeitas:*

$$\Lambda(x) = \{i : a_i^T x - b_i = 0\}, \tag{4.3}$$

com a_i^T a i-ésima linha da matriz A, b_i a i-ésima componente do vetor b e $1 \leqslant i \leqslant m$.

Teorema 4.2.1. *[NW06] Dado um ponto x e o conjunto das restrições ativas $\Lambda(x)$, dizemos que a condição de qualificação de independência linear é satisfeita se o conjunto dos gradientes das restrições ativas $\{\nabla c_i(x), i \in \Lambda(x)\}$ é linearmente independente.*

Observe que, quando todas as restrições $a_i^T x$ são funções lineares e a matriz A é de posto completo a condição de qualificação é satisfeita, e portanto, as condições de qualificação do 4.2.2 são satisfeitas [SY06].

Teorema 4.2.2 (Condições necessárias de primeira ordem [NW06]). *Suponha que o vetor $x^* \in \Re^n$ é uma solução local para o Problema (4.1) onde as condições de qualificação de independência linear é satisfeita em x^*. Então existem vetores $z^* \in \Re^n$ e $y^* \in \Re^n$ tal que para $(x^*, y^*, z^*) = (x, y, z)$ as seguintes condições são satisfeitas*

$$\begin{aligned} A^T y + z &= c && && (4.4a)\\ Ax &= b && && (4.4b)\\ x_i z_i &= 0 && i = 1...n && (4.4c)\\ (x, z) &\geqslant 0. && && (4.4d) \end{aligned}$$

Na terminologia da otimização com restrições, os vetores y e z são os multiplicadores de Lagrange das restrições $Ax - b$ e $x \geqslant 0$, respectivamente. A condição (4.4a), chamada de condição de complementaridade, estabelece que para cada $i = 1, ..., n$, uma das componentes x_i ou z_i deve ser zero. O Sistema (4.4) é chamado de Sistema KKT e um vetor $(x, y, z)^T$ que resolva o Sistema KKT é chamado de ponto KKT.

Do ponto de vista da teoria da dualidade dizemos que os valores x^* e (y^*, z^*) que resolvem o Sistema (4.4) são soluções dos problemas primal e dual, respectivamente. A teoria da dualidade explica a relação entre os problemas primal (4.1) e dual (4.2), assim o conjunto factível e o conjunto de soluções para o problema primal contém muita informação sobre o dual, e vice-versa. Por exemplo, como consequência imediata da condição de complementaridade (ver Equação (4.4c)) temos que $(x^*)^t z = 0$ sempre que x^* e z^* satisfazem a condição (4.4c). E, se (x, y, z) satisfazem as três condições (4.4a), (4.4b) e (4.4d), temos que:

$$0 \leqslant z^t x = (c - A^t y)^t x = c^t x - y^t (Ax) = c^t x - b^t y, \tag{4.5}$$

portanto $b^t y \leqslant c^t x$. Se existe um ponto (x^*, y^*, z^*) que satisfaz as quatro condições em (4.4), então temos que $(x^*)^t z = 0$ implica $b^t y^* = c^t x^*$, pelo que os valores ótimos do primal e do dual são os mesmos. Deriva-se então o seguinte teorema.

Teorema 4.2.3. *Teorema da dualidade para programação linear [NW06]*

- *Se o problema primal tem uma solução com valor objetivo finito, então garante-se que o problema dual também possui solução finita, e vice-versa. Além disso, os valores ótimos das funções objetivos são iguais.*
- *Se a função objetivo do problema primal (ou dual) é ilimitada, então o problema dual (ou primal) não possui pontos factíveis.*

Assim, um vetor (x^*, y^*, z^*) resolve o Sistema (4.4) se, e somente se, x^* resolve o problema primal (4.1) e (y^*, z^*) resolve o problema dual (4.2). O vetor (x^*, y^*, z^*) é dito solução ótima primal-dual.

4.2.1 *Gap* de Dualidade

A partir das condições KKT (4.4), para um x, y e z factíveis, o valor $c^T x - b^T y = z^T x$ é a diferença entre o valor da função objetivo primal e o valor da função objetivo dual. Este valor, sempre não negativo, é denominado o *gap* de dualidade.

Pelo Teorema 4.2.3, os valores ótimos do primal e dual são iguais ($c^T x^* = b^T y^*$), isto sugere que o *gap* de dualidade é uma quantidade ou medida de quão perto estamos de uma solução ótima [LY08].

Para complementar os estudos sobre a teoria da dualidade que explica a relação entre os problemas (4.1) e (4.2) sugerimos estudar [Wri97].

4.2.2 Método de pontos interiores primal-dual

Os métodos de pontos interiores pertencem ao grupo de métodos iterativos tais que a cada iteração calculam uma direção de melhora e determinam o tamanho do passo para caminhar nessa direção. Espera-se que a cada iteração estejamos mais próximos de uma solução, além disso, os pontos calculados a cada iteração devem ser não negativos, e usando a trajetória central garante-se que são pontos interiores. O cálculo da direção e do tamanho do passo caracterizam os métodos de pontos interiores [Wri97].

O método de pontos interiores primal-dual procura soluções primal-dual usando uma variante do método de Newton para resolver o Sistema (4.4), determinando assim as direções de busca. E, define os tamanhos dos passos, garantindo que a desigualdade $(x, z) > 0$ seja estritamente satisfeita a cada iteração.

O Sistema (4.4) pode ser representado usando um mapeamento F de R^{2n+m} para R^{2n+m}:

$$F(x, y, z) : \begin{pmatrix} Ax - b \\ A^T y + z - c \\ XZe \end{pmatrix} = 0, \quad (x, z) \geqslant 0, \tag{4.6}$$

onde X denota una matriz diagonal, cuja diagonal é de fato o vetor x. Usamos, analogamente, a mesma notação para a matriz Z. Denotamos por e como o vetor de uns cuja dimensão varia segundo o contexto.

A cada iteração k, o método de Newton aproxima F a um modelo linear numa vizinhança B_r de um ponto atual (x^k, y^k, z^k), e obtém as direções de busca $(\Delta^k x, \Delta^k y, \Delta^k z)$, resolvendo o sistema linear:

$$J(x^k, y^k, z^k) \begin{pmatrix} \Delta^k x \\ \Delta^k y \\ \Delta^k z \end{pmatrix} = -F(x^k, y^k, z^k), \tag{4.7}$$

onde $(\Delta^k x, \Delta^k y, \Delta^k z) \in B_r(x^k, y^k, z^k)$, e J é o Jacobiano de F. Se o ponto atual é estritamente fatível as equações no passo de Newton torna-se:

$$\begin{pmatrix} 0 & A^t & I \\ A & 0 & 0 \\ S & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta z^k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -XZe \end{pmatrix}. \tag{4.8}$$

Um passo completo nesta direção, geralmente não é permitido porque poderia violar a condição $(x^k, z^k) \geqslant 0$, assim para evitar esta dificuldade se realiza o passo da seguinte maneira:

$$(x^{k+1}, y^{k+1}, z^{k+1}) = (x^k, y^k, z^k) + \alpha(\Delta x^k, \Delta y^k, \Delta z^k), \tag{4.9}$$

onde $0 < \alpha \leqslant 1$.

Como eventualmente o comprimento do passo pode ser pequeno ($\alpha^k \ll 1$) para evitar violar a condição $(x^k, z^k) > 0$, o progresso, no sentido de encontrar uma solução, pode ser lento. Para isso, o método de pontos interiores primal-dual modifica o Sistema (4.7) para que o novo ponto seja mais centralizado no interior do ortante positivo $(x^k, z^k) > 0$, e assim, poder dar passos mais longos antes que x^k ou z^k fiquem negativos. Logo, o conceito de trajetória central [Wri97] é definido a seguir.

Trajetória Central

A trajetória central C é formada pelo conjunto de pontos $(x, y, z) \in C$ tal que resolve o sistema:

$$\begin{aligned} Ax &= b && \text{(4.10a)} \\ A^T y + z &= c && \text{(4.10b)} \\ XZe &= \mu e && \text{(4.10c)} \\ (x, z) &> 0, && \text{(4.10d)} \end{aligned}$$

com $\mu > 0$.

Outra forma de escrever a trajetória central é:

$$\begin{aligned} F(x,y,z) &= [0\ ,\ 0\ ,\ \mu e]^t \\ (x,z) &> 0, \end{aligned} \tag{4.11}$$

com $\mu > 0$.

Assim, a trajetória central é um arco de pontos estritamente factíveis que satisfazem as Equações (4.10). Note que quando $\mu \longrightarrow 0$, a solução (x,y,z) de (4.10) aproxima-se de uma solução do Sistema (4.4).

Usamos a trajetória central para relaxar as restrições de complementaridade em (4.4):

$$\begin{aligned} F(x,y,z): \quad & \begin{pmatrix} Ax-b \\ A^Ty+z-c \\ XZe-\mu e \end{pmatrix} = 0, \\ & (x,z) \geqslant 0, \qquad \mu > 0, \end{aligned} \tag{4.12}$$

onde o Sistema (4.12) é chamado de condições KKT relaxadas. Aplicamos o método de Newton numa vizinhança B_r de (x,y,z) obtendo o sistema linear:

$$J(x^k,y^k,z^k)\begin{pmatrix} \Delta^k x \\ \Delta^k y \\ \Delta^k z \end{pmatrix} = -F(x^k,y^k,z^k) + \begin{pmatrix} 0 \\ 0 \\ \mu^k e \end{pmatrix}, \tag{4.13}$$

assim $(x^k,y^k,z^k) \longrightarrow (x^*,y^*,z^*)$ quando $\mu \longrightarrow 0$.

4.2.3 Interpretação alternativa

Uma segunda derivação dos métodos de pontos interiores associa o Problema (4.1) à um problema de barreira logarítmica:

$$\begin{aligned} \text{minimizar} \quad & c^Tx - \mu\sum_{i=1}^{n} log(x_i) \\ \text{sujeito a} \quad & Ax = b, \end{aligned} \tag{4.14}$$

onde $\mu > 0$. O termo $\sum_{i=1}^{n} log(x_i)$ é chamado da *barreira logarítmica* [1] porque impede que as variáveis x_i sejam negativas. Obtemos soluções $\{x^k\}$ do Problema (4.14) para uma

[1] O uso do mesmo parâmetro μ para representar a trajetória central e a barreira logarítmica é feito de propósito para notar relação entre o método primal-dual para problemas lineares e o método barreira logarítmica primal-dual para problemas não lineares, assim como pode ser visto em [EBTTZ96].

sequência de $\{\mu^k\}$ positivos, tal que $\{x^k\} \longrightarrow \{x^*\}$ quando $\mu^k \longrightarrow 0$, onde x^* é uma solução do Problema (4.1).

Uma das abordagens para resolver o Problema (4.14), consiste em utilizar a função de Lagrange para obter um problema equivalente irrestrito:

$$\text{minimizar} \quad \mathcal{L}(x, y), \tag{4.15}$$

com $\mathcal{L}(x, y) = c^T x - \mu \sum_{i=1}^{n} log(x_i) + y^T(b - Ax)$, para cada $\mu > 0$.

Desta forma, podemos procurar soluções para o problema com restrições de igualdade (Problema (4.14)) no conjunto de pontos estacionários da função de Lagrange associada. O vetor y é denominado multiplicador de Lagrange da restrição de igualdade $Ax = b$:

$$\begin{array}{rlcl} \nabla_x \mathcal{L} := & \mu X^{-1} e + A^T y - c & = & 0, \\ \nabla_y \mathcal{L} := & Ax - b & = & 0. \end{array} \tag{4.16}$$

Observe que o Sistema KKT relaxado (ver Sistema (4.12)), pode ser obtido a partir das Condições (4.16) introduzindo uma variável auxiliar $z = \mu X^{-1} e > 0$ pelo qual é definida uma nova relação $XZe = \mu e$.

Assim, a cada iteração o método resolve o Sistema (4.12). Detalhes do método, as implementações, assim como estudos de convergência podem ser estudadas em [Wri97], [LY08].

4.3 Formulação para o problema não linear

Os métodos de pontos interiores (ou barreira) têm obtido exito para resolver tanto problemas de otimização não linear como de programação linear, atualmente pertencem ao grupo dos métodos que têm sido considerados como os métodos mais robustos para programação não linear de grande porte [NW06]. Algumas das ideias chaves tais como passo primal-dual são derivadas diretamente da programação linear mas com algumas mudanças substanciais, entre os quais estão a manipulação da não convexidade, a estratégia para atualizar o parâmetro barreira na presença da não linealidade e a necessidade de garantir o progresso até uma solução.

Baseados na formulação da programação linear, derivamos o método de pontos interiores primal-dual para o problema geral de programação não linear:

$$\begin{array}{rl} \text{minimizar} & f(x) \\ \text{sujeito a} & h(x) = 0 \\ & g(x) \geqslant 0, \end{array} \tag{4.17}$$

onde $f : \Re^n \to \Re$, $h : \Re^n \to \Re^m$ e $g : \Re^n \to \Re^p$.

Se x é um ponto factível para o Problema (4.17), o conjunto $\mathfrak{B}(x)$ contém os índices associados às restrições de desigualdade tal que

$$\mathfrak{B}(x) = \{i : g_i(x) = 0, i = 1, ..., p\}. \tag{4.18}$$

As condições KKT para o Problema (4.17) são

$$\nabla_x L(x, y, z) = 0 \tag{4.19a}$$
$$h(x) = 0 \tag{4.19b}$$
$$g(x) \geqslant 0 \tag{4.19c}$$
$$Zg(x) = 0 \tag{4.19d}$$
$$z \geqslant 0, \tag{4.19e}$$

onde o Lagrangiano associado ao Problema (4.17) é $L(x, y, z) = f(x) + y^t h(x) - z^t g(x)$, logo $\nabla_x L(x, y, z) = \nabla f(x) + \nabla h(x)y - \nabla g(x)z$.

Antes de relaxar as Condições KKT (Equações (4.19)) e aplicar o método de Newton, assumimos que as seguintes condições para o Problema (4.17) estão sendo satisfeitas localmente [EBTTZ96]:

(A) **existência**. Vamos supor que existe uma solução (x^*, y^*, z^*) para o Problema (4.17) que junto com os multiplicadores associados, satisfazem as condições KKT (4.19),

(B) **suavidade**. As matrizes Hessianas $\nabla^2 f(x)$, $\nabla^2 h(x)$ e $\nabla^2 g(x)$ existem e são Lipschitz localmente contínuas em x^*,

(C) **regularidade**. O conjunto $\{\nabla h_1(x^*), ..., \nabla h_m(x^*)\} \cup \{\nabla g_i(x^*) : i \in \mathfrak{B}(x)\}$ é linearmente independente,

(D) **condições suficientes de segunda ordem**. Para todo $\eta \neq 0$ tal que $\nabla h_i(x^*)^t \eta = 0$, $i = 1, ..., m$ e $\nabla g_i(x^*)^t \eta = 0$, $i \in \mathfrak{B}(x^*)$, temos que $\eta^t \nabla_x^2 L(x^*)\eta > 0$,

(E) **complementaridade estrita**. Para todo i, $z_i^* + g_i(x^*) > 0$.

Com estas condições, temos que para um μ suficientemente pequeno e positivo o sistema relaxado (4.20) possui uma única solução local. A trajetória descrita por estes pontos é chamada trajetória central primal-dual e converge ao ponto (x^*, y^*, z^*) quando

$\mu \longrightarrow 0$.

$$\nabla_x L(x, y, z) = 0 \tag{4.20a}$$
$$h(x) = 0 \tag{4.20b}$$
$$g(x) \geqslant 0 \tag{4.20c}$$
$$Zg(x) - \mu e = 0 \tag{4.20d}$$
$$z \geqslant 0, \tag{4.20e}$$

Assim, como no caso do problema linear, as condições KKT (4.20) podem ser vistas como um mapeamento $F : R^{2n+m} \rightarrow R^{2n+m}$

$$F(x, y, z) : \begin{pmatrix} \nabla_x L(x, y, z) \\ h(x) \\ g(x) - s \\ ZSe \end{pmatrix} = 0, \quad (s, z) \geqslant 0, \tag{4.21}$$

onde temos considerando a variável de folga s com a seguinte preposição sendo satisfeita:

Proposição 4.3.1. *[EBTTZ96] Suponha que as condições (A) e (B) se cumprem. E seja $s^* = g(x^*)$, logo o seguinte é satisfeito:*

(i) As condições (C) até (E) também são satisfeitas,

(ii) A matriz Jacobiana $F'(x^*, y^*, s^*, z^*)$ de $F(x, y, s, z)$ em (4.21) é não singular.

Demostração 4.3.1. *A demostração pode ser estudada em [EBTTZ96].*

Assim, aplicando o método de Newton em F definida por (4.21), numa vizinhança das variáveis (x^k, y^k, s^k, z^k) obtemos o seguinte sistema para ser resolvido nas variáveis $(\Delta x^k, \Delta s^k, \Delta y^k, \Delta z^k)$:

$$\begin{pmatrix} \nabla^2_{xx} L & 0 & \nabla h(x) & -\nabla g(x) \\ 0 & Z & 0 & S \\ \nabla h(x) & 0 & 0 & 0 \\ \nabla g(x) & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta s^k \\ \Delta y^k \\ \Delta z^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x) + \nabla h(x) y - \nabla g(x) z \\ SZe - \mu e \\ h(x) \\ g(x) - s \end{pmatrix}, \tag{4.22}$$

onde k corresponde à k-ésima iteração do método de pontos interiores primal-dual, e o vetor de uns, Z e S as matrizes diagonais.

Com estas informações podemos, inicialmente, apresentar um resumo para o método de pontos interiores:

Método 2 – Pontos interiores barreira logarítmica primal-dual

Dados: (x^0, y^0, z^0, s^0), tal que $(z^0, s^0) > 0$

para $k = 0,1,2...$ **faça**

determinar $\mu^k > 0$;

resolver o sistema linear (4.22) para $(\Delta^k x, \Delta^k y, \Delta^k z, \Delta^k s)$;

calcular

$$\hat{\alpha_z} = max\{\alpha \in (0,1] : s + \alpha\Delta z \geqslant (1-\tau)z\} \tag{4.23a}$$

$$\hat{\alpha_s} = max\{\alpha \in (0,1] : s + \alpha\Delta s \geqslant (1-\tau)s\} \tag{4.23b}$$

onde $\tau \in (0,1)$;

atualizar o passo

$$x^{k+1} = x^k + \alpha_s \Delta^k x, \tag{4.24}$$

$$s^{k+1} = s^k + \alpha_s \Delta^k s, \tag{4.25}$$

$$y^{k+1} = y^k + \alpha_z \Delta^k y, \tag{4.26}$$

$$z^{k+1} = z^k + \alpha_z \Delta^k z \tag{4.27}$$

se *um critério de parada dado é satisfeito* **então**

parar;

senão

faça $k = k + 1$ e volte ao Passo 2;

fim

fim

4.3.1 Modificações do método

Vamos discutir nesta seção algumas modificações do Método 2 que permitam garantir o progresso do método para resolver problemas gerais não lineares na presença da não convexidade e da não linealidade.

4.3.1.1 Resolução do sistema

Além do custo computacional para calcular o valor da função objetivo e suas derivadas a cada iteração, o trabalho computacional do método de pontos interiores também é dominado pela resolução do sistema linear (4.22), portanto, uma forma eficiente (técnicas de fatoração para matrizes esparsas ou métodos iterativos) é indispensável na busca da solução para problemas de grande porte.

Em alguns casos o sistema (4.22) é reescrito de uma forma simétrica da seguinte forma:

$$\begin{pmatrix} \nabla^2_{xx} L & 0 & \nabla h(x) & -\nabla g(x) \\ 0 & \Sigma & 0 & S \\ \nabla h(x) & 0 & 0 & 0 \\ \nabla g(x) & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta s^k \\ \Delta y^k \\ \Delta z^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x) + \nabla h(x) y - \nabla g(x) z \\ z - \mu S^{-1} e \\ h(x) \\ g(x) - s \end{pmatrix}, \tag{4.28}$$

com $\Sigma = S^{-1}Z$. Esta formulação permite usar técnicas para sistemas lineares simétricos. Uma das abordagens é por exemplo começar eliminando a variável $\Delta^k s$ usando a segunda linha em (4.28), obtendo

$$\begin{pmatrix} \nabla^2_{xx} L & \nabla h(x) & -\nabla g(x) \\ \nabla h(x) & 0 & 0 \\ \nabla g(x) & 0 & -\Sigma^{-1} \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta z^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x) + \nabla h(x) y - \nabla g(x) z \\ h(x) \\ g(x) - \mu Z^{-1} e \end{pmatrix}, \tag{4.29}$$

este sistema pode ser fatorado usando fatoração simétrica indefinida, por exemplo: vamos denotar a matriz de coeficientes do Sistema (4.29) como K, e usando este tipo de fatoração obtemos $P^t K P = LBL^t$, onde L é triangular inferior e B é diagonal por blocos, com blocos de tamanho 1×1, ou 2×2, e P matriz de permutações de linhas e colunas que permitem preservar a esparsidade e estabilidade numérica do sistema. Para o estudo de diversos métodos o leitor pode estudar o Capítulo XVI em [NW06].

Outra abordagem, consiste em continuar reduzindo o sistema (4.29), eliminando a variável Δz e obtendo um sistema semelhante, onde a matriz de coeficientes é como segue:

$$\begin{pmatrix} \nabla^2_{xx} L + \nabla^t g(x) \Sigma \nabla g(x) & \nabla^t h(x) \\ \nabla h(x) & 0 \end{pmatrix}, \tag{4.30}$$

o qual é bem menor do que o Sistema (4.28) quando o número de restrições de desigualdade é grande. Ainda que ele perda esparsidade com o termo $\nabla^t g(x) \Sigma \nabla g(x)$, isto é tolerável em algumas aplicações [NW06]. Um caso particularmente favorável é quando $\nabla^t g(x) \Sigma \nabla g(x)$ é diagonal, isto acontece quando as restrições de desigualdade são simples variáveis [NW06].

O sistema, tanto na forma (4.30), (4.29), ou na forma (4.28) é mal condicionado por causa de Σ, pois alguns elementos de Σ se aproximam de ∞ entanto outros vão para zero, quando $\mu \to 0$. Técnicas de álgebra linear iterativas são usadas para calcular as direções, e dado o mal condicionamento da matriz, precondicionadores devem ser usados, por exemplo, suponha a nova variável $\tilde{p} = S^{-1}\Delta s$ em (4.28) e multiplicamos a segunda equação por S transformando o termo Σ por $S\Sigma S$. Assim, quando $\mu \to 0$ e supondo que $SZ \approx \mu I$, temos que como $\Sigma = S^{-1}Z$ então todos os elementos de $S\Sigma S$ ficam perto de μI. Outro tipo de precondicionadores podem ser revisados em [Gon12].

É possível aplicar métodos iterativos para quaisquer dos três sistemas simétricos indefinidos (4.30), (4.29) e (4.28)), tais como GMRES, QMR, LSQR, etc [TBI97]. Importante notar, que além de usar os precondicionadores para resolver o mal condicionamento que acontece pelo parâmetro barreira, devemos resolver o mal condicionamento causados, possivelmente, pela matriz Hessiana $\nabla^2_{xx}L$ e/ou las matrizes Jacobianas $\nabla h(x)$ e $\nabla g(x)$. Neste contexto, os precondicionadores são construídos especificamente para cada problema em particular [NW06].

4.3.1.2 Atualização do parâmetro barreira

A sequência do parâmetro barreira $\{\mu^k\}$ deve convergir a zero para que no limite podamos determinar a solução do problema de programação não linear (4.17). Se μ tem um decréscimo lento, o número de iterações até atingir convergência será grande, entanto que se o decréscimo é muito rápido, algumas das variáveis de folga z ou s podem, prematuramente, ir para zero, implicando um progresso lento da iteração [NW06].

Algumas das estratégias atualizam o parâmetro barreira a cada iteração, em alguns casos tal como na abordagem Fiacco-McCormick este parâmetro é fixo até que satisfaz alguma condição [SY06] . A maioria das estratégias que atualizam a cada iteração, estão baseadas no conceito de complementaridade da programação linear e é definida como

$$\mu^{k+1} = \sigma \frac{(s^k)^t z^k}{n}. \tag{4.31}$$

Note que se x^k e z^k fossem factíveis, γ^k é o *gap* na k-ésima iteração.

Como o propósito da perturbação consiste em centralizar os iterados e adicionalmente deseja-se que $\mu^k \longrightarrow 0$ quando $k \longrightarrow \infty$, usamos um valor no intervalo $(0, 1)$ para σ. Normalmente $\sigma = \dfrac{1}{\sqrt{n}}$ [Wri97].

Desta forma garante-se que os iterandos estão centralizados e que a cada iteração o valor de μ^k está sendo reduzido.

4.3.1.3 Função de mérito e cálculo do tamanho do passo α

O objetivo da função de mérito é determinar quando um passo é produtivo e deveria ser aceito. Como tem sido discutido neste capítulo, os métodos de pontos interiores podem ser vistos como métodos para resolver o problema de barreira, logo é apropriado definir a função de mérito ϕ em termos da função barreira, por exemplo, podemos usar uma função da forma [NW06]

$$\phi_\upsilon(x, s) = f(x) - \mu \sum_{i=1}^{m} log(s_i) + \upsilon \parallel \nabla h(x) \parallel + \upsilon \parallel \nabla g(x) - s \parallel, \tag{4.32}$$

com $\upsilon > 0$, e a norma ℓ_1 ou ℓ_2.

Para garantir um decréscimo suficiente na função de mérito (4.32), uma vez calculadas as direções $(\Delta x, \Delta y, \Delta s, \Delta z)$ e determinados os valores de $\hat{\alpha}_s$ e $\hat{\alpha}_z$, ver a Equação (4.23), usamos a técnica de *backtracking* que determina os valores de:

$$\alpha_s \in (0, \alpha_s], \quad \alpha_s \in (0, \alpha_z], \tag{4.33}$$

os quais garantem o decréscimo suficiente em (4.32) [NW06] e Seção 4.5.

4.4 Problema do trabalho

Nesta última seção apresentamos o método de pontos interiores barreira logarítmica primal-dual, o qual é um caso particular da formulação para problemas não lineares estudado na seção anterior. Suponha o problema

$$\begin{array}{ll} \text{minimizar} & f(x) \\ \text{sujeito a} & Ax - b = 0 \\ & x \geqslant 0, \end{array} \tag{4.34}$$

e vamos supor $f : \Re^n \to \Re$, uma função suave, $x \in \Re^n$, $b \in \Re^m$ e $A \in \Re^{m \times n}$, com $m \leqslant n$ e posto$(A) = m$.

Assim como antes, começamos apresentando a função de Lagrange do Problema (4.34):

$$L(x, y, z) = f(x) + y^t(Ax - b) - z^t x, \tag{4.35}$$

onde y e z são os multiplicadores de Lagrange associados.

As condições KKT para o Problema (4.34) são:

$$\nabla f(x) + A^t y - z = 0 Ax - b = 0 ZXe = 0 (x, z) \geqslant 0. \tag{4.36a}$$

Vamos supor a existência da solução (x^*, y^*, z^*) de (4.36) e que $f(x)$ é uma função suave. Note que na suposição que A é posto completo, então a condição de regularidade é satisfeita. E por fim vamos supor que a condição suficiente de segundo ordem é satisfeita assim como a complementaridade estrita.

Desse modo e relaxando o Sistema (4.36) com $\mu > 0$ temos

$$\nabla f(x) + A^t y - z = 0 Ax - b = 0 ZXe = \mu (x, z) \geqslant 0, \tag{4.37a}$$

Por outro lado, e assim como foi demostrado no caso do problema linear, as condições KKT relaxadas (4.37) são equivalentes às condições KKT para o problema função barreira logarítmica. Considere o problema função barreira logarítmica associado ao Problema

(4.34)

$$\begin{aligned} \text{minimizar} \quad & f(x) \text{ - } \mu \sum_{i=1}^{n} log(x_i), \\ \text{sujeito a} \quad & Ax - b = 0, \\ & x > 0, \end{aligned} \tag{4.38}$$

para $\mu > 0$ fixo. Para cada valor fixo de μ e usando o Lagrangiano associado:

$$\mathcal{L}(x, y) = f(x) - \mu \sum_{i=1}^{n} ln(x_i) + y^T(b - Ax),$$

temos que uma solução x^* do Problema (4.38), deve satisfazer as condições de otimalidade [Wri97]:

$$F_\mu(x, y) : \left(\nabla f(x) - \mu X^{-1} e + A^T y Ax - b\right) = 0, \tag{4.39}$$

o vetor $y \in \Re^m$ é o multiplicador de Lagrange e $F : \Re^{n+m} \longrightarrow \Re^{n+m}$.

Introduzindo uma nova variável $z = \mu X^{-1}$ em (4.39), temos

$$\hat{F}_\mu(x, y) : \left(\nabla f(x) - z + A^T y X Z e - \mu e Ax - b\right) = 0, \tag{4.40}$$

Dado que (4.40) é um sistema não linear, usamos o método de Newton para aproximar $\hat{F}_\mu$ a um modelo linear numa vizinhança de (x, y, z) [LY08]:

$$\begin{pmatrix} H(x) & A^T & IZ & 0 & XA & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \Delta z \Delta y \end{pmatrix} = \begin{pmatrix} r_p r_d r_c \end{pmatrix}, \tag{4.41}$$

onde $H(x) = \nabla^2 f(x)$, $r_p = z - A^T y - \nabla f(x)$, $r_d = \mu e - XZe$, $r_c = b - Ax$.

Usando a segunda equação em (4.41) eliminamos a variável Δz, obtendo o seguinte sistema reduzido:

$$\begin{pmatrix} H(x) - X^{-1}Z & A^T A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \Delta y \end{pmatrix} = \begin{pmatrix} \hat{r}_p r_c \end{pmatrix}, \tag{4.42}$$

onde $\hat{r}_p = r_p - X^{-1} r_u$ e o valor de Δz pode ser obtido de

$$\Delta z = X^{-1}(r_u - Z\Delta x). \tag{4.43}$$

Para resolver o sistema simétrico (4.42) que possui uma matriz esparsa, mal condicionada, e indefinida, utilizamos um método iterativo precondicionado: o método de gradiente bi-conjugado estabilizado para matrizes esparsas [Saa03], com um precondicionador baseado nas entradas da diagonal [Gon12].

Uma vez obtidas as direções de Newton $(\Delta x, \Delta y, \Delta z)$, para garantir que as variáveis $x^{k+1}, y^{k+1}, z^{k+1}$ não se tornem negativas quando o ponto é atualizado, usamos o teste da razão:

$$\alpha = min(1, \tau\alpha_p, \tau\alpha_d), \quad \text{com} \quad \tau \in (0, 1], \tag{4.44}$$

onde

$$\alpha_p = \frac{-1}{min_i(\frac{\Delta x_i}{x_i})} \quad e \quad \alpha_d = \frac{-1}{min_i(\frac{\Delta z_i}{z_i})}. \tag{4.45}$$

A seguir vamos resumir o método de pontos interiores barreira logarítmica primal-dual para o caso particular do Problema (4.34):

Método 3 – Pontos interiores barreira logarítmica primal-dual implementado

Dados: y^0, $(x^0, z^0) > 0$, $\sigma = \frac{1}{2n}$,

para *k = 0,1,2...* **faça**

calcular $\mu = (\frac{\gamma}{2n})$, onde $\gamma = (x)^t z$;

calcular as direções $\Delta x, \Delta y, \Delta z$ resolvendo o sistema linear (4.41) ;

determinar α usando o teste da razão (4.44) e (4.44) ;

atualizar o passo

$$(x^{k+1}, y^{k+1}, z^{k+1}) = (x^k, y^k, z^k) + \alpha(\Delta x, \Delta y, \Delta z);$$

se *o critério de parada dado é satisfeito* **então**

parar;

senão

faça $k = k + 1$ e volte ao Passo 2;

fim

fim

4.5 Controle do tamanho do passo

Como o Problema (4.34) pode ser ou não convexo, as direções dadas pelo Sistema (4.41) nem sempre garantem progresso com tamanho do passo dado por α, ver (4.44) e (4.45).

Para abordar esta dificuldade usamos um controle de passo simples (*backtracking*) para a direção Δx:

Método 4 – Controle de passo

Dados: $0 < \rho < 1$, $\alpha > 0$

enquanto $f(x^k + \alpha\Delta x) > f(x^k)$ **faça**

$\alpha = \rho\alpha$;

fim

Além disso exigimos que o valor dos resíduos e do *gap* seja reduzido para satisfazer a factibilidade da solução, para isso usamos de novamente um controle de passo

simples (*backtracking*) para o valor dos resíduos e do *gap*:

Método 5 – Controle de passo factibilidade

Dados: $0 < \rho < 1$, $\alpha > 0$ e $(\Delta x, \Delta z, \Delta y)$
enquanto $(||r_p^k|| > ||r_p^{k+1}||)$ *e* $(||r_d^k|| > ||r_d^{k+1}||)$ *e* $(||\gamma^k|| > ||\gamma^{k+1}||)$ **faça**
  $\alpha = \rho\alpha$;
fim

Acrescentando o Método 4 e Método 5 antes de continuar com o Passo 5 no Método (3) buscamos um mínimo local factível do Problema (4.34).

5 Experimentos e resultados

5.1 Introdução

Em capítulos anteriores temos estudado o método de pontos interiores para estimar uma gramática probabilística livre de contexto $G_P = (N, \Sigma, P, S)$. O aprendizado da gramática é feito mediante um processo baseado em *corpus*. As características e os objetos da gramática livre de contexto na forma normal de Chomsky, foram extraídas do *corpus* fazendo uso de uma implementação própria, criada em linguagem de programação Perl versão 5.18.2.

Um trabalho prévio foi realizado em [Ló13] no qual foram testados o método de pontos interiores barreira logarítmica primal-dual e o método de pontos interiores barreira logarítmica. Estes métodos foram testados em duas gramáticas com tamanho da amostra de até 4.000 sentenças, a partir dos resultados obtidos nesse trabalho, em termos de número de iterações até convergir, tempo de convergência e ponto inicial concluímos que para a finalidade deste trabalho é suficiente implementar o método de pontos interiores barreira logarítmica primal-dual, descrito no Método 3.

Assim mesmo temos desenvolvido um procedimento, o qual é uma variação do algoritmo CYK que foi detalhado no Capitulo 3. Esta variação do algoritmo CYK foi desenvolvida para obter a função de verossimilhança a partir do *corpus*.

Esse procedimento, assim como o método de pontos interiores foram implementados na linguagem C++, versão 4.8.4. A estrutura *Open Multiprocessing* (OpenMP 2.5) foi aproveitada para melhorar o rendimento dos métodos. Foram usadas duas máquinas, a primeira a chamamos de Máquina 1, com um sistema operacional Ubuntu 14.04.5 LTS, 40 CPUs Intel Xeon de 2.3 GHz e memória Ram 96GiB. A segunda máquina: a Máquina 2 possui sistema operacional Ubuntu 14.04.5 LTS, 24 CPUs Intel Xeon de 2.4 GHz e memória Ram 66GB.

Dado que as implementações feitas possuem alguns desafios computacionais, falaremos brevemente desse assunto antes de apresentar os resultado obtidos.

5.2 Implementação e desafios computacionais

Antes de começar, vamos expor o tamanho do problema que desejamos resolver. Na Tabela 8, vemos que a gramática contém 713.160 regras de derivação, portanto a função de verossimilhança é um polinômio em 713.160 variáveis, cujos valores devem estar no intervalo $[0, 1]$. Supondo que para o treino, trabalhamos com o 80% do *corpus*, teríamos,

segundo a Tabela 7, uma quantidade de 4.320 sentenças. Do qual podemos determinar que temos um sistema linear esparso de dimensão 713.160 × 713.160.

Alguns detalhes da implementação são discutidos a seguir: extrair informação do *corpus*, cálculo da função de verossimilhança e a resolução do sistema linear no método pontos interiores barreira logarítmica primal-dual, bem como alguns dos desafios computacionais.

5.2.1 *Corpus* de treino

Para os experimentos propomos utilizar uma parte do *corpus*[1] que contém 5.399 sentenças, tal que seu comprimento $4 \leqslant \|n\| \leqslant 15$, isto para diminuir o esforço computacional nos testes, um processo similar é feito em [Che95]. Desse modo, o tamanho do vocabulário é de 47.348 palavras e um vocabulário sintático constituído de 15 etiquetas sintáticas. As características do *corpus* do *Tycho Brahe* depois do seletar as sentenças de interesse, estão dadas na Tabela 7.

Nro de sentenças	Meia do comprimento
5.399	10,9253

Tabela 7 – Características das sentenças do *corpus* a ser utilizado.

O *corpus* descrito anterior, é dividido em dois grupos, um conjunto para o treino T_t e um conjunto para calcular a perplexidade por palavra T_{pp}, usualmente se trabalha com um 80% do *corpus* para o trabalho de treino e um 20% para os cálculos dos valores da perplexidade por palavra. Na Tabela 8 apresentamos as características da gramática que vamos treinar. Lembrando que para o processo de aprendizagem da GPLC estamos propondo um método de pontos interiores.

Nro símbolos não terminais $\|N\|$	Nro símbolos terminais $\|\Sigma\|$	Nro de regras $\|P\|$	Símbolo inicial
15	47.348	713.160	S

Tabela 8 – Características da gramática de treino G_p

5.2.2 Extrair informação do *corpus*

O fragmento do *corpus* do *Tycho Brahe* com o qual estamos trabalhando consiste em dezenove arquivos onde cada uma deles contém sentenças pertencentes à língua Portuguesa etiquetadas sintaticamente, ver o exemplo apresentado na Figura 13. A partir destes arquivos e usando expressões regulares, extraímos todas as sentenças do *corpus*, assim como os símbolos não terminais e o vocabulário. As sentenças foram organizadas

[1] O fragmento do *corpus* que temos está constituído de 23.993 sentenças em total.

por tamanho em novos arquivos, essa organização foi feita para um fácil acesso aos dados desde a implementação que calcula o valor da função de verossimilhança.

5.2.3 O cálculo da função de verossimilhança, gradiente e Hessiana

O cálculo da função de verossimilhança, o cálculo do gradiente e da Hessiana, são os maiores desafios nesta proposta, pois eles tem que ser calculados a cada iteração no método de pontos interiores barreira logarítmica primal-dual. Embora o algoritmo *Inside-Outside* utiliza uma maneira eficiente de calcular a probabilidade de uma sentença e sabemos que, em geral, não é eficiente calculá-la somando as probabilidades de todas as possíveis árvores sintáticas, utilizamos uma variação do algoritmo CYK, aproveitamos algumas recursividades que estão descritas no Capítulo 3, assim como as técnicas de programação e programação em paralelo para fazer o cálculo da função de verossimilhança de uma melhor maneira possível, tanto em quantidade de memória, assim como em custo computacional.

Tanto para o cálculo do gradiente como para o cálculo da matriz Hessiana utilizamos o método das diferenças finitas usando uma tolerância de ordem 10^{-8}. Note que só para o cálculo do gradiente temos que avaliar o valor do polinômio para cada um dos $x_i + h$, com $i = 1...505.875$. Esta parte do algoritmo é implementada eficientemente. Uma análise análoga deve ser feita para o cálculo da Hessiana.

5.2.4 Resolução do sistema linear

Temos, a cada iteração, um novo sistema linear para ser resolvido. Estudamos diferentes métodos para a resolução do sistema, assim como o número de condição da matriz e métodos de precondicionadores [TBI97, GVL12].

Dado que a matriz do sistema linear é uma matriz esparsa, mal condicionada, e indefinida, utilizamos um método iterativo precondicionado, especificamente o método de gradiente bi-conjugado estabilizado para matrizes esparsas, com um precondicionador baseado nas entradas da diagonal. Foi usado junto com uma tolerância de $t = 10^{-10}$.

5.3 Estratégias utilizadas nos testes

Uma grande dificuldade é quando, no método de pontos interiores barreira logarítmica primal-dual, os diferentes valores das variáveis com as quais estamos trabalhando atingem valores pequenos causando *underflow* levando com isso problemas estabilidade e arredondamento [TBI97, MAGR97]. Dada a ordem do polinômio do problema, os principais valores que devemos analisar e acompanhar são a função de verossimilhança, o valor do gradiente e o valor da Hessiana. Portanto, nossa proposta é manter os valores

dessas grandezas e os cálculos que os envolvem controlados, tanto para diminuir os erros de arredondamento, como para evitar problemas de *underflow*. Para manter esses valores controlados multiplicamos, quando for preciso, por uma constante C. Basicamente estamos re-escalando os valores numéricos para resolver estes inconvenientes. Um indicador para aumentar o valor de C é quando o valor da função da verossimilhança está perto de atingir *underflow*.

Por outro lado, calibrar o método barreira logarítmica primal-dual requer realizar testes em diferentes valores das constantes e assim achar aqueles valores que logram convergência no método. Na Tabela 9 estão os parâmetros usados com os quais o método atingiu convergência.

	Método de pontos interiores barreira logarítmica primal-dual
Parâmetros	$\epsilon = 10^{-8}$ $\tau = 0,99995$ $\mu = \sigma\frac{\gamma}{2n}$, $\sigma = \frac{1}{\sqrt{2n}}$ $\gamma = x^T z$

Tabela 9 – Parâmetros usados na implementação.

Um terceiro desafio é encontrar um ponto inicial tal que, junto com os parâmetros da Tabela 9, o método atinja convergência. A estratégia utilizada neste caso foi testar pontos iniciais aleatórios num problema menor, quer dizer, utilizamos um sub-conjunto do *corpus* e uma gramática menor $\tilde{G}_p = (\tilde{N}, \tilde{\Sigma}, \tilde{P}, S)$ onde se satisfaz que $\tilde{N} \subset N, \tilde{\Sigma} \subset \Sigma$ e $\tilde{P} \subset P$. As características de $\tilde{G}_p$ estão dadas na Tabela 10. Abusando da língua portuguesa, diremos que $\tilde{G}_p$ é uma sub-gramática da gramática G_p (ver Tabela 8). A amostra Ω para o treino desta sub-gramática, está formada por três sentenças pertencentes ao *corpus* detalhado na Tabela 7, e cujo um comprimento é $n = 4$.

Nro símbolos não terminais $\|N\|$	Nro símbolos terminais $\|\Sigma\|$	Nro de regras $\|P\|$	Símbolo inicial
3	1.200	3.612	S

Tabela 10 – Características da gramática $\tilde{G}_p$.

Uma vez obtida a convergência, criamos um novo problema aumentando o tamanho da amostra[2]Ω. Utilizamos um novo ponto inicial que é uma concatenação do

[2] Aumentando sentenças à amostra, altera, às vezes, no tamanho de Σ e P

ponto inicial do problema anterior e valores aleatórios. Em alguns casos usamos como ponto inicial a concatenação do ponto de convergência do problema anterior com valores aleatórios.

Note que até agora só temos falado de acrescentar o tamanho da amostra, no decorrer dos testes iremos aumentando a quantidade dos símbolos não terminais, com o objetivo de chegar ao número 15 de símbolos não terminais.

5.4 Problemas testados

Lembrando que as características da gramática G_p a ser treinada estão apresentadas na Tabela 8. Com o propósito fazer uma primeira indicação, testamos tempos e capacidades das máquinas utilizadas, assim como o comportamento do método, para determinar, nestas variáveis quão viável é o método que foi construído. Para isso, testamos sub-problemas ou seja, trabalhamos com gramáticas menores e *corpus* menores.

Para um primeiro grupo de sub-problemas, trabalhamos com gramáticas menores extraídas da gramática da Tabela 8, e *corpus* menores extraídos do *corpus* principal (ver Tabela 7). As características destes sub-problemas são apresentadas na Tabela 11.

Adicionalmente, trabalhamos com um segundo grupo de sub-problemas que são apresentados na Tabela 12. Ao invés do primeiro grupo de sub-problemas, neste caso deixamos fixo o número de símbolos não terminais e aumentamos o número de símbolos terminais.

	Nro símbolos não terminais m	Nro símbolos terminais	Nro de regras $\|P\|$	Símbolo inicial
Sub-Problema 1	3	2.500	7.512	S
Sub-Problema 2	5	2.500	12.580	S
Sub-Problema 3	7	2.500	17.752	S
Sub-Problema 4	9	2.500	23.076	S

Tabela 11 – Características da gramática para cada sub-problema do 1 até 4.

	Nro símbolos não terminais m	Nro símbolos terminais	Nro de regras $\|P\|$	Símbolo inicial
Sub-Problema 5	3	4.500	13.512	S
Sub-Problema 6	3	5.500	16.512	S
Sub-Problema 7	3	7.500	22.512	S
Sub-Problema 8	3	7.500	22.512	S

Tabela 12 – Características da gramática para cada sub-problema do 5 até 8.

Observando que o tamanho do sistema linear para ser resolvido em cada problema da Tabelas 12 e da Tabelas 11, é de dimensão $(\|P\| + m) \times (\|P\| + m)$ e onde o número de variáveis do polinômio é de $\|P\|$.

5.5 Resultados

Apresentamos os resultados obtidos a partir dos dois grupos de sub-problemas. Para o primeiro grupo de sub-problemas obtivemos os resultados da Tabela 13, nestes testes utilizamos sentenças pertencentes ao *corpus* de treino, de comprimento $4 \leqslant n \leqslant 5$. O número de sentenças para treinar cada sub-problema está indicado na Tabela 13. O propósito deste primeiro grupo é estudar o comportamento do método e das máquinas em relação ao incremento do número de símbolos não terminais.

Para o segundo grupo deixamos o número de símbolos não terminais fixo, porque o propósito é estudar o método na medida que é aumentado o número de símbolos terminais, assim como o tamanho da amostra e o comprimento das sentenças da amostra. Os resultados podem ser vistos na Tabela 14.

	$\|\Omega\|$	Iterações até convergir	Tempo	Vlr da constante C
Sub-Problema 1	67	9	9m27,401s	10^{25}
Sub-Problema 2	67	9	34m45,830s	10^{20}
Sub-Problema 3	67	9	335m13,625s	10^{15}
Sub-Problema 4	67	9	4558m27,323s	10^{10}

Tabela 13 – Resultados obtidos na Máquina 1, aumentando o tamanho do *corpus* onde o comprimento das sentenças são $n = 4$ e $n = 5$.

	$\|\Omega\|$	Iterações até convergir	Tempo	Vlr da constante C
Sub-Problema 5	143	9	52m36,418s	10^{25}
Sub-Problema 6	161	9	127m19,235s	10^{25}
Sub-Problema 7	388	9	407m19,538s	10^{25}
Sub-Problema 8	310	9	9519m2,029s	10^{15}

Tabela 14 – Resultados obtidos na Máquina 2, aumentando o número de símbolos não terminais e com comprimento de sentenças n, com $4 \leqslant n \leqslant 13$.

5.6 Discussão dos resultados

As estratégias usadas para, determinar o tamanho do passo inicial, evitar a ocorrência de *underflow* e resolução do sistema foram bem sucedidas nos problemas testados e apresentados na Tabela 13 e na Tabela 14.

Como primeiro resultado da Tabela 13 podemos evidenciar que, uma vez obtida a convergência para um problema com número de símbolos não terminais m, podemos aumentar este valor e utilizando a estratégia que foi discutida anteriormente, controlar problemas de *underflow*, e assim obter convergência do método. Na Tabela 14 teve um aumento do tamanho da amostra e do comprimento das sentenças da mesma, havendo um

maior tamanho dos problemas. O método atinge convergência e como no caso do primeiro grupo de testes, o acompanhamento garante que não se gerem problemas de *underflow*, isto basicamente é controlado com o parâmetro C, já discutido anteriormente.

Por outro lado, o método de pontos interiores barreira logarítmica converge num número de iterações pequeno, do qual podemos dizer que é uma boa abordagem para tentar resolver o problema de estimação de parâmetros de uma gramática probabilística livre de contexto, quer dizer para o processo de aprendizado de uma GPLC.

Conclusões

O método de pontos interiores barreira logarítmica primal-dual é projetado como um método viável para estimar as gramáticas probabilísticas livres de contexto. Um dos detalhes relevantes é que o número de iterações para atingir convergência é mantido em nove iterações, quer dizer que o método é estável para diferentes tamanhos tanto da gramática como do *corpus*. Isto é importante porque uma das maiores desvantagem do Método *Inside-Outside* em aplicações práticas é o elevado número de iterações até convergir (em torno das 120 iterações) [Pei99].

Assim mesmo, em aplicações práticas é usado o Método baseado no algoritmo de Viterbi (VS) porque usa menos iterações para convergir. No entanto, os modelos que geram o Método *Inside-Outside* são melhores que aqueles obtidos com o método de VS, isto é explicado porque ao invés do Método *Inside-Outside* o Método VS utiliza só a melhor derivação das sentenças da amostra [Pei99, Ney92]. Sendo importante destacar que neste caso, assim como no Método *Inside-Outside*, estamos consideramos para o processo de estimação todas as possíveis derivações de cada sentença da amostra.

Dos testes, podemos concluir que o método de pontos interiores implementado é robusto e bem comportado na presença de novos dados, sendo um método que pode ser fácil trabalhar.

Perspectivas futuras

Antes de propormos as perspectivas futuras vamos falar brevemente das limitações do trabalho. Por um lado as limitações físicas das máquinas que estamos usando, principalmente para aproveitar a implementação do método que foi feito usando as bibliotecas *Open Multiprocessing* e *Message Passing Interface* (MPI). Por outro lado, um dos *corpus* mais utilizado para testar o Método *Inside-Outside* é o *corpus* do *Wall Street Journal* do projeto *Penn Treebank* [MSM93]. Este é um *corpus* da língua Inglesa, cujos dados são uma coleção de textos de edições do final dos anos 1980 do jornal *Wall Street Journal*. Assim outra das limitações do trabalho é a impossibilidade de utilizar este *corpus* e poder realizar as comparações tanto em termos de tempo como na comparação dos modelos.

Propomos assim as seguintes perspectivas futuras:

- continuar os testes usando a mesma estratégia implementada com o fim de obter um modelo da língua Portuguesa a partir do treino baseado no corpus *Tycho Brahe*,

- usar máquinas com mais capacidade de processamento e que permitam aproveitar as bibliotecas *Open Multiprocessing* e *Message Passing Interface* (MPI) utilizadas na implementação do método de pontos interiores,
- fazer testes usando a mesma estratégia implementada com o fim de obter um modelo da língua Inglesa a partir do treino baseado no corpus do *Wall Street Journal*.
- Definir estratégias para diminuir o custo no cálculo da Hessiana e o gradiente,
- estudar a possibilidade de utilizar, ao invés da função de verossimilhança, o logaritmo da função de verossimilhança, assim como em [LY90].

Referências

[AKW87] Alfred V Aho, Brian W Kernighan, and Peter J Weinberger. *The AWK programming language*. Addison-Wesley Longman Publishing Co., Inc., 1987.

[Bak79] J. K. Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

[BE66] Leonard E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(360-363), 1966.

[BJS10] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. John Wiley and sons, 2010.

[Bri93] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *IN PROCEEDINGS OF THE 31ST ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, pages 259–265. [Col99], 1993.

[BT73] Taylor L Booth and Richard A Thompson. Applying probability measures to abstract languages. *IEEE transactions on Computers*, 100(5):442–450, 1973.

[CDM03] Hinrich Schutze. Christopher D. Manning. *Foundations of statistical natural language processing*. Cambridge, MA : MIT, 2003.

[Cha96] Eugene Charniak. Tree-bank grammars. Technical report, Providence, RI, USA, 1996.

[Cha97] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, pages 598–603. AAAI Press, 1997.

[Che95] Stanley F. Chen. Bayesian grammar induction for language modeling. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, pages 228–235, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics.

[Cho53] Noam Chomsky. Systems of syntactic analysis. *Journal of Symbolic Logic*, 18(3):242–256, September 1953.

[Col99] Michel Collins. *Head-Driven statistical model for natural language parsing*. PhD thesis, University of Pennsylvania., 1999.

[EBTTZ96] A.S. El-Bakry, R.A. Tapia, T. Tsuchiya, and Y Zhang. On the formulation and theory of the newton point-pnterior for nonlinear programming. *Journal of Optimization Theory and Applications*, 1996.

[FM68] AV Fiacco and GP McCormick. Nonlinear programming: Sumt. *Mgmt Sci*, 10:19, 1968.

[Gon12] Jacek Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

[GVL12] Gene H Golub and Charles F Van Loan. Matrix computations, vol. 3, 2012.

[HM97] Ulf Hermjakob and Raymond J. Mooney. Learning parse and translation decisions from examples with rich context, 1997.

[HMU01] Jhon Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, And Computation*. Addison-Wesley, segunda edition, 2001.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[Kas65] Tadao Kasami. An efficient recognition and syntaxanalysis algorithm for context-free languages. Technical report, DTIC Document, 1965.

[KMY89] Masakazu Kojima, Shinji Mizuno, and Akiko Yoshise. A primal-dual interior point algorithm for linear programming. In *Progress in mathematical programming*, pages 29–47. Springer, 1989.

[LY90] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, 4(1):35 – 56, 1990.

[LY08] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer, 3a edition, 2008.

[Ló13] Esther Sofía Mamián López. Método de pontos interiores como alternativa para estimar os parâmetros de uma gramática probabilística livre do contexto. Disertação mestrado, Universidade Estadual de Campinas. Instituto de matemática, Estatística e Computação Científica, 2013.

[MAGR97] Vera Lúcia da Rocha Lopes. Márcia A. Gomes Ruggiero. *Cálculo numérico. Aspectos teóricos e computacionais.* São Paulo, SP : Makron., 1997.

[MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[Ney92] Hermann Ney. Stochastic grammars and pattern recognition. *Speech Recognition and Understanding. Recent Advances*, pages 319–344, 1992.

[NW06] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer Science, 2a edition, 2006.

[Pei99] Joan Andreu Sánchez Peiró. *Estimación de Gramáticas Incontextuales Probabilísticas y su Aplicación en Modelación del Lenguaje*. PhD thesis, Universidad Politécnica de Valencia. Departamento de Sistemas Informáticos y computación, 1999.

[Rou96] Salim Roukos. Language representation. In *Survey of the State of the Art in Human Language Technology*, pages 28–33. 1996.

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*, volume 82. siam, 2003.

[Sip06] Michel Sipser. *Introduction to the theory of computation*. Boston, MA : Cengage Learning/Course Technology, 2006.

[SY06] Wenyu Sun and Ya-Xiang Yuan. *Optimization Theory And Methods Nonlinear Programming*. Springer Science, 1a edition, 2006.

[TBI97] Lloyd N Trefethen and David Bau III. Numerical linear algebra, 1997.

[Vid94] Enrique Vidal. Grammatical inference: An introductory survey. *Lecture Notes in Computer Science*, 1994.

[Wri97] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.

[You66] D. H. Younger. Context-free language processing in time n3. In *7th Annual Symposium on Switching and Automata Theory (swat 1966)*, pages 7–20, Oct 1966.