UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA

GERÊNCIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL - CPGEI

ERNESTO LUIS MALTA RODRIGUES

INFERÊNCIA DE GRAMÁTICAS FORMAIS LIVRES DE CONTEXTO UTILIZANDO COMPUTAÇÃO EVOLUCIONÁRIA COM APLICAÇÃO EM BIOINFORMÁTICA

TESE DE DOUTORADO

CURITIBA
DEZEMBRO - 2007

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

---

TESE

apresentada a UTFPR

como requisito parcial para obtenção do título de

DOUTOR EM CIÊNCIAS

por

ERNESTO LUIS MALTA RODRIGUES

INFERÊNCIA DE GRAMÁTICAS FORMAIS LIVRES DE CONTEXTO UTILIZANDO COMPUTAÇÃO EVOLUCIONÁRIA COM APLICAÇÃO EM BIOINFORMÁTICA

Banca Examinadora:

Presidente e Orientador:

HEITOR SILVÉRIO LOPES (PROF. DR.)	**UTFPR**

Examinadores:

ALAN MITCHELL DURHAM (PROF. DR.)	**USP**
AURORA TRINIDAD RAMIREZ POZO (PROF. DRA.)	**UFPR**
CELSO ANTONIO ALVES KAESTNER (PROF. DR.)	**UTFPR**
RICARDO LÜDERS (PROF. DR.)	**UTFPR**

Curitiba, dezembro de 2007.

ERNESTO LUIS MALTA RODRIGUES

INFERÊNCIA DE GRAMÁTICAS FORMAIS LIVRES DE CONTEXTO UTILIZANDO COMPUTAÇÃO EVOLUCIONÁRIA COM APLICAÇÃO EM BIOINFORMÁTICA

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de "Doutor em Ciências".
Área de Concentração: Informática Industrial.

Orientador: Prof. Dr. Heitor Silvério Lopes

Curitiba
2007

Ficha catalográfica elaborada pela Biblioteca da UTFPR – Campus Curitiba

R696 Rodrigues, Ernesto Luis Malta
Inferência de gramáticas formais livres de contexto utilizando computação evolucionária com aplicação em bioinformática / Ernesto Luis Malta Rodrigues. - Curitiba : [s.n.], 2007.
xviii, 113 p. : il. ; 30 cm

Orientador : Prof. Dr. Heitor Silvério Lopes
Tese (Doutorado) – UTFPR. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2007.
Bibliografia : p. 103-113

1. Bioinformática. 2. Linguagens formais. 3. Inferência (Lógica). 4. Programação genética (Computadores). 5. Computação evolucionária. 6. Reconhecimento de padrões. I. Lopes, Heitor Silvério, orient. II. Universidade Tecnológica Federal do Paraná. Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD : 572.80285

AGRADECIMENTOS

Tive o privilégio de ter Heitor Silvério Lopes como orientador. Suas críticas e sugestões à pesquisa que originou este trabalho foram fundamentais para o seu sucesso.

Sou extremamente grato aos membros da banca que tiveram que ler a tese e souberam, com precisão, identificar as suas falhas textuais e fornecer valiosas sugestões para tornar o seu texto mais claro e conciso.

À minha esposa, Sandra, pelo apoio incondicional que me deu durante todo o doutoramento, trazendo sempre uma palavra de motivação, mesmo nos momentos mais difíceis. Aos meus dois filhos, Marianne e Ernesto, que compreenderam os momentos que tive de me ausentar das brincadeiras.

Aos meus pais, Wallace e Luzia, que tiveram a coragem de assistir a minha defesa e prover todo o apoio emocional necessário, apesar de não entenderem o porquê de tanto sofrimento.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
CE	Computação Evolucionária
CYK	Cocke-Younger-Kasami
DNA	Ácido Desoxirribonucléico
EG	Evolução Gramatical
EM	*Expectation Maximization*
FNC	Forma Normal de Chomsky
GLC	Gramática Livre de Contexto
HMM	*Hidden Markov Models*
KBANN	*Knowledge Based Artificial Neural Network*
PG	Programação Genética
PROSITE	*Protein Functional Pattern Database*
SCFG	*Stochastic Context Free Grammar*
SVM	*Support Vector Machines*

RESUMO

A inferência gramatical lida com o problema de aprender um classificador capaz de reconhecer determinada construção ou característica em um conjunto qualquer de exemplos.

Neste trabalho, um modelo de inferência gramatical baseado em uma variante de Programação Genética é proposto. A representação de cada indivíduo é baseada em uma lista ligada de árvores representando o conjunto de produções da gramática. A atuação dos operadores genéticos é feita de forma heurística. Além disto, dois novos operadores genéticos são apresentados. O primeiro, denominado Aprendizagem Incremental, é capaz de reconhecer, com base em exemplos, quais regras de produção estão faltando. O segundo, denominado Expansão, é capaz de prover a diversidade necessária. Em experimentos efetuados, o modelo proposto inferiu com sucesso seis gramáticas regulares e duas gramáticas livres de contexto: parênteses e palíndromos de quatro letras, tanto o comum quanto o disjunto, sendo superior a abordagens recentes.

Atualmente, modelos de inferência gramatical têm sido aplicados a problemas de reconhecimento de seqüências biológicas de DNA. Neste trabalho, dois problemas de identificação de padrão foram abordados: reconhecimento de promotores e *splice-junction*. Para o primeiro, o modelo proposto obteve resultado superior a outras abordagens. Para o segundo, o modelo proposto apresentou bons resultados.

O modelo foi estendido para o uso de gramáticas *fuzzy*, mais especificamente, as gramáticas *fuzzy* fracionárias. Para tal, um método de estimação adequado dos valores da função de pertinência das produções da gramática é proposto. Os resultados obtidos na identificação de *splice-junctions* comprovam a utilidade do modelo de inferência gramatical *fuzzy* proposto.

ABSTRACT

Grammatical inference deals with the task of learning a classifier that can recognize a particular pattern in a set of examples.

In this work, a new grammatical inference model based on a variant of Genetic Programming is proposed. In this approach, an individual is a list of structured trees representing their productions. Ordinary genetic operators are modified so as to bias the search and two new operators are proposed. The first one, called Incremental Learning, is able to recognize, based on examples, which productions are missing. The second, called Expansion is able to provide the diversity necessary to achieve convergence. In a suite of experiments performed, the proposed model successfully inferred six regular grammars and two context-free grammars: parentheses and palindromes with four letters, including the disjunct one. Results achieved were better than those obtained by recently published algorithms.

Nowadays, grammatical inference has been applied to problems of recognition of biological sequences of DNA. In this work, two problems of this class were addressed: recognition of promoters and splice junction detection. In the former, the proposed model obtained results better than other published approaches. In the latter, the proposed model showed promising results.

The model was extended to support fuzzy grammars, namely the fuzzy fractional grammars. Furthermore, an appropriate method of estimation of the values of the production's membership function is also proposed. Results obtained in the identification of splice junctions shows the utility of the fuzzy inference model proposed.

CAPÍTULO 1

INTRODUÇÃO

A inferência gramatical está presente em várias áreas como uma ferramenta poderosa de reconhecimento de padrões estruturais (HIGUERA, 2005). Basicamente, dado um conjunto de exemplos obtidos de um alfabeto finito que tenha determinada característica, deseja-se obter uma gramática que possa explicar como estes exemplos são formados. Além disto, espera-se que esta gramática seja capaz de reconhecer outros exemplos não apresentados, mas que possuam a mesma característica. Desta forma, a inferência gramatical lida com o problema de aprender um classificador capaz de reconhecer determinada característica em um conjunto bem definido de exemplos.

Diferente de outros métodos de reconhecimento de padrões tais como as Redes Neurais Artificiais (HAYKIN, 1994) ou as Máquinas de Vetores de Suporte (*Support Vector Machines*, SVMs) (VAPNIK, 1998), a inferência gramatical permite gerar classificadores que são compreensíveis e não "caixas pretas".

Recentemente, modelos de inferência gramatical têm sido aplicados também a problemas de análise de seqüências biológicas, tanto de DNA quanto de RNA (SAKAKIBARA, 2005; SEARLS, 2002; DURBIN, EDDY, KROGH *et al*, 1998). Isto se deve ao fato de que a quantidade de dados biológicos tem aumentado continuamente e a análise destas seqüências biológicas tem atraído a atenção de diversos pesquisadores no sentido de identificar características funcionais ou estruturais destas seqüências (BALDI e BRUNAK, 2001). Recentemente, tem se desenvolvido uma área de pesquisa denominada Bioinformática, cujo objetivo é desenvolver e usar técnicas matemáticas e computacionais para ajudar a resolver problemas de Biologia Molecular (COHEN, 2004; SOLTO, LORENA, DELBEM *et al*, 2003; LUSCOMBE, GREENBAUM e GERSTEIN, 2001; SETUBAL e MEIDANIS, 1997). Neste sentido, vários modelos gramaticais têm sido aplicados para reconhecimento de seqüências biológicas, tais como:

- Gramáticas Estocásticas Regulares (KASHIWABARA, VIEIRA, LIMA *et al*, 2007; ROSS, 2002).
- Gramáticas Livres de Contexto (UNOLD, 2007).
- Gramáticas Estocásticas Livres de Contexto (SAKAKIBARA, 2005; KNUDSEN e HEIN, 2003).
- Gramáticas de Cláusulas Definidas (*Definite Clause Grammars*) (SEARLS, 2002).
- TAGs (*Tree Adjoining Grammars*) (CAI, MALMBERG e WU, 2003).

Métodos híbridos também têm sido propostos, tais como gramáticas estocásticas livres de contexto combinadas com Modelos Ocultos de Markov (GARCIA-GOMEZ e BENEDI, 2004).

Em geral, os autores argumentam que a modelagem por gramáticas, comparada com outras técnicas de aprendizado de máquina, tem um potencial maior para analisar seqüências biológicas e compreendê-las. Isto se deve ao fato de que estas seqüências são cadeias de caracteres e possivelmente apresentam padrões sintáticos e estruturais que as caracterizam de alguma forma (GANAPATHIRAJU, BALAKRISHNAN, REDDY *et al*, 2005).

Dentre os diversos tipos de gramáticas existentes, as gramáticas estocásticas livres de contexto[1] (*Stochastic Context Free Grammars*, SCFG) têm sido usadas com mais freqüência para caracterizar seqüências biológicas (SAKAKIBARA, 2005; BALDI e BRUNAK, 2001). Em geral, parte-se de uma gramática genérica capaz de reconhecer qualquer seqüência para, em seguida, ajustar os parâmetros estocásticos do modelo de forma semelhante ao que é feito nos Modelos Ocultos de Markov (*Hidden Markov Models*, HMM). O problema é seu custo computacional da ordem de $O(n^3m^3)$, onde n é o tamanho médio das seqüências e m é o número de símbolos não-terminais da gramática. Para tornar viável o processo de inferência, muitas restrições são impostas, limitando sua generalidade (BALDI e BRUNAK, 2001).

Outros modelos gramaticais podem ser adotados, tais como as gramáticas *fuzzy* (MORDESON e MALIK, 2002). Elas permitem lidar com a incerteza do modelo e tendem a ser robustas com relação a ruídos nos dados, comuns em seqüências biológicas. Porém, não há relatos de pesquisa publicados sobre o uso de gramáticas *fuzzy* em Bioinformática (TORRES

[1] Uma boa referência sobre gramáticas estocásticas livres de contexto pode ser encontrada em (HUANG, ACERO e HON, 2001).

e NIETO, 2006), apenas em reconhecimento de séries temporais, classificação de sinais e reconhecimento de imagens (MORDESON e MALIK, 2002).

De forma semelhante às SCFGs, a inferência de gramáticas *fuzzy* engloba dois passos: definição da estrutura da gramática e cálculo de seus parâmetros. Para a definição da estrutura, os mesmos métodos podem ser utilizados, podendo, inclusive, partir-se de uma gramática genérica. O cálculo de parâmetros da gramática envolve basicamente a obtenção da função de pertinência.

Abordagens baseadas em Computação Evolucionária (CE) tais como Algoritmos Genéticos (AG) (HOLLAND, 1975; GOLDBERG, 1989) e Programação Genética (PG) (KOZA, 1992; BANZHAF, NORDIN, KELLER *et al*, 1998) têm apresentado resultados promissores em diversos problemas de alto custo computacional (ASHLOCK, 2006; FOGEL, 2005; MENON, 2004).

As abordagens em CE se baseiam em uma população de indivíduos que representam soluções candidatas ao problema. Cada indivíduo na população recebe um valor de mérito[2] com base na sua habilidade em resolver o problema. Através de um processo de seleção baseada no mérito, indivíduos são escolhidos para serem alterados pelos operadores genéticos. Estes operadores alteram estes indivíduos de forma a tentar melhorá-los e, com o tempo, produzir uma solução ótima ou, pelo menos, próxima a ótima.

Porém, a maior dificuldade na aplicação da CE aos problemas reside na correta representação dos indivíduos e uma eficiente avaliação de mérito que permita identificar os melhores.

Neste trabalho, um novo método de inferência de gramáticas livres de contexto usando uma variante de Programação Genética (PG) é proposto. Este método se baseia na representação dos indivíduos como listas de árvores. O método inclui uma nova forma de criar a população inicial, bem como uma heurística para atuação dos operadores genéticos de cruzamento e mutação baseada na utilidade de cada produção no reconhecimento de exemplos. Além disto, no modelo proposto, os operadores genéticos atuam de forma controlada para evitar a perda de produções úteis e manter as gramáticas consistentes. Como conseqüência, estes operadores atuam diretamente na substituição de produções inúteis nas gramáticas, favorecendo a convergência. Além disto, dois novos operadores são propostos: um de otimização local (Aprendizagem Incremental) e outro capaz de promover a criação espontânea de novas produções (Expansão).

[2] Esta função de mérito recebe o nome de função de *fitness*.

1.1 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um novo modelo de inferência gramatical baseado em uma variação de Programação Genética capaz de inferir gramáticas livres de contexto.

Como objetivos específicos têm-se:

- Aplicar o modelo proposto na descoberta de gramáticas livres de contexto conhecidas (parênteses e palíndromos);
- Aplicar o modelo proposto a problemas de reconhecimento de padrões em seqüências de DNA e comparar com resultados obtidos por outros métodos de aprendizagem de máquina. Os problemas a serem abordados são identificação de sítios de início de tradução (promotores) e identificação de sítios de *splicing*.

A escolha destes dois problemas de identificação em seqüências de DNA se deve ao fato de existirem bases de dados para *benchmark* disponíveis e diversos métodos de aprendizagem de máquina com resultados sendo aplicados. Isto permite comparar o desempenho do método proposto em relação aos existentes.

1.2 CONTRIBUIÇÕES

As contribuições oferecidas por este trabalho são:

- Um novo modelo baseado em Programação Genética para inferência de gramáticas livres de contexto com operadores genéticos específicos cuja atuação é controlada heuristicamente;
- Adequação do modelo para uso de gramáticas *fuzzy*, particularmente as gramáticas *fuzzy* fracionárias, no reconhecimento de sentenças cuja gramática é desconhecida;
- Proposta de uma métrica adequada para a estimação de parâmetros do modelo gramatical *fuzzy*.

Até a conclusão desta tese, os resultados produziram quatro publicações: duas em congresso internacional (*6th International Conference on Hybrid Intelligent Systems*, HIS'06 e *7th International Conference on Intelligent Systems Design and Applications*, ISDA'07), uma em congresso nacional (I Simpósio Brasileiro de Inteligência Computacional) e uma como

capítulo de livro (*Encyclopedia of Artificial Intelligence*, a ser publicado em maio, 2008 pela IGI Global).

1.3 ORGANIZAÇÃO DO DOCUMENTO

Esta tese está organizada em seis capítulos.

Os Capítulos 2 e 3 apresentam uma revisão da literatura que busca fundamentar a metodologia utilizada neste trabalho. A revisão inclui: linguagens e gramáticas, incluindo as gramáticas *fuzzy*, e Computação Evolucionária, incluindo Algoritmos Genéticos e Programação Genética.

O Capítulo 4 descreve em detalhes o modelo de inferência gramatical livre de contexto proposto baseado em uma variante de Programação Genética. Modificações necessárias na representação dos indivíduos e no comportamento dos operadores genéticos são apresentadas. As modificações que permitem adequar o uso de gramáticas *fuzzy* também são apresentadas.

O Capítulo 5 relata como foram conduzidos os experimentos, os exemplos utilizados e os parâmetros adotados em cada caso. Os resultados obtidos são apresentados e discutidos, tanto na abordagem clássica quanto na *fuzzy*. Os resultados são comparados com outros obtidos por pesquisadores da área.

Finalmente, o Capítulo 6 apresenta as conclusões, ressaltando as principais características do modelo proposto, suas limitações e propostas de continuidade deste trabalho.

CAPÍTULO 2

LINGUAGENS E GRAMÁTICAS

2.1 CONCEITOS BÁSICOS

A Teoria das Linguagens é o estudo de modelos matemáticos que possibilitam a especificação e o reconhecimento das linguagens, suas classificações, estruturas, propriedades e características.

Para definir o que é a Teoria das Linguagens, torna-se necessário definir o que é linguagem. No Dicionário Aurélio consta que linguagem é o "uso da palavra articulada ou escrita como meio de expressão e comunicação entre pessoas". Esta definição não é precisa o suficiente para apoiar um modelo matemático. Há a necessidade de se formalizar algumas definições.

Definição 2.1 Alfabeto

Um alfabeto é um conjunto finito de símbolos. Símbolo pode ser qualquer coisa tais como letras, dígitos, desenhos, comandos etc.

Definição 2.2 Sentença ou Palavra

Uma sentença ou palavra sobre um alfabeto é qualquer seqüência finita formada pelos seus símbolos. A sentença vazia, representada por ε, é uma sentença sem símbolo. Se $\Sigma = \{a,b\}$, então "a", "aa", "b", "ababbba" são exemplos de sentenças de Σ.

Definição 2.3 Tamanho

O tamanho de uma sentença ou palavra *w*, representado por $|w|$, é o número de símbolos que a compõem.

Definição 2.4 Conjunto Potência de um Alfabeto

O conjunto potência de um alfabeto Σ, denotado por Σ^n, é o conjunto de todas as sentenças de tamanho *n* formadas pelos elementos de Σ. Por exemplo, se $\Sigma = \{a,b\}$ então $\Sigma^1 = \{a, b\}$. Da mesma forma, $\Sigma^2 = \{aa, ab, ba, bb\}$, $\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$ e assim sucessivamente.

O conjunto de todas as sentenças (incluindo a nula, ε) de um alfabeto Σ é representado por Σ^*. Portanto, $\Sigma^* = \{\varepsilon\} \cup \Sigma^1 \cup \Sigma^2 \cup ...$ Analogamente, Σ^+ representa o conjunto de todas as sentenças, excetuando-se a sentença nula, isto é, $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Definição 2.5 Linguagem Formal

Uma linguagem formal é um conjunto (possivelmente infinito) de sentenças formadas por um alfabeto, isto é, $L \subseteq \Sigma^*$.

Exemplo 2.1

Considere que $\Sigma = \{a, b\}$. São exemplos de linguagens possíveis sobre Σ:

- $L_0 = \{ab, ba, aab, aba, baa\}$
- $L_1 = \{aa, aba, abba, abbba, abbbba, abbbbba, \ldots\}$
- $L_2 = \{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, \ldots\}$

A linguagem L_0 contém um número finito de sentenças. A linguagem L_1 é infinita e segue o padrão ab^ia com $i \in N$. Da mesma forma, L_2 é infinita e segue o padrão a^ib^i com $i \in N$.

Definição 2.6 Gramática

Uma gramática é um conjunto finito de regras que especificam uma linguagem. Formalmente é uma quádrupla $G = \{\Sigma, N, S, P\}$ onde:

- Σ representa o conjunto finito de símbolos terminais ou alfabeto;
- N representa o conjunto finito de símbolos não-terminais;
- S representa o símbolo inicial onde $S \in N$;
- P representa o conjunto finito de regras de produção.

Definição 2.7 Regras de Produção

As regras de produção definem como podem ser geradas as sentenças de L(G).

Uma regra de produção é representada por $\alpha \rightarrow \beta$, onde $\alpha \in (\Sigma \cup N)^+$ e $\beta \in (\Sigma \cup N)^*$. A sucessiva aplicação das regras de produção permite derivar as sentenças da linguagem representada pela gramática.

Uma seqüência de regras de produção da forma $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, ..., $\alpha \rightarrow \beta_n$ (mesma componente no lado esquerdo) pode ser abreviada como uma única produção na forma: $\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$.

Definição 2.8 Derivação

Uma derivação de uma gramática G = {Σ, N, S, P} é um par da relação denotada por $\Rightarrow$ com domínio em $(\Sigma \cup N)^+$ e imagem em $(\Sigma \cup N)^*$. O par $\alpha \Rightarrow \beta$ significa que é possível derivar (produzir) β a partir de α. Por exemplo, se $z \rightarrow w$ é uma regra de produção de G e $xzy \in (N \cup \Sigma)^*$, então *xwz* é dito diretamente derivável de *xzy* e se representa por $xzy \Rightarrow xwz$. Quando houver mais passos de derivação, estes são explicitados da seguinte forma:

$\Rightarrow^*$ significando zero ou mais passos de derivação;

$\Rightarrow^+$ significando um ou mais passos de derivação;

$\Rightarrow^i$ significando exatos *i* passos de derivação, sendo $i \in N$.

Definição 2.9 Linguagem Gerada

A linguagem gerada pela gramática G, denotada por L(G), é composta por todas as sentenças deriváveis a partir do símbolo inicial S, isto é, $L(G) = \{w \mid w \in \Sigma^*; S \Rightarrow^+ w\}$.

Esta relação estabelece que a linguagem L(G) é formada por combinações de símbolos terminais (alfabeto) que possam ser derivados pela gramática G em um ou mais passos de derivação.

Exemplo 2.2

Considere a gramática G = {Σ, N, S, P}, onde Σ = {0, 1}, N = {S, A}, P = {S → A, S → AS, A → 0, A → 1}. Neste caso, a linguagem L(G) é o conjunto dos números binários. Como exemplo, a derivação do número 101 é como segue:

$$S \Rightarrow AS \Rightarrow 1S \Rightarrow 1AS \Rightarrow 10S \Rightarrow 10A \Rightarrow 101$$

Logo, pode-se afirmar que $S \Rightarrow^+ 101$, portanto $101 \in L(G)$.

2.2 CLASSIFICAÇÃO DE CHOMSKY

Uma gramática pode ser caracterizada pela forma de suas produções. Dependendo do formato das produções, as gramáticas são classificadas em quatro classes (HOPCROFT, MOTWANI e ULLMAN, 2001; CHOMSKY, 1959).

Definição 2.10 Gramáticas Recursivamente Enumeráveis (Tipo 0)

Uma gramática é dita recursivamente enumerável tiver as suas regras de produção da forma apresentada na Equação 1.

$$\alpha \rightarrow \beta \quad \text{sendo } \alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^* \text{ e } \beta \in (N \cup \Sigma)^+ \qquad (1)$$

Neste tipo de gramática, qualquer combinação de símbolos, tanto terminais como não terminais, é permitida em qualquer um dos lados da produção. A única restrição é que deve haver ao menos um não-terminal no lado esquerdo de cada produção. Além disto, torna-se necessária a presença de pelo menos uma produção $\alpha \rightarrow \beta$ onde $|\alpha| > |\beta|$. Tal produção pode ser usada para "remover" símbolos, por exemplo, a regra de produção $\alpha\beta\delta \rightarrow \alpha\delta$ remove β do contexto $\alpha\beta\delta$ (MENEZES, 2001).

Definição 2.11 Gramáticas Sensíveis a Contexto (Tipo 1)

Uma gramática é dita sensível a contexto se tiver as suas regras de produção da forma apresentada na Equação 2.

$$\alpha \rightarrow \beta \quad \text{sendo } \alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^* \text{ e } \beta \in (N \cup \Sigma)^+ \qquad (2)$$

Para diferenciá-la das recursivamente enumeráveis, impõem-se a restrição que todas as regras de produção $\alpha \rightarrow \beta$, $|\alpha| \geq |\beta|$.

Definição 2.12 Gramáticas Livre de Contexto (Tipo 2)

Uma gramática é dita livre de contexto se tiver as suas regras de produção da forma apresentada na Equação 3.

$$\alpha \rightarrow \beta \quad \text{sendo } \alpha \in N \text{ e } \beta \in (N \cup \Sigma)^+ \qquad (3)$$

Neste tipo de gramática, qualquer combinação de símbolos, tanto terminais como não-terminais, é permitida no lado direito das produções. Porém, o lado esquerdo deve conter um símbolo não-terminal.

Definição 2.13 Gramáticas Regulares (Tipo 3)

Uma gramática regular apresenta duas formas: regular à direita e regular à esquerda. Uma gramática regular à direita (Equação 4) apresenta o lado direito de qualquer produção consistindo de um ou mais símbolos terminais, opcionalmente seguidos por um símbolo não-terminal. Seu lado esquerdo é sempre um símbolo não-terminal.

$$\alpha \rightarrow \beta\delta \quad \text{sendo} \quad \alpha \in N, \quad \beta \in \Sigma^{+} \quad \text{e} \quad \delta \in N^{*} \qquad (4)$$

No caso das gramáticas regulares à esquerda (Equação 5), o lado direito de qualquer produção consiste de um ou mais símbolos terminais, opcionalmente precedidos por um símbolo não-terminal. Seu lado esquerdo é sempre um símbolo não-terminal.

$$\alpha \rightarrow \delta\beta \quad \text{sendo} \quad \alpha \in N, \quad \beta \in \Sigma^{+} \quad \text{e} \quad \delta \in N^{*} \qquad (5)$$

Ao analisar o formato das regras de produção de cada tipo de gramática, verifica-se que as gramáticas regulares (tipo 3) são um subconjunto das gramáticas livre de contexto (tipo 2) que, por sua vez, são um subconjunto das gramáticas sensíveis ao contexto (tipo 1). Da mesma forma, as gramáticas sensíveis ao contexto (tipo 1) são um subconjunto das gramáticas recursivamente enumeráveis (tipo 0). A hierarquia das gramáticas de Chomsky está na Figura 1.

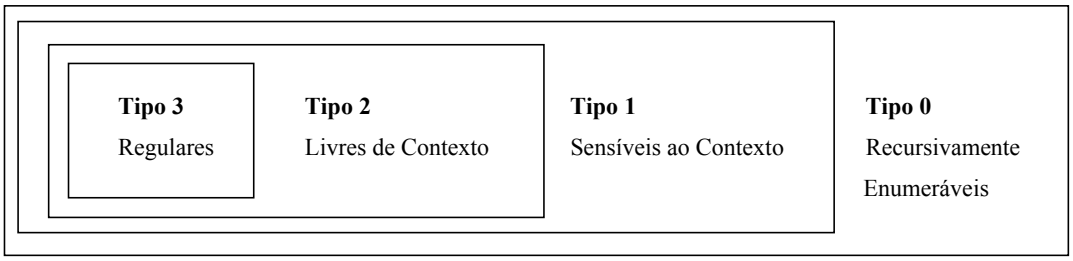

Figura 1: Hierarquia das gramáticas.

Uma linguagem L(G) é do tipo *k* se puder ser gerada por uma gramática do tipo *k*. Então, por exemplo, uma linguagem é dita livre de contexto se uma gramática livre de contexto puder ser usada para definí-la.

2.3 LINGUAGEM E GRAMÁTICA FUZZY

A teoria dos conjuntos *fuzzy* é um meio de especificar o quanto um objeto satisfaz a uma descrição vaga. Como exemplo, considere a expressão: "João é alto". Esta expressão será verdadeira se João tiver 1,78 m? A maioria das pessoas hesitaria em responder "sim" ou "não", preferindo dizer "talvez". É importante notar que isto não é uma questão de incerteza, pois se sabe a altura de João. A questão é que o termo lingüístico "alto" não se refere a uma demarcação nítida de objetos em duas classes. Portanto, diferentemente da abordagem clássica de conjuntos em que um elemento pertence ou não, os conjuntos *fuzzy* são funções que mapeiam cada elemento a um grau de pertinência no intervalo [0, 1], onde o valor 0 indica uma completa não-pertinência ao conjunto e o valor 1, completa pertinência (ALMEIDA e EVUSUKOFF, 2003).

Definição 2.14 Conjunto *Fuzzy*

Um conjunto *fuzzy* é caracterizado por uma função de pertinência μ que associa a cada elemento de um domínio ou universo U um número real no intervalo [0,1], isto é, μ: U $\rightarrow$ [0, 1] (PEDRYCZ e GOMIDE, 1998).

A representação de um conjunto *fuzzy* A em U é feita através de pares ordenados de seus elementos com os seus respectivos graus de pertinência $\mu_A(x)$ da seguinte forma: $A = \{(\mu_A(x) / x) \mid x \in U\}$. Por exemplo, A = {0,0/0; 0,1/1; 0,5/2; 0,8/3; 1,0/4; 1,0/5} é um possível conjunto *fuzzy* do universo $U = \{x \in \mathbf{N} \mid 0 \leq x \leq 5\}$.

Por outro lado, quando o universo de discurso é contínuo ou possui uma grande quantidade de elementos, a forma usual de representação é o gráfico de sua função de pertinência, chamado de diagrama de Hassi-Euler (ALMEIDA e EVUSUKOFF, 2003).

Definição 2.15 Gramática *Fuzzy*

Uma gramática *fuzzy* é definida pela sêxtupla G = {Σ, N, S, P, J, μ} onde (MORDESON e MALIK, 2002):

1. Σ representa o conjunto finito de símbolos terminais;
2. N representa o conjunto finito de símbolos não-terminais;
3. S representa o símbolo inicial;
4. P representa o conjunto finito de produções;
5. J é um conjunto *fuzzy* que referencia os elementos de P;
6. μ é a função de pertinência fuzzy onde μ: J → [0, 1].

Definição 2.16 Linguagem *Fuzzy* Gerada

De forma semelhante a gramáticas clássicas, a linguagem *fuzzy* gerada pela gramática *fuzzy* G, denotada por L(G), é composta por todas as sentenças que tenham grau de pertinência maior que zero, isto é, $L(G) = \{w \mid w \in \Sigma^*; \mu_G(w) \geq 0\}$.

Uma sentença *w* pertence a uma linguagem descrita pela gramática *fuzzy* G se existir uma forma sentencial, sendo o grau de pertinência dado pela Equação 6 onde *n* é o número de derivações possíveis e t_k é o tamanho da *k*-ésima derivação.

$$\mu_G(w) = \max_{1 \leq k \leq n} \left\{ \min_{1 \leq i \leq t_k} \mu\left(p_i^k\right) \right\} \qquad (6)$$

Exemplo 2.3

Considere que a gramática *fuzzy* G seja definida pelos conjuntos Σ = {a, b}, N = {S, A, B, C, D}, S é o símbolo inicial e P, J e μ definidos como:

p_1: A → a $\mu(p_1) = 1$
p_2: B → b $\mu(p_2) = 0{,}9$
p_3: C → SB $\mu(p_3) = 0{,}7$
p_4: D → AS $\mu(p_4) = 0{,}9$
p_5: S → AB $\mu(p_5) = 1$
p_6: S → AC $\mu(p_5) = 0{,}8$
p_7: S → DB $\mu(p_5) = 0{,}6$

Para a seqüência “aabb”, uma possível derivação é:

$$S \Rightarrow^{0,8} AC \Rightarrow^{1,0} aC \Rightarrow^{0,7} aSB \Rightarrow^{1,0} aABB \Rightarrow^{1,0} aaBB \Rightarrow^{0,9} aabB \Rightarrow^{0,9} aabb$$

O valor de $\mu_G(w)$ para a derivação apresentada vale *min* (0,8; 1,0; 0,7; 0,9) = 0,7. Para a seqüência apresentada existem ainda mais uma derivação possível, a saber:

$$S \Rightarrow^{0,6} DB \Rightarrow^{0,9} ASB \Rightarrow^{1,0} aSB \Rightarrow^{1,0} aABB \Rightarrow^{1,0} aaBB \Rightarrow^{0,9} aabB \Rightarrow^{0,9} aabb$$

Nesta segunda derivação possível, $\mu_G(w)$ vale *min* (0,6; 0,9; 1,0) = 0,6. O valor final de $\mu_G(w)$ é o maior valor obtido dentre todas as derivações possíveis. Neste caso, $\mu_G(w)$ = *max*(0,7; 0,6) = 0,7.

Exemplo 2.4

Considere que a gramática *fuzzy* G seja definida pelos conjuntos $\Sigma = \{a, b\}$, N = {S, A, B}, S é o símbolo inicial e P, J e μ definidos como:

p_1: $S \rightarrow AB$ $\quad \mu(p_1) = 1$

p_2: $A \rightarrow a$ $\quad \mu(p_2) = 1$

p_3: $B \rightarrow b$ $\quad \mu(p_3) = 1$

p_4: $A \rightarrow aAB$ $\quad \mu(p_4) = 0,9$

p_5: $A \rightarrow aB$ $\quad \mu(p_5) = 0,5$

p_6: $A \rightarrow AB$ $\quad \mu(p_6) = 0,5$

p_7: $A \rightarrow B$ $\quad \mu(p_7) = 0,2$

A linguagem gerada por esta gramática *fuzzy* consiste de sentenças na forma $a^n b^m$. Dependendo dos valores de *n* e *m*, determinadas produções são usadas. Basicamente todas as derivações começam com a produção $S \rightarrow AB$, pois é a única produção que parte do símbolo inicial S. Se forem usadas unicamente as produções p_2 e p_3, a pertinência será um, pois as produções têm o valor de μ iguais a um. Neste caso é gerada a sentença “ab”. Ao se incluir a produção p_4 ($A \rightarrow aAB$), é possível gerar sentenças com qualquer número de a’s e apenas um b. Como $\mu(p_4) = 0,9$, sentenças do tipo $a^n b$ têm $\mu = 0,9$. Para outras sentenças, a pertinência μ será dada por:

$$\mu(a^n b^m) = \begin{cases} 1 & \text{se } m = n = 1 \\ 0{,}9 & \text{se } m = 1 \text{ e } n > 1 \\ 0{,}5 & \text{se } m > 1 \text{ e } n > 1 \\ 0{,}2 & \text{se } m = 2 \text{ e } n = 0 \\ 0 & \text{caso contrário} \end{cases}$$

Um aspecto interessante das gramáticas *fuzzy* reside no fato de que elas podem ser usadas para gerar linguagens com o uso de limiares (*thresholds*).

Definição 2.17 Linguagem *Fuzzy* Gerada com uso de limiar

A linguagem *fuzzy* gerada com uso de um limiar c, sendo c um número real maior que zero, pela gramática *fuzzy* G, denotada por $L_c(G)$, é composta por todas as sentenças que tenham grau de pertinência maior que c, isto é, $L(G) = \{w \mid w \in \Sigma^*; \mu_G(w) \geq c\}$ (MORDESON e MALIK, 2002).

O uso de limiar permite adequar uma gramática *fuzzy* a tarefa de classificação, onde o seu valor é ajustado com base nos exemplos apresentados.

2.4 RECONHECIMENTO GRAMATICAL

Verificar se uma determinada palavra pertence a uma linguagem é uma das principais questões relacionadas com o estudo de Linguagens Formais (MENEZES, 2001). Para cada tipo de linguagem existe um reconhecedor associado, conforme a Tabela 1.

Tabela 1: Reconhecedores.

Linguagem	**Reconhecedor**
Tipo 3 ou Regulares	Autômato Finito
Tipo 2 ou Livres de Contexto	Autômato com Pilha
Tipo 1 ou Sensíveis a Contexto	Autômato Limitado Linearmente
Tipo 0 ou Recursivamente Enumeráveis	Máquina de Turing

Para o caso das linguagens livres de contexto, outros reconhecedores de melhor eficiência podem ser aplicados, desde que se imponham determinadas restrições na construção das gramáticas que geram a linguagem (AHO, LAM, SETHI *et al*, 2007). É o caso do algoritmo de Cocke-Younger-Kasami, CYK (YOUNGER, 1967; KASSAMI, 1963).

O algoritmo de Cocke-Younger-Kasami (CYK) permite gerar todas as árvores de derivação possíveis. É construído sobre uma gramática na Forma Normal de Chomsky (FNC).

Definição 2.18 Árvore de Derivação

A representação das derivações de uma sentença em uma gramática livre de contexto pode ser feita em forma de árvore, denominada Árvore de Derivação. A sua construção é feita como segue:

- A raiz da árvore é o símbolo inicial da gramática (S);
- Os nós internos são símbolos não-terminais. Se A é um nó interno e a_1, a_2,..., a_n são filhos de A, então $A \rightarrow a_1a_2...a_n$ é uma regra de produção da gramática;
- Um nó folha é um símbolo terminal.

Exemplo 2.5

Considere a gramática $G = \{\Sigma, N, S, P\}$, onde $\Sigma = \{a, b\}$, $N = \{S, A, B\}$, $P = \{A \rightarrow a; B \rightarrow b; S \rightarrow AB; S \rightarrow AS\}$. A derivação da sentença "aab" é como segue: $S \Rightarrow AS \Rightarrow aS \Rightarrow aAB \Rightarrow aaB \Rightarrow aab$. A sua árvore de derivação está na Figura 2.

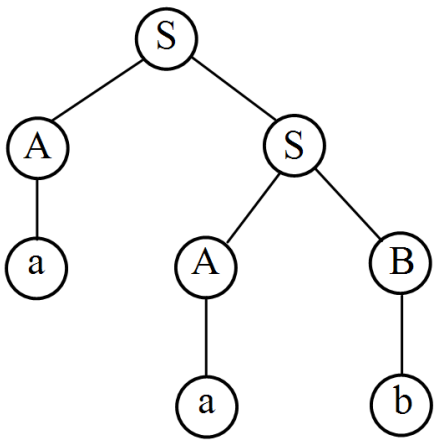

Figura 2: Exemplo de árvore de derivação.

Definição 2.19 Forma Normal de Chomsky

Uma gramática livre de contexto G está na Forma Normal de Chomksy (FNC) se todas as suas produções são da forma $\alpha \rightarrow \beta\delta$ ou $\alpha \rightarrow \sigma$, onde $\alpha, \beta, \delta \in N$ e $\sigma \in \Sigma^+$.

Teorema 2.1

Para qualquer gramática livre de contexto G que não gere a sentença nula ε, existe uma gramática G' na Forma Normal de Chomsky que gera a mesma linguagem, isto é, L(G) = L(G') (HOPCROFT, MOTWANI e ULLMAN, 2001).

Suponha que $w = a_1a_2..a_n$ é uma sentença a ser verificada. O algoritmo CYK é composto pelos dois passos seguintes, onde V_{rs} representa as células de uma matriz triangular de derivação que é ilustrada na Figura 3.

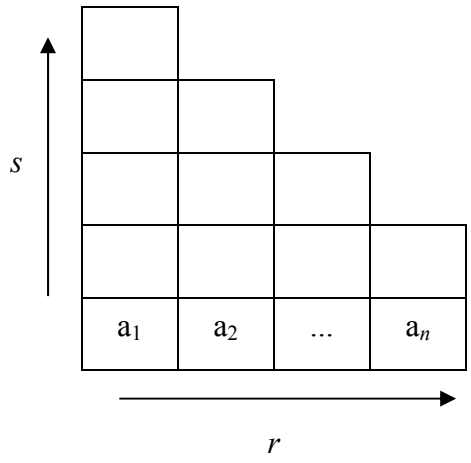

Figura 3: Tabela triangular de derivação.

Passo 1: Regras de produção que geram diretamente símbolo terminal.

```
Para r variando de 1 até n faça
    V_r1 = {A | A → a_r ∈ P}
Fim Para
```

Passo 2: Regras de produção que geram dois símbolos não-terminais.

```
Para s variando de 2 até n faça
    Para r variando de 1 até n-s+1 faça
        V_rs = ∅
        Para k variando de 1 até s - 1 faça
            Se B ∈ V_rk e C ∈ V_(r+k)(s-k) e A → BC ∈ P
                Então
                    V_rs = V_rs ∪ {A | A → BC ∈ P, B ∈ V_rk e C ∈ V_(r+k)(s-k) }
            Fim Se
        Fim Para
    Fim Para
Fim Para
```

É possível perceber que:

- Os vértices V_{rk} e $V_{r+k\ s-k}$ são as raízes das subárvores de V_{rs};
- Se uma célula for vazia significa que ela não gera nenhuma subárvore.

A entrada *w* é aceita somente se, ao final, o símbolo inicial estiver presente no vértice V_{1n} (raiz da árvore de derivação).

Exemplo 2.6

Considere a gramática G = {Σ, N, S, P} onde N = {S, A}, ∑ = {a, b} e P = {A → a; S → b; S → AA; S → AS; A → AS; A → AS}. A matriz triangular de derivação para a sentença de entrada "abaab" é ilustrada na Figura 4. Como o símbolo inicial S está presente na raiz da árvore de derivação, a entrada é aceita.

S, A				
S, A	S, A			
S, A	S	S, A		
S, A	A	S	S, A	
A	S	A	A	S
a	b	a	a	b

Figura 4: Tabela triangular de derivação para a seqüência "abaab".

2.5 INFERÊNCIA GRAMATICAL

A inferência gramatical é a tarefa de aprender uma gramática a partir de sentenças (RUSSEL e NORVIK, 2003, HIGUERA, 2005). Portanto, para um conjunto de exemplos positivos (sentenças geradas pela gramática) E^+ e, eventualmente, um conjunto de exemplos negativos (sentenças não geradas pela gramática) E^-, deseja-se encontrar um conjunto de produções que reconheça os exemplos positivos e rejeite os negativos. Além disto, tal conjunto de produções deve ser capaz de reconhecer outros exemplos positivos e rejeitar outros exemplos negativos não apresentados no treinamento.

2.5.1 Inferência de Gramáticas Formais

Os tipos de gramáticas formais usadas principalmente para inferência são as regulares e livres de contexto. Para a inferência de gramáticas regulares existem diversos algoritmos que se baseiam em autômatos finitos determinísticos (AFD), uma vez que o mapeamento entre uma gramática regular e um AFD é possível (HOPCROFT, MOTWANI e ULLMAN, 2001). Como o foco deste trabalho está nas gramáticas livres de contexto, apenas os principais métodos para esta classe de gramáticas serão brevemente descritos.

a) Sequitur

O método Sequitur (NEVILL-MANNING e WITTEN, 1997) é capaz de produzir uma gramática livre de contexto que cubra uma determinada seqüência e somente esta. Inicialmente idealizado para compressão de longos textos, também pode ser usado para reconhecer a estrutura de um documento. O princípio do método reside na idéia de que uma boa gramática é uma gramática compacta. Para tal, impõe duas restrições:

i. Nenhum par de símbolos adjacentes deve aparecer mais de uma vez na gramática;
ii. Toda regra deve ser usada pelo menos duas vezes.

A primeira restrição garante que se o par de símbolos AB aparecer no lado direito de várias produções, então deve ser criada a regra $C \rightarrow AB$ e substituir todas as ocorrências de AB por C. Já a segunda restrição evita produções inúteis, isto é, se um símbolo não-terminal

C aparecer apenas uma vez na gramática, então deve-se eliminar a produção para C e substituir seu único uso pelo seu lado direito da produção.

Estas duas restrições são aplicadas em uma busca gulosa que varre o texto de entrada da esquerda para direita, construindo de forma incremental uma gramática à medida que prossegue e impondo as restrições sempre que possível. A grande vantagem deste método reside no fato do algoritmo ser de tempo linear. Porém, o método é capaz de gerar apenas a gramática livre de contexto que gera exclusivamente a sentença informada.

Melhorias neste método foram propostas por Charikar e seus colegas para obter não apenas a gramática que somente gera a seqüência, mas que esta gramática seja necessariamente a mínima (CHARIKAR, LEHMAN, LIU *et al*, 2005).

Conforme mencionado, apesar de eficiente, o método é capaz de encontrar apenas a gramática que gera exclusivamente a sentença informada. Desta forma, há a necessidade de se estender o método de forma a gerar uma gramática que cubra mais de uma sentença. A soma de duas ou mais gramáticas necessita de um operador especial de composição que depende da forma com que as gramáticas são geradas (WINTNER, 2003). Além disto, há a necessidade de se garantir certo nível de generalização na gramática obtida sob pena de *overfitting*, isto é, a gramática só reconhece os exemplos positivos apresentados no treinamento.

Exemplo 2.7

A geração do conjunto de regras de produção para a sentença “abcdbcabc” está mostrada na Tabela 2.

Tabela 2: Exemplo de aplicação do Sequitur.

Passo	Entrada	Gramática Gerada	Observações
1	a	$S \rightarrow a$	
2	ab	$S \rightarrow ab$	
3	abc	$S \rightarrow abc$	
4	abcd	$S \rightarrow abcd$	
5	abcdb	$S \rightarrow abcdb$	
6	abcdbc	$S \rightarrow abcdbc$	bc ocorre duas vezes
		$S \rightarrow aAdA$; $A \rightarrow BC$	
7	abcdbca	$S \rightarrow aAdAa$; $A \rightarrow bc$	
8	abcdbcab	$S \rightarrow aAdAab$; $A \rightarrow bc$	
9	abcdbcabc	$S \rightarrow aAdAabc$; $A \rightarrow bc$	bc ocorrendo novamente
		$S \rightarrow aAdAaA$; $A \rightarrow bc$	aA duas vezes
		$S \rightarrow BdAB$; $A \rightarrow bc$; $B \rightarrow aA$	Gramática Encontrada!

b) eg-GRIDS

Método proposto por Petasis, Paliouras, Spyropoulos e Halatsis (2004) que tem por principal característica usar apenas exemplos positivos. O seu funcionamento é o seguinte: inicialmente o algoritmo gera uma gramática que contém $n + 1$ símbolos não-terminais cada um gerando um dos n exemplos. A seguir, usa as cinco operações a seguir na análise das produções para generalizar a gramática:

i. Juntar símbolos não-terminais

Junta dois símbolos não-terminais em um único símbolo não-terminal, conseqüentemente substituindo todas as suas ocorrências nas outras produções.

ii. Criar um símbolo não-terminal

Cria um novo símbolo não-terminal X e uma produção começando por ele e tendo uma seqüência de dois ou mais símbolos não-terminais como derivação.

iii. Criar um símbolo não-terminal "opcional"

Duplica a produção criada pela operação (ii) e adiciona um símbolo não-terminal já existente no fim do corpo da produção, tornando este símbolo opcional.

iv. Detectar alinhamento central embutido;

Tenta capturar o alinhamento central embutido, através da localização do 4-gram[3] mais freqüente na forma AABB. Uma vez localizado, este operador cria um novo símbolo não-terminal X tal como a operação (ii). Contudo, assumindo que este 4-gram foi criado com base no alinhamento central embutido envolvendo X, este operador cria mais uma produção na forma $X \rightarrow AAXBB$ e substitui todas as ocorrências que casam com o padrão.

v. Substituir corpo de produções.

Este operador verifica se o corpo da produção P está contido nos corpos de outras regras de produção. Neste caso, todas as ocorrências do corpo de R nas outras regras de produção são substituídas pelo símbolo à esquerda da regra de produção P.

Estas operações são aplicadas juntamente com uma estratégia de busca baseada em alvo (*beam search*) e em algoritmos genéticos (AG).

c) Synapse

Proposto por Nakamura e Matsumoto (2002), o método Synapse (*Synthesis by Analyzing Positive String Examples*) monta iterativamente o conjunto de produções de uma gramática na Forma Normal de Chomsky com base em exemplos positivos.

Para tal, o método se baseia em dois passos. No primeiro, inicia com um conjunto de produções fornecido inicialmente pelo usuário (podendo ser vazio) e um conjunto denominado TS (*test set*) inicialmente vazio. Em seguida, executa-se o algoritmo CYK para cada exemplo positivo, sendo que ao adicionar a produção $\alpha \rightarrow \beta\delta$ ao elemento V_{rs}, o par (β, δ) é adicionado ao conjunto TS. Na Figura 5 apresenta-se o passo 2 do algoritmo CYK com as operações adicionadas pelo Synapse em negrito.

[3] Um *n*-gram de uma sentença *w* é qualquer subseqüência de *w* contendo *n* símbolos consecutivos (BROWN, MERCER, PIETRA e LAI, 1992).

Ao final deste primeiro passo, o conjunto TS conterá pares de símbolos não-terminais que foram usados durante o reconhecimento dos exemplos positivos.

```
TS = {}
Para s variando de 2 até n faça
    Para r variando de 1 até n-s+1 faça
        Para k variando de 1 até s - 1 faça
            Se B ∈ V_rk e C ∈ V_(r+k)(s-k) e A → BC ∈ P
                Então
                    V_rs = V_rs ∪ {A | A → BC ∈ P, B ∈ V_rk e C ∈ V_(r+k)(s-k) }
                    TS = TS ∪ {(B, C)}
            Fim Se
        Fim Para
    Fim Para
Fim Para
```

Figura 5: Algoritmo Synapse.

O segundo passo se responsabiliza pela criação das regras de produção que estão faltando para a cobertura de todos os exemplos positivos. Isto é feito através da adição de produções $\alpha \rightarrow \beta\delta \notin P$ onde $(\beta, \delta) \in TS$ e $\alpha \in N$.

Uma desvantagem deste método reside no fato de que apenas exemplos positivos são usados e necessita de um conjunto relativamente grande de exemplos para obter sucesso. Para contornar esta desvantagem, Nakamura (2004) fez modificações no Synapse para permitir o uso de exemplos negativos adotando as seguintes heurísticas:

i. Limitação na forma das regras de produção geradas

   Quando é gerada a regra de produção $\alpha \rightarrow \beta\delta$, permite-se que $\alpha \notin N$, isto é, a geração de um novo símbolo não-terminal. Porém, o passo 2 do algoritmo não encerra até que uma produção contendo α no seu lado direito seja também gerada.

ii. Retrocesso inteligente

   Considere o caso em que após a adição de diversas regras de produção para a cobertura de um exemplo positivo, alguns exemplos negativos venham a ser cobertos. Quando isto ocorre, a última regra de produção gerada é retirada e uma

nova é produzida. Caso ainda ocorra a cobertura de exemplo negativo, as duas últimas regras de produção geradas são retiradas e uma nova é produzida e assim sucessivamente.

iii. Uso de memória *hash* para evitar busca repetitiva

Nos experimentos conduzidos por Nakamura (2004), descobriu-se que há geração repetida de regras de produção iguais durante a busca. Para evitar que a busca gere desnecessariamente regras de produção já processadas, um mecanismo de memória *hash*[4] é usado. Este mecanismo reduz o número de regras de produção geradas na busca, porém, tendo o custo de armazenamento e pesquisa na memória *hash*.

Apesar das heurísticas terem melhorado sensivelmente o Synapse, não é possível gerar mais do que 14 regras de produção devido ao seu custo computacional (NAKAMURA, 2004).

2.5.2 Inferência de Gramáticas Fuzzy

A inferência de uma gramática *fuzzy* é semelhante à de uma gramática estocástica, tendo dois aspectos: determinação da sua estrutura e cálculo das pertinências das produções. Normalmente, a estrutura da gramática é determinada *a priori* sem a utilização de nenhum método em particular. Para o cálculo das pertinências, o método normalmente utilizado é o seguinte: dado o conjunto de sentenças válidas, a pertinência inicial de cada produção $p_i \in P$ é dada pela Equação 7, onde *N* é o número total de sentenças e *n* o número de sentenças em que a *i*-ésima produção foi usada (KUBICEK e ZENDULKA, 2002).

$$f_i(G) = \frac{n}{N} \qquad (7)$$

Como será demonstrado no Capítulo 4, tal forma de cálculo não se mostra adequada para inferência de gramáticas livres de contexto, sendo que, neste trabalho, propõe-se uma métrica mais adequada.

[4] Detalhes sobre como funcionam as memórias *hash* podem ser obtidas em (CORMEN, LEISERSON, RIVEST e STEIN, 2001).

CAPÍTULO 3

COMPUTAÇÃO EVOLUCIONÁRIA

3.1 INTRODUÇÃO

A teoria da seleção natural das espécies proposta por Charles Darwin (1859) defende que, na natureza, sobrevivem os seres que têm a melhor capacidade de se adaptarem ao meio ambiente. Estes indivíduos passarão suas características genéticas para seus descendentes e, ao final de muitas gerações, obtém-se uma população de indivíduos com estas características selecionadas naturalmente (QUAMMEN, 2007).

Em vez de uma população de seres vivos, na Computação Evolucionária (CE) tem-se uma população de indivíduos que representam soluções candidatas a um determinado problema. O objetivo é obter uma solução através da evolução destes indivíduos. Para isto, começa-se com uma população inicial aleatória que representa uma amostra inicial do espaço de busca. Em seguida, geração após geração, aplica-se os operadores genéticos para simular o processo evolutivo até que um determinado critério de término seja satisfeito (ASHLOCK, 2006; MENON, 2004).

Várias técnicas de CE têm sido propostas por vários pesquisadores em um conjunto relativamente grande de problemas. Porém, os ingredientes principais têm sido sempre:

a) Estabelecer a representação de cada indivíduo;
b) Definir um método de avaliação da qualidade de cada indivíduo (*fitness*);
c) Construir uma população destes indivíduos;
d) Submetê-los a um método de seleção probabilístico baseado na qualidade;
e) Aplicar probabilisticamente operadores genéticos que permitam modificar os indivíduos selecionados;
f) Repetir (d) e (e) até que um determinado critério de término seja satisfeito.

3.1.1 Avaliação dos Indivíduos

A avaliação de cada indivíduo, conhecida por avaliação de *fitness*, deve ser capaz de diferenciar os melhores indivíduos dos piores, pois ela representa a heurística que direciona o algoritmo de CE na busca pela solução. O ideal é ter uma função que associe, a cada indivíduo possível no espaço de busca, um único valor para permitir a sua diferenciação dos demais.

Definição 3.1 Avaliação de *fitness*

A avaliação de *fitness* é caracterizada por uma função f que associa a cada elemento do espaço de busca Ω um número real positivo, isto é, f: $\Omega \rightarrow$ R.

3.1.2 Seleção

Para que o processo evolutivo ocorra, um conjunto de indivíduos da população atual deve ser escolhido para sofrer a ação de um ou mais operadores genéticos e substituir a população atual. Isto é feito através de um método de seleção.

O método de seleção deve favorecer a escolha dos melhores indivíduos, mas não deve se limitar a somente escolhê-los. Atualmente existem diversos métodos de seleção diferentes, tais como Seleção Proporcional, Torneio Estocástico, Truncamento, Ordenamento Linear e Ordenamento Exponencial. A seguir, são definidos os dois métodos mais comuns (BACK, FOGEL e MICHALEWICZ, 2000):

Definição 3.2 Seleção Proporcional

A seleção proporcional inicialmente associa a cada indivíduo da população uma probabilidade calculada pela Equação 8.

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \qquad (8)$$

Em seguida, um procedimento de escolha é feito através do pseudocódigo apresentado na Figura 6, onde $p(i)$ é a probabilidade do i-ésimo indivíduo e `random()` é uma função que gera um valor no intervalo [0, 1] seguindo uma distribuição uniforme.

```
i = 1
S = p(i)
p = random()
Enquanto S < p Faça
    i = i + 1
    S = S + p(i)
Fim Enquanto
Retorna i
```

Figura 6: Algoritmo da Seleção Proporcional.

Este processo de seleção se assemelha a uma roleta onde cada indivíduo ocupa uma fatia proporcional ao seu valor de *fitness*.

Definição 3.3 Seleção por Torneio Estocástico

Dado um valor inteiro *k*, conhecido como tamanho do torneio, a seleção por torneio estocástico escolhe uniformemente *k* indivíduos da população. Destes indivíduos, o que apresentar o melhor valor de *fitness* é o escolhido. Caso exista mais de um, escolhe-se aleatoriamente um deles.

3.1.3 Operadores Genéticos

Os operadores genéticos podem ser de dois tipos: sexuados ou assexuados. A diferença entre eles está no fato de que, no operador sexuado usam-se dois indivíduos que trocam partes entre si, gerando dois novos indivíduos. No operador assexuado, um único indivíduo gera um descendente. Usualmente, o operador sexuado é denominado recombinação ou cruzamento e o assexuado, de mutação (BACK, FOGEL e MICHALEWICZ, 2000).

A aplicação dos operadores genéticos é feita de forma probabilística, isto é, não-determinística. As probabilidades de cruzamento e mutação, juntamente com o tamanho da população, são parâmetros que devem ser previamente definidos. A sua escolha pode favorecer ou não o sucesso da abordagem. Atualmente, há várias pesquisas que tentam buscar auto-adaptação de parâmetros (LOBO, LIMA e MICHALEWICZ, 2007; MARUO, LOPES e DELGADO, 2005).

As diversas técnicas de Computação Evolucionária se diferenciam basicamente na representação dos indivíduos e na forma como aplicam os operadores genéticos. Como

principais representantes têm-se (ASHLOCK, 2006): Algoritmos Genéticos (HOLLAND, 1975; GOLDBERG, 1989; MITCHELL, 1996) e Programação Genética (KOZA, 1992; BANZHAF, NORDIN, KELLER *et al*, 1998).

3.1.4 Critério de Término

É responsável por interromper o laço de repetição do processo evolutivo que, idealmente, não teria fim. O critério mais comum é limitar o número máximo de gerações ou até que uma solução satisfatória seja encontrada. Outra forma possível é manter o processo enquanto não houver convergência.

Definição 3.4 Convergência Populacional

Considera-se que uma população de indivíduos convergiu se, após algumas gerações, ocorrer simultaneamente: (a) não há melhoria no valor de *fitness* do melhor indivíduo, (b) a média do valor de *fitness* está próxima do melhor indivíduo e (c) o desvio padrão dos valores de *fitness* dos indivíduos é baixa em relação à média de *fitness* (perda de diversidade populacional).

3.2 ALGORITMOS GENÉTICOS

Atualmente, os Algoritmos Genéticos (AG) são a técnica mais conhecida de Computação Evolucionária devido a sua aplicabilidade em vários problemas (ASHLOCK, 2006). Idealizado por John Holland (1975) na Universidade de Michigan, AG apresenta duas características principais: usa uma representação de tamanho fixo[5] e faz uso intensivo do operador de recombinação, sendo a mutação aplicada com probabilidade bem baixa (GOLDBERG, 1989; MITCHELL, 1996). O algoritmo usado no AG está representado através do pseudocódigo da Figura 7.

[5] Há variações de AG que usam indivíduos de tamanho variável.

```
Geração = 0
Pais = CriaPopulacaoInicial()
Avaliar(P)
Enquanto não satisfeito Critério de Término Faça
      Geração = Geração + 1
      Filhos = Seleciona(Pais)
      Cruza(Filhos)
      Muta(Filhos)
      Pais = Filhos
      Avaliar(Pais)
Fim Enquanto
Retorna MelhorIndividuo(Pais)
```

Figura 7: Pseudocódigo para Algoritmo Genético.

3.2.1 Representação dos Indivíduos

Tradicionalmente, a representação mais usada para AG são cadeias de bits denominadas cromossomos (BACK, FOGEL e MICHALEWICZ, 2000). Sendo assim, há a necessidade de se estabelecer um mapeamento entre cada cromossomo (genótipo) e a solução candidata (fenótipo) no domínio do problema. Isto é feito dividindo o cromossomo em pequenos pedaços denominados genes. Um gene pode ser composto por um ou mais bits.

Exemplo 3.1

Considere um problema clássico da Inteligência Artificial (RUSSEL e NORVIK, 2003): Oito rainhas. O objetivo é distribuir oito rainhas em um tabuleiro 8 x 8 de tal forma que nenhuma rainha seja atacada (Figura 8). Neste caso, pode-se usar uma cadeia de 64 bits, onde cada bit identifica se há uma rainha na posição representada. Neste caso, o cromossomo tem tamanho de 64 e o gene, tamanho 1.

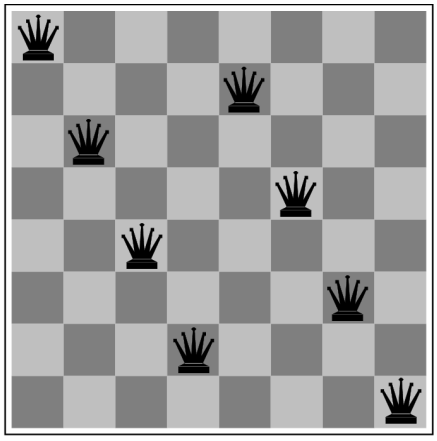

Figura 8: Uma solução possível para o problema das oito rainhas.

3.2.2 Avaliação dos Indivíduos

Os indivíduos em AG são avaliados através de uma função de *fitness* composta por dois passos: (a) extrair os parâmetros do problema que estão codificados no cromossomo e (b) calcular o valor da função objetivo, isto é, calcular o valor resultante, dentro do domínio do problema, usando os parâmetros codificados no cromossomo.

No caso do Exemplo 3.1, o primeiro passo é gerar o tabuleiro a partir do cromossomo. No segundo passo, calcular o número de rainhas que não estão sendo atacadas.

3.2.3 População Inicial

A criação da população inicial é feita de forma aleatória, representando uma amostra inicial do espaço de busca. No caso do AG, geram-se cromossomos diferentes entre si, apenas tendo o cuidado de que haja um mapeamento válido entre o genótipo e o fenótipo. No caso do Exemplo 3.1, cada cromossomo deve ter exatamente oito genes com valor 1 e todos os restantes com o valor 0.

3.2.4 Operadores Genéticos

A forma mais comum de operador genético em AG é o cruzamento em um ponto (MENON, 2004).

Definição 3.5 Cruzamento em um ponto

Considere uma população $\wp$ de cromossomos c de tamanho l e um inteiro positivo λ, denominado ponto de cruzamento, tal que $\lambda < l$. O cruzamento em um ponto é feito em quatro passos, como segue: (a) Escolha dois cromossomos c_1, $c_2 \in \wp$, (b) Construa dois novos cromossomos c_1' e c_2' idênticos a c_1 e c_2, respectivamente, (c) Copie as λ primeiras posições de c_1 para as λ primeiras de c_2' e as λ primeiras de c_2 para as λ primeiras de c_1' e (d) Insira c_1' e c_2' na próxima população.

Exemplo 3.2

Considere cada cromossomo representado por uma cadeia de cinco bits. Um exemplo de cruzamento em um ponto está mostrado na Figura 9, onde o ponto de cruzamento está no terceiro gene, isto é, $\lambda = 3$.

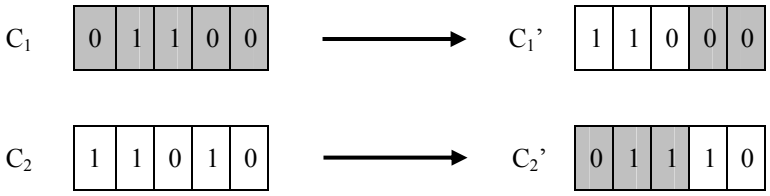

Figura 9: Exemplo de cruzamento em um ponto em AG.

A mutação, por sua vez, é usada com probabilidade baixa (BACK, FOGEL e MICHALEWICZ, 2000).

Definição 3.6 Mutação

Considere uma população $\wp$ de cromossomos c de tamanho l, uma probabilidade p_m, denominada probabilidade de mutação, e um inteiro positivo λ, denominado ponto de mutação, tal que $\lambda < l$. A mutação é feita em três passos, como segue: (a) Escolha um cromossomo $c \in \wp$, (b) Para cada gene $g_i \in c$ onde $1 \leq i \leq l$, altere o seu valor com probabilidade p_m e (c) Insira c na próxima população.

Uma característica interessante dos AG é o fato de todos os indivíduos selecionados serem probabilisticamente afetados pelos operadores de cruzamento e mutação, isto é, pode acontecer de um mesmo indivíduo sofrer cruzamento e mutação, enquanto que outro indivíduo, em outro extremo, não sofrer nenhuma alteração, apesar de selecionado.

3.3 PROGRAMAÇÃO GENÉTICA

A Programação Genética (PG) é uma abordagem para a geração automática de programas de computador desenvolvida por John Koza (1992). A técnica se baseia na combinação de idéias da computação evolucionária e teoria de compiladores (representação de programas como árvores sintáticas). Basicamente, a PG é um algoritmo que busca, dentre um espaço infinito de programas de computador, uma solução ou, pelo menos, uma boa aproximação para resolver determinado problema. Atualmente tem sido aplicada a vários problemas com sucesso (O'REILLY, YU, RIOLO *et al*, 2005; KOZA, KEANE, STREETER *et al*, 2003).

O algoritmo é semelhante ao AG e seu pseudocódigo está na Figura 10.

```
Geração = 0
P = CriaPopulacaoInicial()
Avaliar(P)
Enquanto não satisfeito Critério de Término Faça
    Geração = Geração + 1
    P' = SelecionaPais(P)
    Para cada indivíduo I de P'
        Escolhe probabilisticamente o operador a aplicar
        Se escolher cruzamento
            Então Escolhe outro indivíduo J de P'
                Recombina(I, J)
        Fim Se
        Se escolher mutação
            Então Muta(I)
        Fim Se
    Fim Para
    P = P'
    Avaliar(P)
Fim Enquanto
```

Figura 10: Pseudocódigo para Programação Genética

É possível perceber que em PG o operador genético a aplicar é sempre escolhido probabilisticamente. No caso dos AG, os operadores são sempre escolhidos, sendo a sua aplicação probabilística. Desta forma, pode ocorrer que um indivíduo em AG sofra a ação de ambos os operadores, o mesmo não acontecendo em PG.

3.3.1 Representação dos Indivíduos

A representação dos programas em PG tradicionalmente se baseia diretamente em uma árvore sintática[6], isto é, os programas são formados pela livre combinação de funções e terminais adequados ao domínio do problema (BANZHAF, NORDIN, KELLER *et al*, 1998; KOZA, 1992).

Parte-se de dois conjuntos: F como sendo o conjunto de funções e T como o conjunto de terminais. O conjunto F pode conter operadores aritméticos (`+`, `-`, `*` etc.), funções matemáticas (`sen` , `log` etc.), operadores lógicos (`E`, `OU` etc.) dentre outros. Cada $f \in$ F tem associada uma aridade (número de argumentos) superior a zero. O conjunto T é composto pelas variáveis, constantes e funções de aridade zero (sem argumentos).

Exemplo 3.3

Considerando o conjunto dos operadores aritméticos como sendo o conjunto de funções e a variável *x* e o número dois como terminais, isto é: F = { `+` , `-` , `*` , `/` } e T = { `x`, `2` }, então expressões matemáticas simples tais como `x*x+2` podem ser produzidas. A representação é feita por uma árvore como mostrado na Figura 11.

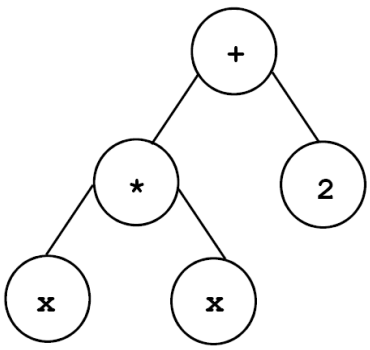

Figura 11: Árvore representando `x*x+2`

O espaço de busca é determinado por todas as árvores que possam ser criadas pela livre combinação de elementos dos conjuntos F e T.

[6] Basicamente, a PG pode ser aplicada em qualquer estrutura, não apenas a árvores sintáticas (KOZA, KEANE, STREETER *et al*, 2005).

3.3.2 Avaliação dos Indivíduos

Usualmente em PG, para se proceder à avaliação de *fitness*, é fornecido um conjunto de casos de treinamento, denominados *fitness cases*, contendo valores de entrada e saída a serem aprendidos. A cada programa são fornecidos valores de entrada e confronta-se a sua resposta ao valor esperado de saída. Quanto mais próxima a resposta do programa estiver do valor de saída, melhor é o programa.

Desta forma, a avaliação de *fitness* estabelece uma forma de se diferenciar os melhores dos piores, servindo como a força mestre do processo evolutivo, sendo a medida (usada durante a evolução) do quanto o programa aprendeu a predizer as saídas correspondentes às entradas dentro de um domínio de aprendizagem.

3.3.3 População Inicial

Tradicionalmente, a população inicial é composta por árvores geradas aleatoriamente a partir dos conjuntos de funções F e de terminais T. Inicialmente escolhe-se aleatoriamente uma função $f \in$ F. Para cada um dos argumentos de *f*, escolhe-se um elemento de $\{F \cup T\}$. O processo continua até que ocorram apenas terminais como nós-folha da árvore. Para se evitar árvores muito grandes, uma profundidade máxima é estabelecida.

Apesar de simples, esta forma de criação da população inicial é incapaz de gerar uma boa amostragem do espaço de busca na maioria dos problemas (LUKE, 2000).

Para melhorar a qualidade dos programas gerados na população inicial, há diversos outros métodos, sendo os mais comuns (LUKE, 2000): *ramped-half-and-half* (KOZA, 1992), *random-branch* (CHELLAPILLA, 1997), *uniform* (BOHM e GEYER-SCHULZ, 1996) e *probabilistic tree-creation* (LUKE, 2000).

3.3.4 Operadores Genéticos

O operador de cruzamento opera de forma semelhante ao cruzamento em um ponto do AG. Um ponto aleatório de cruzamento é escolhido em cada programa-pai e as subárvores, a partir destes pontos, são trocadas. Um exemplo de cruzamento é mostrado na Figura 12. Neste exemplo, foram escolhidos os programas: `((2*(x+x))+1)` e `(((x+1)*x)-2)`. Aleatoriamente um nó em cada árvore foi selecionado, identificado com um traçado mais denso na figura. As subárvores são então trocadas, gerando os dois novos programas: `((x+1)+1)` e `(2*((x+x)*x)-2)`.

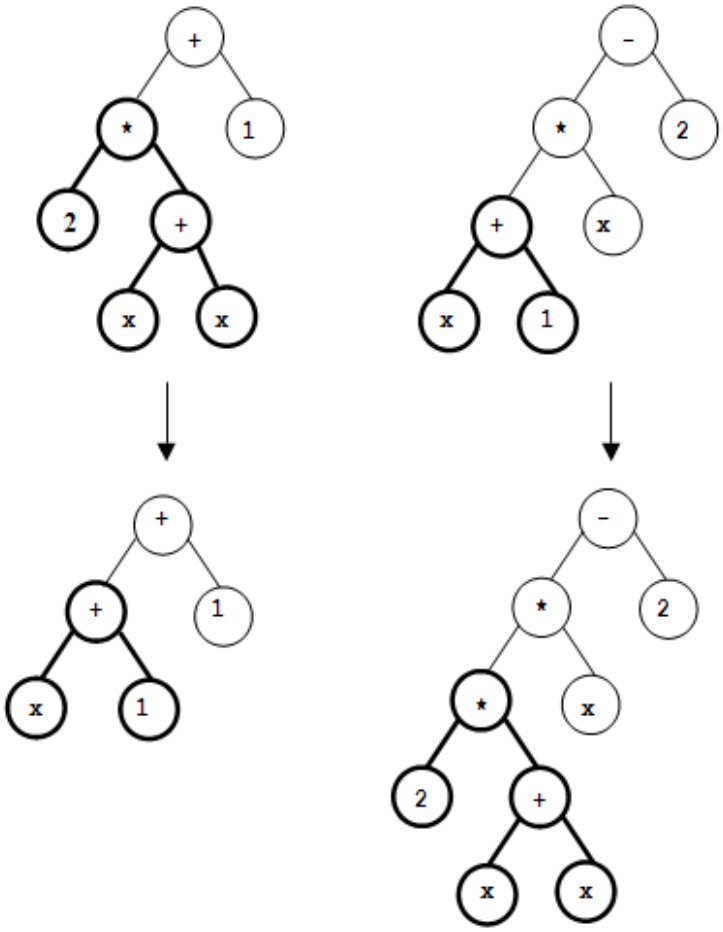

Figura 12: Atuação do operador de cruzamento em PG

Para que o cruzamento seja sempre possível, o conjunto de funções deve apresentar a propriedade de fechamento (*closure*), isto é, as funções devem aceitar como argumento qualquer terminal ou o resultado de qualquer função. Se não for possível, devem-se estabelecer critérios de restrição na escolha dos pontos de cruzamento. As abordagens mais conhecidas são (POLI, LANGDON, MCPHEE *et al*, 2007): Programação Genética Fortemente Tipada (MONTANA, 1995; SPECTOR, 2004) e a Programação Genética Orientada a Gramáticas (WHIGHAM, 1995; RATLE e SEBAG, 2001). Na primeira abordagem, cada função e terminal têm associado um tipo de dado. Ao se proceder a aplicação do cruzamento, só é permitida entre nós de mesmo tipo de dado. Na segunda abordagem, uma gramática livre de contexto é usada na representação dos indivíduos e o cruzamento atua trocando símbolos não-terminais de mesma natureza.

3.4 COMPUTAÇÃO EVOLUCIONÁRIA E INFERÊNCIA GRAMATICAL

Há vários trabalhos envolvendo Computação Evolucionária para inferência gramatical. O primeiro deles (ZHOU e GREFENSTETTE, 1986) aplicou Algoritmos Genéticos (AG) para inferência de autômatos finitos determinísticos (AFD) para quatro linguagens diferentes com alfabeto {0, 1} usando tanto exemplos positivos quanto negativos. A representação era de tamanho fixo, limitando em oito o número de estados possíveis do autômato. Infelizmente, faltam detalhes da implementação no artigo, tais como é feita exatamente a avaliação de *fitness* e os parâmetros do AG usado.

Em seguida, Wyard aplicou AG para inferência de regras de produção em uma gramática livre de contexto (WYARD, 1993). A representação de cada indivíduo era feita por uma lista de regras de produção na Forma Normal de Greibach[7]. Além disto, o operador de cruzamento era limitado a não quebrar a lista de não-terminais. A avaliação de *fitness* era feita aplicando a gramática a cada exemplo positivo ou negativo. Se a gramática cobrisse um exemplo positivo, ganhava um ponto e caso cobrisse um exemplo negativo, perdia um ponto. Após a avaliação de todos os exemplos, cada gramática recebia a pontuação obtida. Infelizmente, o método teve resultados positivos apenas na inferência de dois tipos de gramáticas: parênteses aninhados e seqüências $(ab)^+$. Avanços na aplicação do método foram propostas em 1994, onde foi possível inclusive inferir gramática para palíndromos de, no máximo, três terminais (WYARD, 1994).

Dupont aplicou AG para inferir AFD para um conjunto de quinze linguagens regulares e comparou com os resultados obtidos na época por outros métodos, obtendo resultados melhores (DUPONT, 1994).

Lankhorst propôs em 1994 uma forma de representação para AG mais eficiente baseando-se em cadeias binárias e no mapeamento gene $\rightarrow$ regra de produção. Conseguiu inferir com sucesso nove tipos de gramáticas (LANKHORST, 1994). Contudo, a quantidade máxima de regras de produção da gramática é limitada pelo tamanho do cromossomo e o mapeamento não é trivial (KAMMEYER e BELEW, 1996).

Para lidar com o problema específico de indução de gramáticas para linguagem natural usando AG, surgem os trabalhos de Smith e Witten (1995) usando conectivos `AND` e `OR` em um formato de gramática semelhante a expressões da linguagem LISP (WINSTON e HORN, 1989). Os resultados são limitados a gramáticas simples do inglês.

---

[7] A Forma Normal de Greibach tem produções $\alpha \rightarrow \beta\delta$ ou $\alpha \rightarrow \beta$ onde $\alpha, \delta \in N$ e $\beta \in \Sigma^*$

A inferência de gramáticas estocásticas com AG aparece pela primeira vez no trabalho de Schwehm e Ost (1995) com o uso de AG em máquinas paralelas, contudo se restringindo às gramáticas regulares. Kammeyer e Belew (1996), por sua vez, usam gramáticas livres de contexto estocásticas e aplicam busca local como refinamento no cálculo das probabilidades das produções. Melhorias no método foram propostas por Keller e Lutz (2005), tais como o uso de *corpora*[8], uso de comprimento de descrição mínimo (*Minimum Description Length,* MDL) (GRUNALD, 2007; VAPNIK, 1998) e uma variante do operador de cruzamento denominada RAOC (*Randomized And/Or Crossover*). Esta variante funciona da seguinte forma: como a representação usada é a binária, para cada gene gerado em um dos filhos é usada a operação AND entre os genes dos pais com probabilidade α ou a operação OR, com probabilidade 1-α. No segundo filho gerado, é feito o inverso, isto é, se é usada a operação AND no primeiro, no segundo é usada a operação OR (KELLER e LUTZ, 2005).

A Programação Genética (PG) aplicada à inferência gramatical surgiu nos trabalhos de Dunay (1994), usando como representação o autômato de pilha determinístico. A inferência direta de regras de produção apareceu nos trabalhos de Korkmaz e Ucoluk (2001) com o uso de conhecimento prévio. O uso de PG para inferir gramáticas regulares surge nos trabalhos de Hu (1998) para reconhecimento de *motifs* em proteínas em seqüências obtidas do PROSITE (SIGRIST, CERUTTI, HULO *et al*, 2002; HULO, BAIROCH, BULLIARD *et al*, 2006). Ross (2002) usa PG para inferir gramáticas regulares lógicas e aplica também no reconhecimento de *motifs* em proteínas, porém com limitado sucesso por reconhecer apenas um grupo pequeno de famílias. Neste caso, alternativas na representação das gramáticas produzem melhor resultado, tais como uso de vetores ao invés de árvores, porém limitam sua generalidade (SEEHUUS, TVEIT e EDSBERG, 2005).

Em geral, a computação evolucionária tem sido uma alternativa para inferência de gramáticas de diversos tipos, sendo que a diferenciação reside na representação das gramáticas e na criação ou modificação dos operadores genéticos existentes (REYNOSOL e MONTANO, 2005; JOHNSON e FARRELL, 2004; MERNIK, GERLIC, ZUMER *et al*, 2003).

[8] *Corpora* é plural de *corpus* que é uma grande coleção de texto.

CAPÍTULO 4

MODELO PROPOSTO DE INFERÊNCIA GRAMATICAL

A inferência gramatical busca encontrar uma gramática no conjunto de todas as gramáticas possíveis capaz de reconhecer os exemplos positivos e rejeitar os negativos. Além disso, a gramática deve ser capaz de reconhecer outros exemplos positivos não apresentados durante o treinamento. Isto é, a gramática não deve se restringir a reconhecer apenas os exemplos positivos apresentados durante a inferência (HIGUERA, 2005).

Neste trabalho apresenta-se um modelo de inferência gramatical livre de contexto usando Computação Evolucionária (CE) que se baseia em uma proposta de uma variante de Programação Genética (PG) com otimizadores locais (RODRIGUES e LOPES, 2006). A escolha de PG se deve ao fato das regras de produção serem facilmente representáveis como árvores e não como cadeias binárias, caso fosse usado Algoritmos Genéticos (AG).

Em seguida, o modelo proposto é estendido de forma a suportar gramáticas *fuzzy*, mais adequadas para inferência quando não se sabe ou não exista uma gramática livre de contexto capaz de reconhecer os exemplos positivos e rejeitar os negativos.

4.1 REPRESENTAÇÃO DOS INDIVÍDUOS

Como PG permite evoluir qualquer tipo de estrutura, uma representação adequada deve ser feita a fim de permitir a aplicação dos operadores genéticos sem tornar o indivíduo inválido. Além disto, a representação deve favorecer, quando possível, a aplicação da função de *fitness* (ASHLOCK, 2006; O'REILLY, YU, RIOLO *et al*, 2005).

No caso de gramáticas livres de contexto (GLC), as regras de produção podem ser representadas em forma de árvore com o lado esquerdo de cada produção representando uma função cujos argumentos são os símbolos que ela produz. Neste caso, a raiz da árvore é o símbolo inicial. Apesar de simples, esta representação apresenta dois problemas. O primeiro problema se refere ao fato de normalmente existir mais de uma regra de produção com o mesmo lado esquerdo, tendo como conseqüência mais de uma função com o mesmo nome. O segundo se refere ao fato de que as regras de produção podem ser dependentes uma das outras. Ao se aplicar um operador genético, o indivíduo pode facilmente se tornar inválido. Por exemplo, no caso do cruzamento, uma boa parte pode ser substituída, tornando o indivíduo facilmente inválido por perder parte das produções necessárias.

Uma alternativa apresentada por Mernik, Gerlik, Zumer e Bryant (2003) trata as gramáticas como uma lista de produções, onde cada produção é representada por uma árvore. Desta forma, as produções estão ligadas uma as outras de forma não-hierárquica.

No modelo proposto, cada gramática é composta por uma lista de terminais, uma lista de não-terminais e uma lista ligada de árvores representando cada regra de produção. Para facilitar a aplicação da função *fitness*, adotou-se a Forma Normal de Chomsky[9] (FNC) para permitir a aplicação do algoritmo CYK[10]. Em cada árvore, o seu nó raiz contém o não-terminal do lado esquerdo da regra de produção e os seus nós filhos contém um símbolo terminal ou dois símbolos não-terminais (RODRIGUES e LOPES, 2006). A Figura 13 exibe a gramática G = (∑, N, P, S), sendo ∑ = {a, b}, N = {S, A} e P = {S → AS; S → b; A → SA; A → a}.

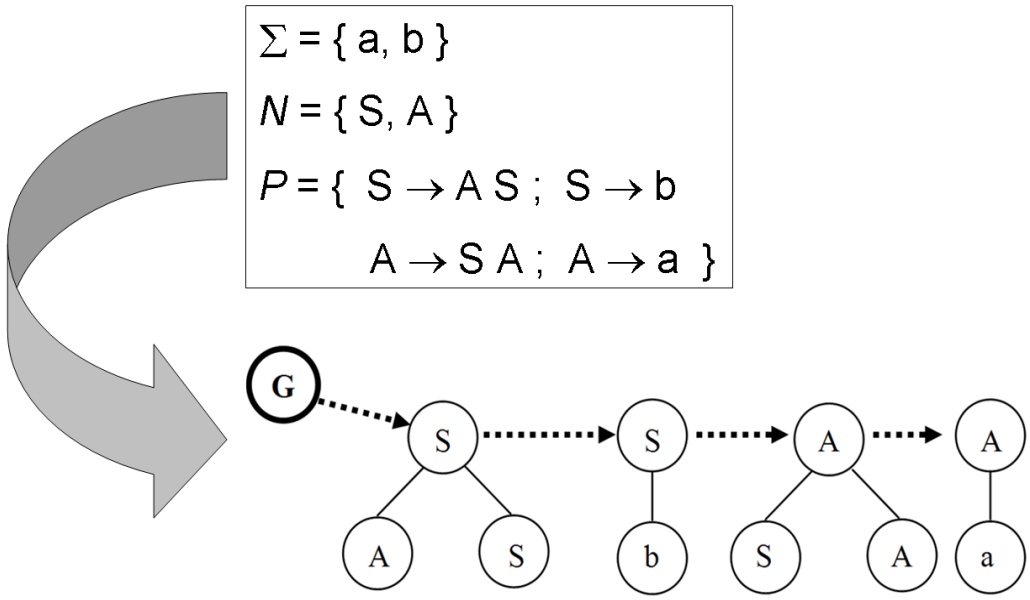

Figura 13: Exemplo de representação de uma gramática.

Como vantagem desta representação, pode-se citar que operadores genéticos podem atuar isoladamente em cada produção, substituindo-a quando possível, como será mostrado na Seção 4.4. Neste caso, o efeito destrutivo dos operadores genéticos pode ser evitado.

[9] Na FNC, as produções são da forma $\alpha \rightarrow \beta\delta$ ou $\alpha \rightarrow \sigma$, onde $\alpha, \beta, \delta \in N$ e $\sigma \in \Sigma^+$

[10] O algoritmo CYK está descrito na Seção 2.4 (Reconhecimento Gramatical)

Além disto, após a aplicação dos operadores genéticos, as gramáticas geradas são verificadas quanto a sua consistência e, caso sejam inválidas, a operação não é efetuada. Desta forma, a busca ocorre apenas entre gramáticas consistentes.

4.2 AVALIAÇÃO DOS INDIVÍDUOS

Como o sistema proposto é baseado em Programação Genética (PG), existe uma população composta por gramáticas que cooperam entre si na descoberta da melhor gramática. Isto é feito através de sucessivas trocas ou alterações de regras de produção entre as gramáticas que melhor se aproximem da gramática desejada. Para avaliar a qualidade das gramáticas, há a necessidade de se estabelecer um método de avaliação que permita reconhecer quais são as mais promissoras. Portanto, deve ser estabelecida uma métrica adequada de *fitness* para avaliar cada gramática. Além disto, deve-se estabelecer um mecanismo capaz de reconhecer quais regras de produção devem ser afetadas pelos operadores genéticos.

Tradicionalmente em inferência gramatical usando computação evolucionária, cada gramática é submetida a um conjunto de exemplos positivos e negativos para verificar se reconhece ou não cada um deles, isto é, se é possível gerar os exemplos através da gramática. Em seguida, calcula-se a sua taxa de acerto (LANKHORST, 1994; DUNAY, 1994; KORKMAZ e UCOLUK, 2001).

Definição 4.1 Taxa de Acerto

Dado um conjunto finito de exemplos, a taxa de acerto é o percentual de exemplos corretamente classificados.

Apesar de coerente, a taxa de acerto é incapaz de identificar, dentre duas gramáticas diferentes, qual a mais promissora se elas tiverem a mesma taxa de acerto. Por exemplo, suponha que os conjuntos de exemplos tenham o mesmo tamanho, ou seja, a mesma quantidade de sentenças. Uma gramática que rejeite todos os exemplos positivos e negativos terá uma taxa de acerto de 50%, enquanto outra que cubra metade dos exemplos positivos e rejeite metade dos exemplos negativos, terá a mesma taxa de acerto (50%). Porém, esta segunda gramática é muito mais promissora, pois contém algumas produções úteis. Portanto, métricas que avaliem o "comportamento" em relação à cobertura ou não dos exemplos devem ser levadas em consideração.

Outra forma possível é estabelecer uma métrica que avalie o quanto a gramática cobre de cada exemplo apresentado. Neste caso, o valor está no intervalo [0, 1] onde 1 (um) significa que a gramática cobriu a sentença em sua totalidade e valores menores representam a porcentagem coberta. Por exemplo, se o valor for 0,5 significa que a gramática reconheceu a primeira metade do exemplo apresentado. O valor final da gramática é obtido pela soma das avaliações de cada exemplo positivo apresentado. Um fator de penalidade é usado para cada exemplo negativo coberto (MERNIK, GERLIC, ZUMER *et al*, 2003; KORKMAZ e UCOLUK, 2001). O grande problema com esta abordagem reside em determinar como deve ser o cálculo do fator de penalidade. Além disto, a sua generalização não é possível, isto é, o fator de penalidade acaba sendo ajustado para cada problema de inferência.

Para conseguir métricas que caracterizem melhor uma gramática e que independam da inferência em particular, propõem-se o uso de uma matriz de confusão (WITTEN e FRANK, 2005; MONARD e BARANAUSKAS, 2003).

A matriz de confusão é construída com base em quatro valores (Tabela 3):

- **VP** é a quantidade de exemplos positivos reconhecidos (verdadeiro positivo);
- **VN** é quantidade de exemplos negativos não reconhecidos (verdadeiro negativo);
- **FP** é a quantidade de exemplos negativos reconhecidos (falso positivo);
- **FN** é a quantidade de exemplos positivos não reconhecidos (falso positivo);

Tabela 3: Matriz de confusão para classificação com duas classes

Classe	**Predita Positiva**	**Predita Negativa**
Verdadeira Positiva	Verdadeiros Positivos (VP)	Falsos Negativos (FN)
Verdadeira Negativa	Falsos Positivos (FP)	Verdadeiros Negativos (VN)

Desta matriz, várias métricas podem ser extraídas tais como: confiabilidade positiva *prel* (*positive reliability*), confiabilidade negativa *nrel* (*negative reliability*), suporte *sup*, taxa de acerto *tacc* (*total accuracy*), sensibilidade *sens* (*sensitivity rate*), especificidade *spec* (*specificity rate*) e cobertura *cov* (*coverage*) calculadas a partir das Equações 9 a 15, respectivamente (WEISS e KULIKOWSKI, 1991).

$$prel = \frac{VP}{VP + FP} \quad (9)$$

$$nrel = \frac{VN}{VN + FN} \quad (10)$$

$$\text{sup} = \frac{VP}{VP + VN + FP + FN} \quad (11)$$

$$tacc = \frac{VP + VN}{VP + VN + FP + FN} \quad (12)$$

$$sens = \frac{VP}{VP + FN} \quad (13)$$

$$spec = \frac{VN}{VN + FP} \quad (14)$$

$$\text{cov} = \frac{VP + FP}{VP + VN + FP + FN} \quad (15)$$

É possível perceber que quando a gramática aceita todos os exemplos positivos e rejeita todos os negativos, os valores de *prel*, *nrel*, *tacc*, *sens* e *spec* são iguais a um e *sup* e *cov* são iguais à proporção do número de exemplos positivos apresentados.

Deve-se, então, estabelecer uma combinação de métricas que consiga favorecer a diferenciação entre as gramáticas, levando à escolha da mais promissora. Uma forma de cálculo comum é o produto da sensibilidade pela especificidade (Equação 16), permitindo um compromisso entre a cobertura dos exemplos positivos (*sens*) e a rejeição dos negativos (*spec*). Este produto foi proposto inicialmente por Lopes, Coutinho e Lima (1998) e tem sido usado em vários problemas de classificação (WEINERT e LOPES, 2006; BOJARCZUK, LOPES e FREITAS, 2004).

$$fitness = sens.spec \quad (16)$$

Porém, no caso de inferência gramatical, pode ocorrer que uma gramática cubra todos os exemplos positivos e negativos. Neste caso, esta gramática receberá zero como valor de *fitness* devido ao valor da especificidade ser zero. Isto a igualaria a outra que não cobre nenhum exemplo positivo (sensibilidade zero). Durante o processo evolutivo, facilmente uma gramática de sensibilidade um e especificidade zero seria perdida, apesar de conter produções que poderiam ser úteis, pois cobrem todos os exemplos, tanto positivos quanto negativos. Para evitar esta perda, a avaliação de *fitness* adotada foi a média entre a sensibilidade e a especificidade (Equação 17).

$$fitness = \frac{(sens + spec)}{2} \qquad (17)$$

Portanto, o valor de *fitness* estará no intervalo [0, 1] sendo que quanto maior, melhor o indivíduo. Isto é diferente da abordagem clássica em PG, onde habitualmente a avaliação de *fitness* é feita com base na taxa de erro obtida na avaliação dos *fitness cases* (Seção 2.4) onde quanto menor o valor, melhor.

4.3 POPULAÇÃO INICIAL

Tradicionalmente em PG a população inicial é criada de forma aleatória, representando uma amostragem inicial do espaço de busca. Através da avaliação dos indivíduos pela função de *fitness*, torna-se possível escolher os melhores indivíduos, alterá-los pelo uso dos operadores genéticos e convergir para uma possível solução (O'REILLY, YU, RIOLO *et al*, 2005).

Para o caso de gramáticas, se a população inicial for criada de forma totalmente aleatória, há o risco delas serem incapazes de reconhecer qualquer exemplo apresentado e, portanto, não conter elementos que possam auxiliar a convergência. Isto obrigaria o uso de um grande número de indivíduos na população inicial para prover uma boa diversidade, aumentando desnecessariamente o tempo de execução. Assim, há a necessidade de que uma parte das produções inicialmente usadas seja útil para reconhecer, pelo menos, uma parte dos exemplos positivos.

Para poder descobrir que produções devem ser criadas para reconhecer os exemplos positivos, uma das alternativas é usar o método Sequitur[11] (NEVILL-MANNING e WITTEN, 1997) para cada exemplo positivo e, em seguida, unir os conjuntos de produções obtidos. O problema reside no fato de que a simples união destes conjuntos de produções, onde cada um gera apenas uma sentença, não generaliza o suficiente.

Outra alternativa é construir iterativamente o conjunto de regras de produção usando os exemplos positivos, tal como é feito no método Synapse[11] (NAKAMURA e MATSUMOTO, 2002). Porém, esta alternativa acaba exigindo uma quantidade muito grande de exemplos positivos.

Neste trabalho, optou-se por inicialmente gerar um conjunto φ de regras de produção na Forma Normal de Chomsky (FNC) capaz de reconhecer Σ^+. Em seguida, com base nos exemplos positivos e negativos, as regras de produção não usadas para reconhecer exemplos positivos, mas usadas para exemplos negativos, são retiradas do conjunto φ.

4.3.1 Geração do Conjunto Inicial de Regras de Produção

Como ponto de partida, considere um conjunto de *n* símbolos terminais $\{a_1, a_2,..., a_n\}$, representado como Σ_n. Deseja-se produzir um conjunto de regras de produção capaz de gerar sentenças de Σ^+ maiores que um a partir de Σ_n.

Definição 4.2 Conjunto de Regras de Produção para Σ^+

Considerando o conjunto de símbolos terminais Σ_n, tem-se que o conjunto de regras de produção $P = P_1 \cup P_2$, onde P_1 e P_2 são definidos como segue:

i. Regras de produção que geram sentenças de tamanho dois.

$$P_1 = \{P_i \rightarrow \alpha \mid i \in [1, n], \alpha \in \Sigma_n\} \cup \{S \rightarrow P_i\ P_j \mid i, j \in [1, n]\}$$

ii. Regras de produção que geram sentenças maiores que dois.

$$P_2 = \{P_k \rightarrow P_i P_j \mid i, j \in [1, n], k \in [1, n+1, n^2]\} \cup \{S \rightarrow S\ P_k \mid k \in [n+1, n^2]\}$$

O conjunto de regras de produção assim formado é capaz de gerar Σ^+ para sentenças de tamanho maior que um. É importante frisar que este conjunto não é mínimo, pois o objetivo é gerar uma boa diversidade de regras de produção.

[11] Os métodos de inferência gramatical Sequitur e Synapse estão descritos na Seção 2.5.1

Exemplo 4.1

Considere o caso em que $n = 3$. Para facilitar a análise, ao invés de usar diretamente os subconjuntos de $\Sigma_3 = \{a_1, a_2, a_3\}$, serão usados os subconjuntos de $\{1, 2, 3\}$ como índices dos símbolos não-terminais. Sendo assim, tem-se que o conjunto de símbolos não-terminais $N = \{P_1, P_2, P_3, P_4, P_5, P_6\ P_7, P_8, 9_3\ P_{10}, P_{11}, P_{12}, S\}$ e o conjunto $P = \{P_1 \rightarrow a_1, P_2 \rightarrow a_2, P_3 \rightarrow a_3; S \rightarrow P_1\ P_1; S \rightarrow P_1\ P_2; S \rightarrow P_1\ P_3; S \rightarrow P_2\ P_1; S \rightarrow P_2\ P_2; S \rightarrow P_2\ P_3; S \rightarrow P_3\ P_1; S \rightarrow P_3\ P_2; S \rightarrow P_3\ P_3; P_4 \rightarrow P_1\ P_1; P_5 \rightarrow P_1\ P_2; P_6 \rightarrow P_1\ P_3; P_7 \rightarrow P_2\ P_1; P_8 \rightarrow P_2\ P_2; P_9 \rightarrow P_2\ P_3; P_{10} \rightarrow P_3\ P_1; P_{11} \rightarrow P_3\ P_2; P_{12} \rightarrow P_3\ P_3; S \rightarrow S\ P_1; S \rightarrow S\ P_2; S \rightarrow S\ P_3; S \rightarrow S\ P_4; S \rightarrow S\ P_5; S \rightarrow S\ P_6; S \rightarrow S\ P_7; S \rightarrow S\ P_8; S \rightarrow S\ P_9; S \rightarrow S\ P_{10}; S \rightarrow S\ P_{11}; S \rightarrow S\ P_{12}\}$ gera Σ^+ para sentenças maiores que um.

Pelo método proposto, a gramática gerada tem o total de regras de produção igual a $3n^2+2n$. O pseudocódigo para este método está na Figura 15.

Este conjunto de regras de produção P reconhece Σ^+ e, portanto, reconhece tanto os exemplos positivos (E^+) quanto os negativos (E^-). O próximo passo é remover as produções que não são usadas para reconhecer os exemplos positivos. Na Figura 14 apresenta-se o pseudocódigo do método proposto.

```
Para cada w ∈ E+ faça
    Marque cada produção p ∈ P que participar da cobertura
Fim Para
Para cada p ∈ P faça
    Se p não foi marcada
        Então P = P - {p}
    Fim Se
Fim Para
```

Figura 14: Pseudocódigo de refinamento do conjunto inicial de produções.

Sendo assim, o conjunto inicial de regras de produção é composto pelas regras que reconhecem todos os exemplos positivos e, possivelmente, uma boa parte dos exemplos negativos.

4.3.2 Geração da População Inicial

Uma vez construído o conjunto inicial de regras de produção, metade[12] da população inicial é construída com base nestas produções, conforme o pseudocódigo da Figura 15, sendo P_G o conjunto de produções da gramática gerada e n_p, a quantidade de produções a serem adicionadas. A outra metade da população inicial é construída de forma aleatória.

```
P_G = ϕ
Para cada gramática a ser criada faça
    Para cada terminal a_i ∈ Σ faça
        P_G = P_G ∪ { P_i → a_i }
    Fim Para
    Para i = 1 até n_p
        Escolha aleatoriamente p ∈ P
        P_G = P_G ∪ {p}
    Fim Para
Fim Para
```

Figura 15: Pseudocódigo de criação do conjunto de produções de cada gramática.

A quantidade n_p de produções de cada gramática é controlada entre um tamanho mínimo e máximo determinado inicialmente, sendo que a distribuição dos tamanhos é feita de forma uniforme. Por exemplo, no caso de dez gramáticas com tamanhos variando de dez a quatorze, então duas gramáticas terão tamanho dez, duas terão tamanho onze e assim por diante.

4.3.3 Validação da População Inicial

Para evitar criação de gramáticas inválidas, isto é, contendo produções inúteis, aplica-se um operador de correção após a sua criação[13] que funciona em duas etapas. A primeira etapa procura garantir que qualquer não-terminal gere um terminal. A segunda, garantir que qualquer não-terminal seja alcançável pelo símbolo inicial. Se após estas duas

[12] Poderia ser qualquer percentual da população inicial, mas testes preliminares indicam que a proporção de 50% é a mais indicada.

[13] Este operador também é aplicado após cada operador genético.

etapas não restar nenhuma produção na gramática, então a gramática não é adicionada à população inicial. As etapas estão descritas a seguir.

i. Garantir que qualquer símbolo não-terminal gere símbolo terminal

   Para tal, constrói-se um conjunto $V \subseteq N$ de símbolos não-terminais da forma indicada na Figura 16, considerando que Σ é o conjunto de terminais e N, o conjunto de não-terminais da gramática.

```
Para cada produção { α → β } ∈ P onde β ∈ Σ faça
    V = V ∪ { α }
Fim Para
Para cada produção { α → β δ } ∈ P onde β, δ ∈ N faça
    V = V ∪ { α }
Fim Para
Para cada produção { α → β δ } ∈ P onde β, δ ∈ N faça
    Se α ∉ V ou β ∉ V
        Então retire esta produção de P
    Fim Se
Fim Para
```

Figura 16: Pseudocódigo para garantir que qualquer não-terminal gere terminal.

Ao final desta etapa, todas as produções que tenham, em seu corpo, não-terminais inexistentes na gramática são eliminadas.

ii. Garantir que qualquer símbolo não-terminal seja alcançável pelo símbolo inicial

   Nesta etapa são construídos dois conjuntos: um de não-terminais ou variáveis $V \subseteq N$ e outro de terminais $T \subseteq \Sigma$. A Figura 17 mostra o pseudocódigo, considerando que Σ é o conjunto de terminais, N é o conjunto de não-terminais da gramática e P, o conjunto de produções.

```
V = { S }
T = ∅
Repita
    adicionou = 0
    Para cada produção p ∈ P faça
        Se p é da forma { α → β } onde α ∈ V e β ∈ Σ
            Então
                T = T ∪ { β }
                adicionou = 1
        Fim Se
        Se p é da forma { α → β δ } onde α ∈ V e β, δ ∈ N
            Então
                V = V ∪ { β, δ }
                adicionou = 1
        Fim Se
  Fim Para
Até que adicionou = 0
Para cada produção { α → β } ∈ P onde β ∈ Σ faça
    Se β ∉ T
        Então
            Retire esta produção de P
    Fim Se
Fim Para
Para cada produção { α → β δ } ∈ P onde β, δ ∈ N faça
    Se β ∉ V ou δ ∉ V
        Então
            Retire esta produção de P
    Fim Se
Fim Para
```

Figura 17: Pseudocódigo para garantir que qualquer não-terminal seja alcançável.

Ao término desta etapa, a gramática conterá apenas produções úteis. Se não houver nenhuma produção, então a gramática estava anteriormente inconsistente e não é incluída na população inicial.

4.4 MÉTODO DE SELEÇÃO

Neste trabalho, adotou-se o torneio estocástico como método de seleção. Porém, uma modificação é proposta. Caso haja empate no torneio, o critério de desempate é pelo valor da especificidade, isto é, escolhe-se o indivíduo de maior valor de especificidade. Caso o empate ainda persista, escolhe-se aquela gramática com maior quantidade de regras de produção. Caso ainda persista, a escolha passa a ser aleatória.

O critério de desempate baseado no maior valor de especificidade se deve ao fato de se preferir a gramática que mais rejeite exemplos negativos. Isto se deve ao fato do sistema de PG usado usar um operador de aprendizagem incremental que favorece o valor de sensibilidade, como será explicado na Seção 4.5.3. Da mesma forma, o critério seguinte da escolha da maior gramática (maior número de regras de produção) se deve ao fato de preferir aquela que tenha maior material "genético".

4.5 OPERADORES GENÉTICOS

O objetivo principal dos operadores genéticos é permitir a troca ou alteração nos indivíduos selecionados de forma a tentar melhorá-los. Os operadores principais em PG são reprodução e cruzamento, sendo o operador de mutação raramente é usado (KOZA, KEANE, STREETER *et al*, 2003). O operador de reprodução não causa nenhuma alteração na gramática, enquanto que o cruzamento e a mutação causam, sendo operadores de busca global.

Além destes operadores, o modelo proposto apresenta dois novos operadores: um para busca local (Aprendizagem Incremental) e outro para prover maior diversidade (Expansão) (RODRIGUES e LOPES, 2006; 2007).

Por se adotar uma representação de indivíduos como uma lista de produções, os operadores genéticos de cruzamento e mutação podem facilmente atuar isoladamente nas produções, modificando-as se desejado. Sendo assim, pode-se escolher aleatoriamente a produção que se deseja cruzar ou sofrer mutação e então proceder a aplicação do operador.

Porém, a convergência pode ser comprometida, pois em determinados momentos, uma produção útil pode ser inadvertidamente alterada, prejudicando a gramática. Resultados preliminares demonstraram tal efeito destrutivo dos operadores em várias execuções. Assim, há a necessidade de se estabelecer um critério que favoreça as melhores trocas ou, pelo menos, tente não prejudique as gramáticas.

A solução adotada foi usar um escore baseado em *Credit Assignment* (BIANCHI, 1996) para identificar as melhores e piores produções em cada gramática. O valor do escore tem por objetivo permitir a diferenciação entre produções úteis e aquelas de menor importância.

Durante a avaliação das gramáticas, calcula-se o número de vezes que cada produção participou da cobertura de exemplos positivos, representado por p^+, e o número de vezes que participou da cobertura de exemplos negativos, representado por p^-. Com base nestes dois valores, calcula-se um valor de "crédito", denominado $c(p)$, que representa o grau de utilidade de cada produção dentro da gramática. Este valor é calculado conforme a Equação 18.

$$c(p) = \frac{p^+ - p^-}{p^+ + p^-} \qquad (18)$$

Desta forma, cada uma das produções recebe um valor no intervalo [-1, 1], sendo que, quanto maior, melhor a participação da produção na cobertura dos exemplos. Se o valor for negativo, significa que a produção cobriu mais exemplos negativos do que positivos.

Após a atuação dos operadores genéticos, aplicam-se as etapas descritas na Seção 4.3.3 para garantir que as gramáticas geradas sejam consistentes.

4.5.1 Cruzamento

Tradicionalmente, o cruzamento em PG é feito através da troca de partes de dois indivíduos escolhidos previamente. As partes são selecionadas aleatoriamente, formando subárvores que são trocadas entre os indivíduos. A Figura 18 mostra o efeito do cruzamento em dois indivíduos. A única preocupação reside no fato de que as funções estejam aptas a lidar com qualquer tipo ou valor como argumento. Em PG isto é denominado propriedade de fechamento (*closure*) do conjunto de funções (KOZA, 1992). Um exemplo típico é a divisão aritmética, que em PG é adotada de forma "protegida", isto é, ao se dividir por zero, convenciona-se 1 como resposta. Apesar de incoerente, tal "proteção" não inviabiliza a convergência do método.

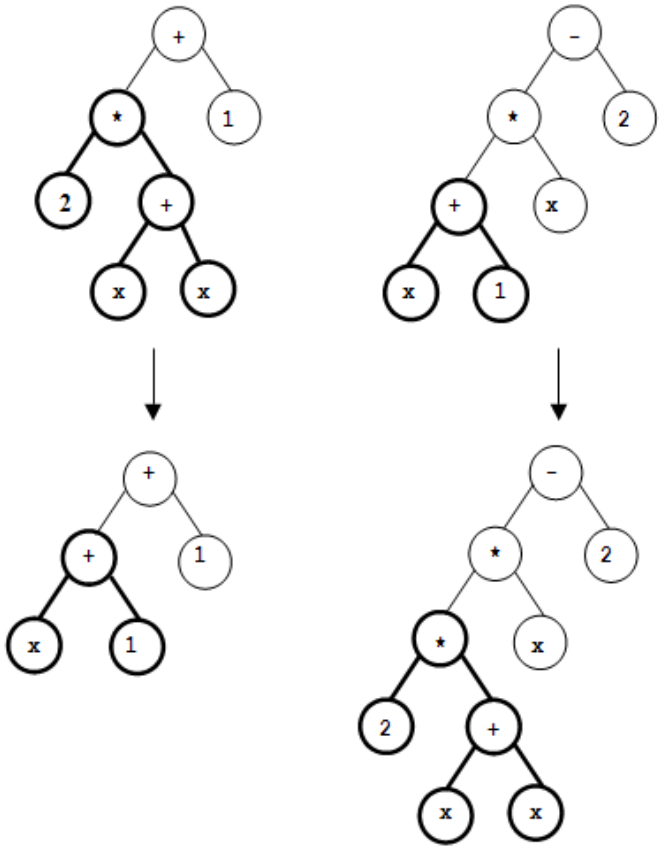

Figura 18: Exemplo de aplicação do operador de cruzamento em PG tradicional.

A representação de gramática proposta neste trabalho não segue o formato clássico dos indivíduos em PG, isto é, cada indivíduo é uma lista de produções em forma de árvore e não uma única árvore. Sendo assim, o operador de cruzamento deve operar de forma diferenciada. Isto é, deve haver apenas a troca de uma produção entre as gramáticas e não de um conjunto delas, como normalmente seria feito. A razão para este comportamento diferenciado reside no fato das produções poderem estar inter-relacionadas, isto é, a troca eliminar a única produção que tem o símbolo não-terminal α no lado esquerdo que é referenciado em outras produções.

Exemplo 4.2

O efeito destrutivo que o cruzamento pode causar em duas gramáticas está ilustrado na Figura 19. Na gramática gerada G_1', existe a produção $P_1 \rightarrow A\ P_3$. Porém, não há mais nenhuma produção começando com P_3. De forma semelhante, na gramática G_2' existe uma produção inútil: $P_3 \rightarrow P_1\ A$, pois o símbolo não-terminal P_3 não é referenciado por nenhuma outra produção de G_2'.

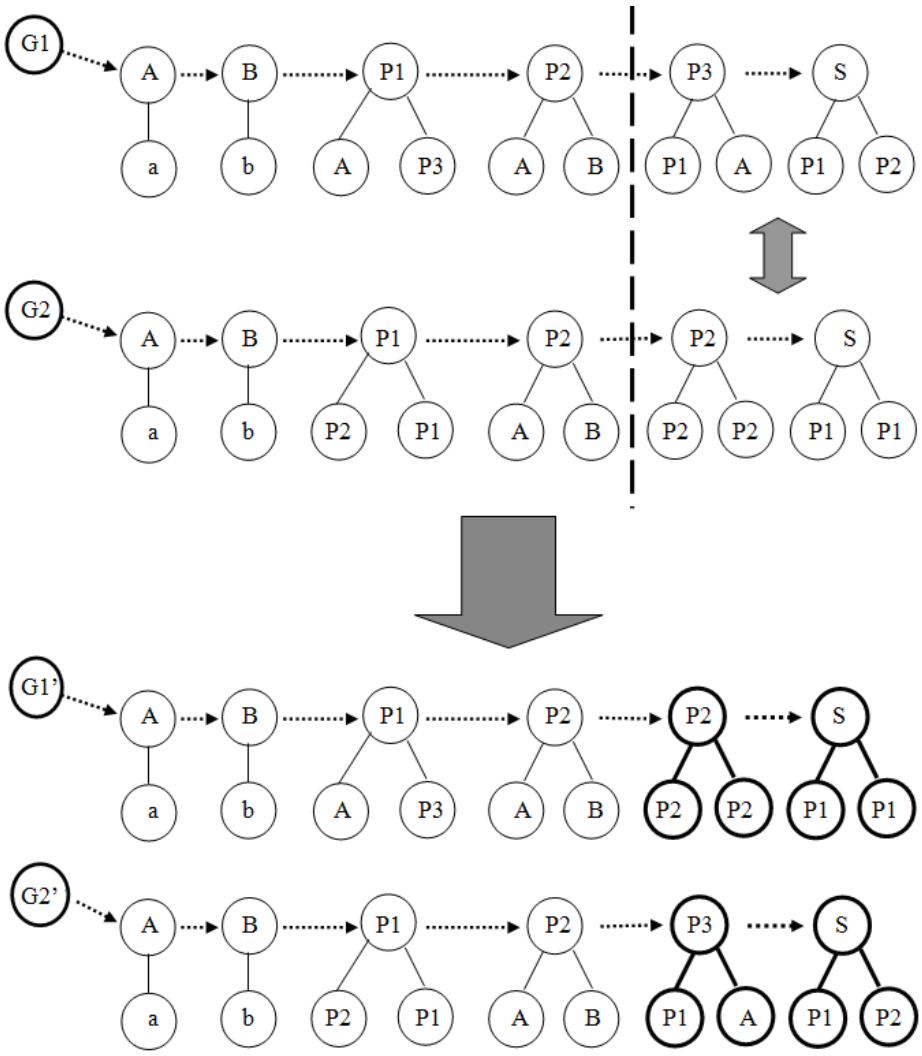

Figura 19: Exemplo de efeito destrutivo do cruzamento entre gramáticas.

Portanto, a troca de produções não deve gerar gramáticas incompletas[14] ou inconsistentes[15]. Para evitar este efeito destrutivo do cruzamento, o algoritmo proposto está na Figura 20. As gramáticas escolhidas para cruzamento estão representadas como $G_1 = (\Sigma_1, N_1, P_1, S)$ e $G_2 = (\Sigma_2, N_2, P_2, S)$. A função `seleciona` escolhe uma gramática da população atual P com base no método de seleção (Seção 4.4), a função `tamanho` retorna o

[14] Entende-se por gramática incompleta aquela que contém símbolos não-terminais que não derivam símbolos terminais.

[15] Entende-se por gramática inconsistente aquela que apresenta símbolos não-terminais que ocorrem no lado direito das produções, mas não ocorrem em nenhum lado esquerdo.

número de elementos em um conjunto e a função `random` escolhe aleatoriamente um elemento do conjunto.

```
G1 = seleciona(P)
G2 = seleciona(P)
P' = ∅
P'' = ∅
menor = 1
Para cada p = { α → β δ } ∈ P1 e α, β, δ ∈ N2 faça
    Se c(p) < 0 Então P' = P' ∪ { p }
    Se c(p) < menor Então menor = c(p)
Fim Para
Se tamanho(P') = 0
   Então Para cada p = {α → β δ} ∈ P1 e α, β, δ ∈ N2 faça
            Se c(p) = menor Então P' = P' ∪ { p }
         Fim Para
Fim Se
p1 = random(P')
menor = 1
Para cada p = {α → β δ} ∈ P2 e α é o símbolo a direita de p1 faça
    Se c(p) < menor Então menor = c(p)
Fim Para
Para cada p= {α → β δ} ∈ P2 e α é o símbolo a direita de p1 faça
    Se c(p) = menor Então P'' = P'' ∪ { p }
Fim Para
p2 = random(P'')
p1 = P1 - { p1 }
p1 = P1 ∪ { p2 }
p2 = P2 - { p2 }
p2 = P2 ∪ { p1 }
```

Figura 20: Pseudocódigo de aplicação do operador de cruzamento[16].

A razão da escolha da produção de menor valor de $c(p)$ se deve ao fato de que não se deseja piorar a gramática. A troca de uma produção muito útil poderia prejudicar a

[16] Para facilitar a legibilidade do pseudocódigo, foram omitidos alguns "Fim Se".

convergência do método. As exigências de que os símbolos não-terminais já existam nas gramáticas serve para garantir que as novas gramáticas sejam consistentes.

Exemplo 4.3

Na Figura 21 apresenta-se um exemplo da aplicação do operador de cruzamento, onde a produção $P_2 \rightarrow P_1\ P_2$ é escolhida na primeira gramática (G_1) com base no critério do menor valor de $c(p)$. Em seguida, a produção $P_2 \rightarrow B\ P_1$ é escolhida na segunda gramática (G_2) seguindo os critérios de mesmo símbolo não terminal (P_2) e o menor valor de $c(p)$. Após a escolha das produções, faz-se a troca entre as gramáticas. Se alguma das gramáticas geradas for inconsistente, a troca das produções não é feita.

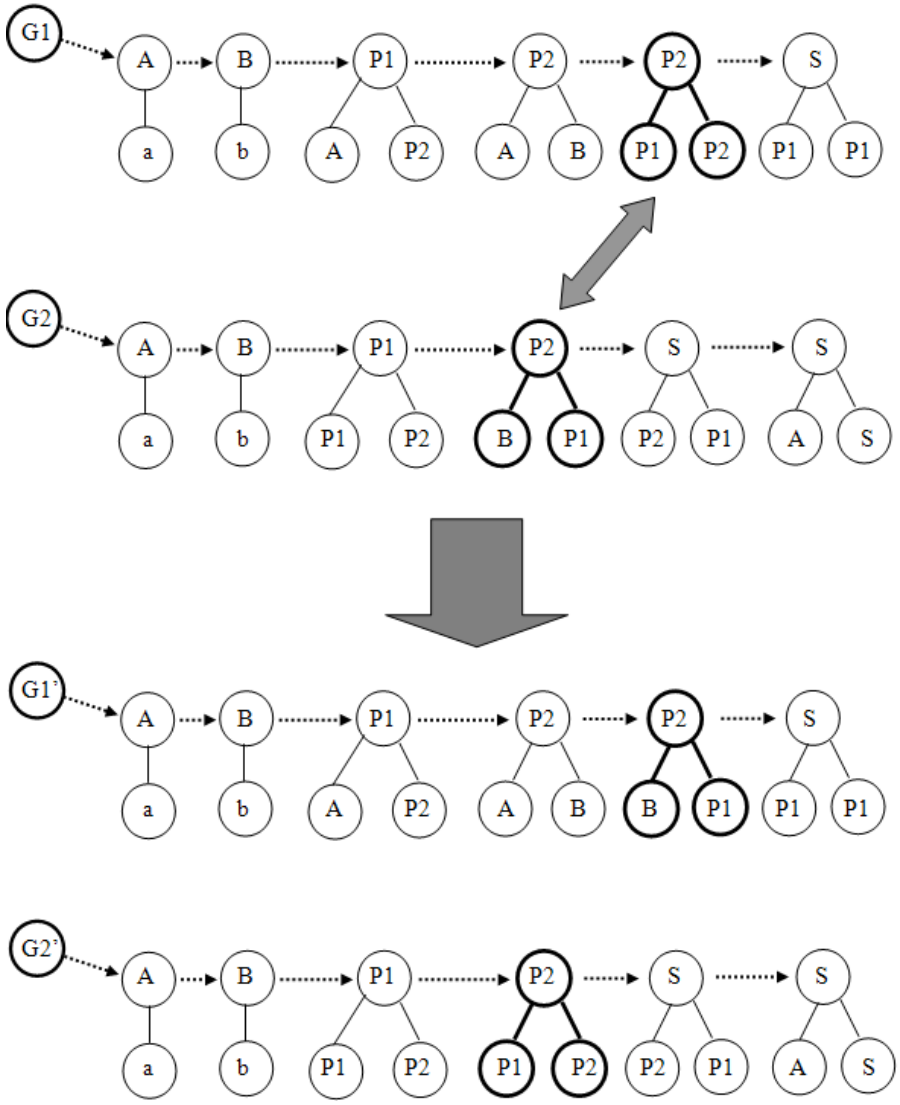

Figura 21: Exemplo de aplicação do cruzamento proposto.

4.5.2 Mutação

Tradicionalmente, a mutação em PG é feita através da troca de parte de um indivíduo escolhido previamente. É um processo semelhante ao cruzamento, exceto que neste caso a subárvore que substituirá é gerada aleatoriamente, não vindo de nenhum outro indivíduo.

O objetivo do operador de mutação é promover diversidade genética na população através de inclusão de partes novas nos indivíduos. Da mesma forma que o cruzamento, não é possível aplicá-lo indiscriminadamente, sob pena de gerar gramáticas inconsistentes.

Neste trabalho, o operador de mutação funciona conforme o pseudocódigo da Figura 22. A função `seleciona` escolhe uma gramática da população atual P com base no método de seleção (Seção 4.4), a função `tamanho` retorna o número de elementos em um conjunto, a função `random` escolhe aleatoriamente um elemento do conjunto e a função `head` retorna o símbolo não-terminal à esquerda da produção, isto é, se $p = \alpha \rightarrow \beta\delta$, então `head(p)` $= \alpha$. Por sua vez, a função `build` constrói uma produção, isto é, `build(`α`, `β`, `δ`)` gera a produção $\alpha \rightarrow \beta\ \delta$.

```
G = seleciona(P)
P' = ∅
Para cada p = { α → β δ } ∈ P faça
    Se c(p) < 0
        Então P' = P' ∪ { p }
    Fim Se
Fim Para
Se tamanho(P') = 0
 Então Encerre
Fim Se
p = random(P')
G = G - {p}
h = head(p)
d1 = random(N)
d2 = random(N)
p = build(h, d1, d2)
G = G ∪ {p}
```

Figura 22: Pseudocódigo de aplicação do operador de mutação.

A razão pela qual o operador de mutação é encerrado caso não tenha nenhuma produção com *c*(p) negativo se deve ao fato de não se desejar perder uma produção que cubra mais exemplos positivos que negativos. É importante frisar que o operador de mutação não adiciona novas produções à gramática, simplesmente altera uma das existentes. Um exemplo da aplicação do operador de mutação está na Figura 23.

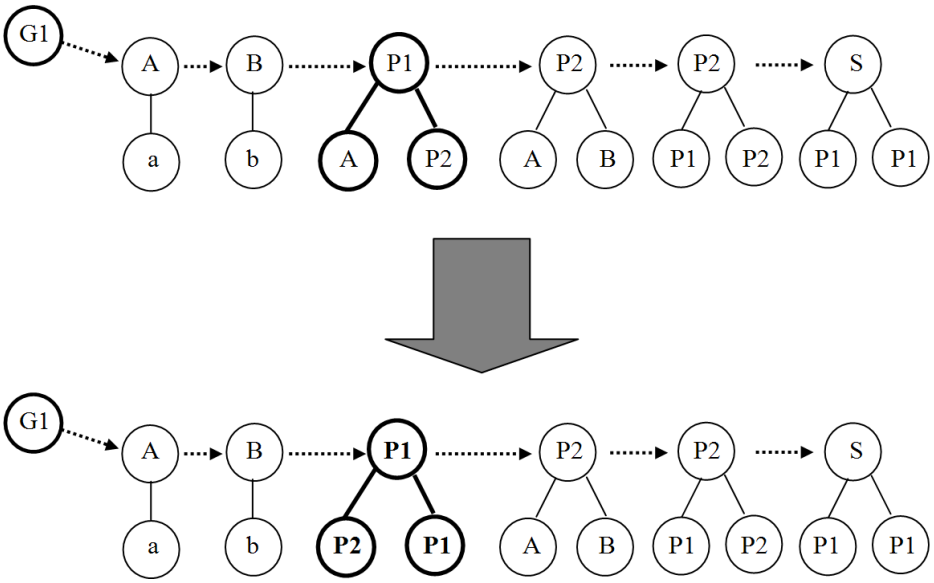

Figura 23: Exemplo de aplicação da mutação.

4.5.3 Aprendizagem Incremental

Os operadores de cruzamento e mutação são operadores globais e, portanto, incapazes de fazer otimização local, tal como adicionar uma produção que está faltando para se obter uma solução. No caso do operador de mutação onde uma nova produção é criada, nem sempre a nova produção é útil para aumentar a cobertura de exemplos positivos. Além disto, somente com o valor de *fitness* não é possível saber quais produções estão faltando. Com base nisto, um novo operador denominado *Incremental Learning* (Aprendizagem Incremental) foi proposto para fazer a busca local (RODRIGUES e LOPES, 2006). Este operador atua de forma similar ao método Synapse (NAKAMURA, 2002). Este operador é aplicado sempre antes da avaliação de cada gramática na população. Para cada exemplo positivo, aplica-se o algoritmo da Figura 24, onde V é a matriz triangular obtida na aplicação do algoritmo CYK sendo V_{rs} o conjunto de produções presentes na coluna *r* e na linha *s*.

```
Monte a matriz V com o CYK com base na gramática G
Se ∃ { S → α β } ∈ V1n
    Então
        Encerre
Fim Se
Se ∃ { α → β δ } ∈ V1n
    Então
        G = G ∪ { S → β δ }
        Encerre
Fim Se
m = n - 1
Enquanto m > 0 faça
    Se ∃ p ∈ V1 m
        Então
            Se ∃ p ∈ Vm+1 n-m
                Então
                    G = G ∪ { S → V1 m  Vm+1 n-m }
                    Encerre
            Fim Se
    Fim Se
    m = m -1
Fim Enquanto
```

Figura 24: Pseudocódigo de aplicação do operador de aprendizagem incremental.

O objetivo do operador de aprendizagem incremental é obter uma produção partindo do símbolo inicial que esteja faltando para cobrir o exemplo sendo analisado com base na matriz obtida pelo algoritmo CYK.

Exemplo 4.4

Seja uma gramática G com o conjunto de produções P = {S → P_4 P_2; P_1 → a; P_2 → b; P_3 → P_2 P_1; P_4 → P_1 P_3; P_5 → P_1 P_2}. Deseja-se verificar se a sentença "ababab" é reconhecida por G. A matriz CYK montada é mostrada na Figura 25. Como o símbolo inicial S não está presente em $V_{1n} = V_{16} = \varnothing$, significa que a sentença não é coberta por G.

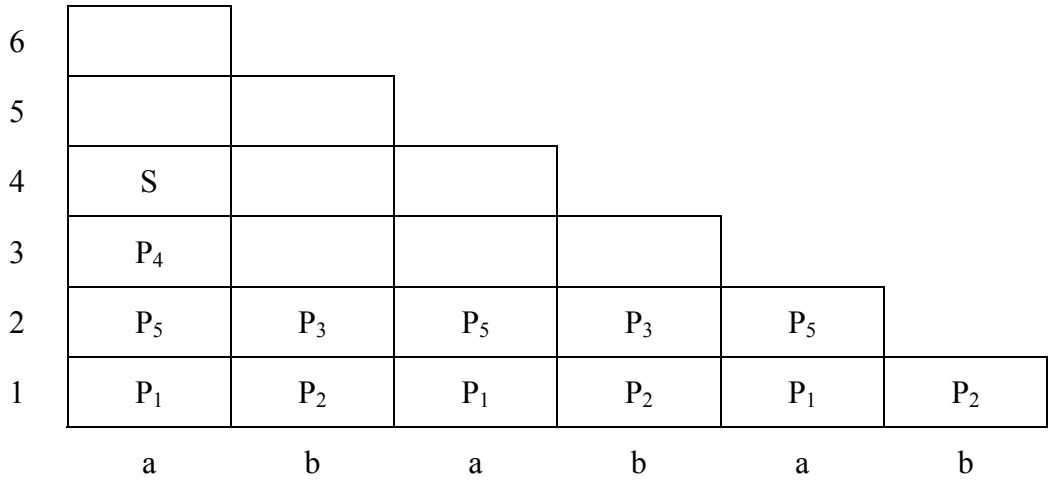

Figura 25. Matriz CYK para a sentença “ababab”.

Ao aplicar o operador de aprendizagem incremental, descobre-se que uma nova produção ($S \rightarrow S\ P_5$) deve ser adicionada à gramática, fazendo com que ela reconheça a sentença. Ao montar novamente a matriz CYK com a nova produção, como S está presente em $V_{1n} = V_{16} = \{S\}$, significa que a sentença é coberta. A matriz correspondente é mostrada na Figura 26.

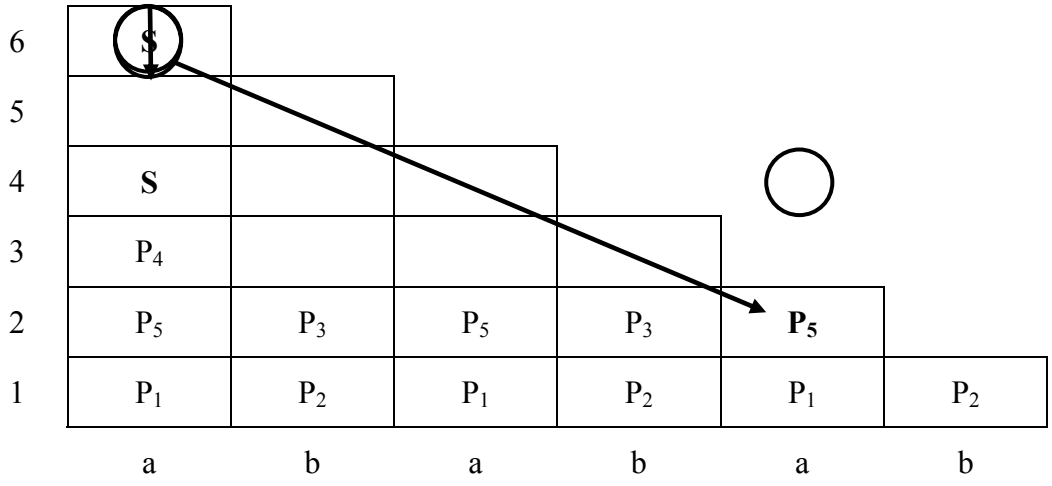

Figura 26. Matriz CYK para a sentença “ababab” com a inclusão da nova produção.

Quanto à complexidade, no melhor caso, onde já exista um símbolo não-terminal em V_{1n}, o operador é instantâneo, pois não adiciona nenhuma produção. No pior caso, o operador de aprendizagem terá que verificar todos os elementos da primeira coluna da matriz CYK em busca de um símbolo não-terminal. Como o número de linhas da primeira coluna é igual a *n*, isto é, ao tamanho da sentença, o algoritmo tem complexidade O(*n*) no pior caso.

Também é proposta uma variante mais poderosa deste operador que pode ser aplicada quando o algoritmo anterior não adicionar nenhuma produção. No caso do exemplo apresentado na Figura 25, o operador falha se não existir previamente a produção $P_5 \rightarrow P_1\ P_2$, isto é, ao encontrar um símbolo não-terminal em V_{1m}, não há um símbolo não-terminal em $V_{m+1\ n-m}$. Para tanto, há a necessidade de se adicionar mais do que uma produção à gramática.O pseudocódigo é apresentado na Figura 27.

```
Para r = 2 até n-1 faça
    m = n - r
    Enquanto m > 1 faça
        Se existe algum não-terminal em V2m
          Então
            Se existe algum não-terminal em Vm+2 n-m-1
              Então
                Adicione um novo símbolo não-terminal X
                Adicione a produção X → V2m Vm+2 n-m-1
            Fim Se
        Fim Se
        m = m – 1
    Fim Enquanto
Fim Para
```

Figura 27. Variante do operador de aprendizagem incremental.

Se, após a execução deste pseudocódigo, alguma produção for adicionada à gramática, então o operador de aprendizagem incremental é aplicado novamente. Desta forma, a gramática será capaz de cobrir o exemplo sendo analisado.

4.5.4 Expansão

O número de produções de cada gramática é determinado na criação da população inicial. Exceto pela atuação do operador de aprendizagem incremental, não há nenhum mecanismo de expansão do conjunto de produções. Desta forma, se alguma produção necessária não estiver presente e não for possível obtê-la pelo operador de aprendizagem incremental, o algoritmo apresentado é incapaz de criá-la espontaneamente. Em experimentos com gramáticas de maior complexidade de formação tais como palíndromos (Seção 4.2), observou-se que, em alguns casos, a convergência é prejudicada pelo fato da gramática ser incapaz de gerar novas produções espontaneamente.

Sendo assim, um novo operador denominado *Expansion* (Expansão) foi proposto (RODRIGUES e LOPES, 2007). Ele atua segundo o pseudocódigo na Figura 28. A função `random` escolhe aleatoriamente um elemento do conjunto.

```
Crie aleatoriamente um símbolo não-terminal α | α ∉ N
β = random(N)
δ = random(N)
γ = random(N)
N = N ∪ { α }
P = P ∪ { α → β δ }
P = P ∪ { S → α γ }
```

Figura 28. Algoritmo de aplicação do operador de expansão

A produção $S \rightarrow \alpha\ \gamma$ é adiciona na gramática para que a nova produção não seja eventualmente eliminada pela validação da população (Seção 4.3.3). Portanto, é importante frisar que o operador de expansão adiciona duas novas produções à gramática.

Exemplo 4.5

Na Figura 29 tem-se um exemplo da aplicação do operador de Expansão, onde o novo símbolo não-terminal criado é P_3. Em seguida duas produções novas são criadas: $P_3 \rightarrow P_2\ P_1$ e $S \rightarrow P_3\ B$.

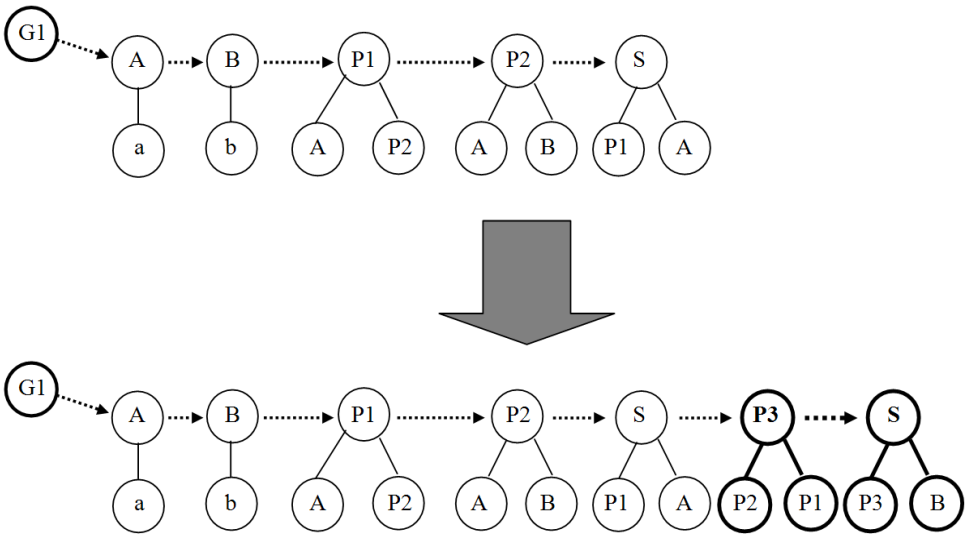

Figura 29: Exemplo de aplicação do operador de expansão.

Uma desvantagem deste operador reside no fato de aumentar a quantidade de produções, podendo promover um aumento descontrolado no tamanho das gramáticas. Para evitar este aumento, deve-se estabelecer um limite máximo de quantidade de produções.

4.6 ADEQUAÇÃO PARA INFERÊNCIA DE GRAMÁTICAS FUZZY

Nos testes realizados, normalmente após um determinado número de execuções, é possível encontrar uma gramática que reconheça todos os exemplos positivos. Porém, ela não é capaz de rejeitar todos os negativos. Isto é percebido quando o valor da sensibilidade atinge um e o valor da especificidade é menor do que um e não melhora.

Uma abordagem possível é verificar quais produções participam do reconhecimento de exemplos negativos e não participam do reconhecimento de exemplos positivos, bastando eliminá-las da gramática. O problema surge quando estas produções participam tanto do reconhecimento de exemplos positivos quanto de negativos. Neste caso, deve-se escolher a melhor gramática, isto é, a que apresenta a melhor taxa de acerto e dotá-la de algum mecanismo de "ajuste fino", isto é, usar um modelo gramatical aproximado. O mecanismo proposto neste trabalho é a gramática *fuzzy* (MORDESON e MALIK, 2002).

4.6.1 Representação do indivíduo

Para adequar a representação, associa-se a melhor gramática encontrada a um vetor de valores reais no intervalo [0, 1]. O problema reside na determinação dos seus valores. Uma abordagem usada é a apresentada na Equação 19, onde cada produção recebe um valor proporcional a sua participação na cobertura de exemplos positivos (KUBICEK e ZENDULKA, 2002).

$$\mu(p) = \frac{p^+}{p^+ + p^-} \qquad (19)$$

Onde p^+ é o número de vezes em que a produção p participa da cobertura de exemplos positivos e p^-, da cobertura de negativos. Desta forma, o valor de μ estará no intervalo de 0 a 1, sendo que quanto mais próximo de 1, mais útil é a produção no reconhecimento de exemplos positivos. Porém, por levar em conta apenas os exemplos positivos, uma produção que seja usada em todos os exemplos tanto positivos quanto negativos receberá o valor máximo de μ, o que não é desejável. Para contornar este problema, um tipo de gramática *fuzzy* denominada fracionária (*fractionally fuzzy grammar*) pode ser usada.

A gramática *fuzzy* fracionária foi inicialmente proposta por De Palma e Yau (1975) para reconhecimento de padrões e se baseia em um cálculo de pertinência das sentenças diferente da abordagem clássica de gramáticas *fuzzy*.

Definição 4.3

A gramática fuzzy fracionária é a sétupla G = (N, Σ, P, S, J, *g*, *h*), onde N é o conjunto de símbolos não-terminais, Σ é o conjunto de símbolos terminais, P é o conjunto de produções, S é o símbolo inicial e J, o conjunto de indexadores de cada produção. As funções *g* e *h* mapeiam os elementos de J em valores inteiros positivos incluindo o zero, tal que $g(p_i) \leq h(p_i)\ \forall\ p_i \in J$. A pertinência de uma sentença é dada pela Equação 20. Neste cálculo, convenciona-se que a divisão por zero vale zero.

$$\mu(x) = \sup\left\{ \frac{\sum_{j=1}^{l_k} g\left(p_j^k\right)}{\sum_{j=1}^{l_k} h\left(p_j^k\right)} \right\} \qquad k \in \{ 1, 2, 3,..., m \} \qquad (20)$$

Exemplo 4.6

Considere que a gramática *fuzzy* fracionária G seja definida pelos conjuntos $\Sigma = \{a, b\}$, $N = \{S\}$, S é o símbolo inicial e P, J, *g* e *h* definidos como[17]:

p_1: $S \rightarrow ab$	$g(p_1) = 1$	$h(p_1) = 1$
p_2: $S \rightarrow aSb$	$g(p_2) = 1$	$h(p_2) = 1$
p_3: $S \rightarrow aS$	$g(p_3) = 0$	$h(p_3) = 1$
p_4: $S \rightarrow Sb$	$g(p_4) = 0$	$h(p_4) = 1$

Considere a sentença a^3b^5. Todas as seqüências possíveis de se gerar tal sentença são as seguintes: $\{p_2, p_2, p_4, p_4, p_1\}$, $\{p_2, p_3, p_4, p_4, p_4, p_1\}$ e $\{p_3, p_3, p_4, p_4, p_4, p_4, p_1\}$. Aplicando a Equação 20, obtém-se $\mu(a^3b^5) = sup\ \{ 3/5, 2/5, 1/5 \} = 3/5$. Ao analisar esta gramática, o primeiro par de produções deriva as sentenças a^nb^n onde $n > 0$ e o segundo par, permitem sentenças com quantidades diferentes de a's e b's. Quanto *n* for mais próximo de *m*, maior a pertinência. Isto se deve ao fato de $g(p) = h(p)$ para o primeiro par de produções e $g(p) < h(p)$ para o segundo. Portanto, é possível concluir que esta gramática gera a linguagem $\{a^nb^m \mid n, m \in \mathbf{N}^+\}$, onde $\mathbf{N}^+$ é o conjunto dos números naturais exceto o zero, com pertinência $\mu(a^nb^m) = inf\ \{n, m\}\ /\ sup\ \{n, m\}$.

Neste trabalho, adotou-se que a função $h(p) = 1$ se a produção $p \in P$ participar da cobertura de ao menos um exemplo positivo. A função *g* será dada pelo valor de crédito *c* de cada produção (Equação 18) com a modificação apresentada na Equação 21. Desta forma, os valores de $g(p)$ serão valores reais positivos e os valores de $h(p) \in \{0, 1\}$.

[17] Exemplo retirado do livro de MORDESON e MALIK, 2002.

$$g(p) = \begin{cases} \dfrac{p^+ - p^-}{p^+ + p^-} & se \quad p^+ > p^- \\ \\ 0 & caso \quad contrário \end{cases} \qquad (21)$$

4.6.2 Determinação do limiar[18]

Após a determinação dos valores de *g*(p) e *h*(p) para cada p ∈ P, calcula-se o grau de pertinência μ de cada exemplo positivo e negativo para cada gramática usando a Equação 20. Deseja-se, idealmente, que a gramática *fuzzy* aceite cada exemplo positivo e rejeite os negativos. Nos testes feitos, o melhor valor possível de *c* capaz de aceitar a maioria dos exemplos positivos e rejeitar a maioria dos negativos foi a média entre os valores encontrados.

Em alguns casos, pode ocorrer que exista mais de uma gramática com a melhor taxa de acerto. Neste caso, não há apenas uma gramática a ser transformada em gramática *fuzzy*, mas sim um conjunto delas, isto é, haverá uma população de gramáticas, cada uma com seu valor *c*. O conjunto destas gramáticas forma, então, um arranjo de classificadores (TAN, GILBERT e DEVILLE, 2003; NERBONNE, BELZ, CANCEDDA *et al*, 2001).

A classificação, neste caso, pode ser feita da seguinte forma (WITTEN e FRANK, 2005):

1. Submete-se a sentença a classificar a cada uma das gramáticas que compõem o arranjo;
2. Contabilizam-se quantas gramáticas cobriram e quantas rejeitaram com base no limiar de cada uma delas;
3. Se a quantidade de gramáticas que cobriram for maior do que as que rejeitaram, a sentença é aceita. Caso contrário, rejeitada.

Desta forma a classe mais votada é apresentada usando um processo semelhante ao de *bagging* (WITTEN e FRANK, 2005).

[18] O limiar se refere a gramáticas *fuzzy* fracionárias (Definição 4.3)

CAPÍTULO 5

EXPERIMENTOS

Neste capítulo são apresentados os resultados obtidos pela aplicação do modelo proposto inferência de gramáticas regulares, gramáticas livres de contexto e em dois problemas de reconhecimento de padrões em dados biológicos (DNA). O modelo foi desenvolvido em C++ em código multi-plataforma e recebeu o nome de *Genetic Programming for Grammatical Inference* (GPGI).

5.1 INFERÊNCIA DE GRAMÁTICAS REGULARES

Para a validação do modelo na inferência de gramáticas regulares, foram utilizadas as linguagens apresentadas na Tabela 4 propostas por Luke, Hamahashi e Kitano (1999) em uma abordagem de computação evolucionária.

Tabela 4: Linguagens usadas para validação.

Linguagem	**Descrição**
1	1^+ Qualquer seqüência de 1's
2	$(10)^+$ Qualquer seqüência do par 10
3	(0|11)*(1*|(100(00|1)*)) Qualquer seqüência sem um número ímpar de 0's consecutivos depois de um número ímpar de 1's consecutivos
4	1*((0|00)11*)*(0|00|1*) Qualquer seqüência sem mais de três 0's consecutivos
5	(((1|0)(1|0))*(1|0))|((11|00)*((01|10)(00|11)*(01|10)(00|11)*)*(11|00)*) Qualquer seqüência de tamanho par no qual, ao conter pares, tem um número par de 01 ou 10
6	((0(01)*(1|00))|(1(10)*(0|11)))* Qualquer seqüência em que a diferença entre a quantidade de 1's e 0's é um múltiplo de três

O conjunto de treinamento usado para cada uma das linguagens está na Tabela 5 e foi baseada em Tomita (1982). Os parâmetros utilizados, por sua vez, estão apresentados na Tabela 6. Para este problema em particular, a população inicial foi criada de forma aleatória e a avaliação de *fitness* utilizada foi o produto da sensibilidade pela especificidade (Equação 16).

Tabela 5: Conjunto de exemplos usados para cada linguagem.

Linguagem	**Exemplos Positivos**	**Exemplos Negativos**
1	1, 11, 111, 1111, 11111, 111111, 1111111, 11111111	0, 10, 01, 00, 011, 110, 11111110, 10111111
2	10, 1010, 101010, 10101010, 10101010101010	1, 0, 11, 00, 01, 101, 100, 1001010, 10110, 110101010
3	1, 0, 01, 11, 00, 100, 110, 111, 000, 100100	10, 101, 010, 1010, 1110, 10001, 111010, 1001000, 1111000
4	1, 0, 10, 01, 00, 100100, 001111110100, 0100100100, 11100, 0010	000, 11000, 0001, 000000000, 11111000011, 1101010000010111, 1010010001, 0000, 00000
5	11, 00, 1001, 0101, 1010, 1000111101, 1001100001111010, 111111, 0000	0, 111, 010, 000000000, 1000, 01, 10, 1110010100, 010111111110, 0001, 011
6	10, 01, 1100, 101010, 111, 000000, 10111, 0111101111, 100100100	1, 0, 11, 00, 101, 011, 11001, 1111, 00000000, 010111, 101111011111, 1001001001

Tabela 6: Parâmetros usados para inferência de gramáticas regulares.

Parâmetro	**Valor**
Número de execuções independentes	50
Número máximo de gerações	50
Tamanho da população	100
População Inicial	
Número Mínimo de Produções	10
Número Máximo de Produções	25
Tamanho do Torneio	7
Probabilidade de cruzamento	60%
Probabilidade de mutação	30%
Probabilidade de expansão	10%
Elitismo	Sim

5.1.1 Resultados Obtidos

Os resultados da aplicação do modelo proposto para cada uma das linguagens estão na Tabela 7. Os valores mínimo, médio e máximo de gerações se referem apenas às execuções que obtiveram sucesso.

Tabela 7: Resultados obtidos para gramáticas regulares.

Linguagem	Execuções com Sucesso	Variação no número de gerações nas execuções com sucesso		
		Mínimo	Média	Máximo
1	50	1	0,4	2
2	50	1	0,3	2
3	12	28	39	48
4	16	39	42	44
5	7	42	46	49
6	50	9	29	38

Na Tabela 8 comparam-se os resultados obtidos com os reportados por Luke, Hamahashi e Kitano (1999). Apesar de não obter sucesso em todas as execuções, a abordagem proposta apresentou resultados melhores, tendo um número maior de execuções em que obteve a solução.

Tabela 8: Número de execuções com sucesso de um conjunto de 50 execuções feitas.

Linguagem	Luke, Hamahashi e Kitano	GPGI
1	31	50
2	7	50
3	1	12
4	3	16
5	0	7
6	47	50

Nas execuções que tiveram sucesso, a forma e a quantidade de produções de cada solução encontrada variaram muito. Em algumas execuções, a gramática final apresentou produções sem utilidade ou redundantes. Por exemplo, no caso da linguagem 2 (qualquer seqüência de pares 10), as soluções que foram encontradas nas primeiras três execuções feitas estão na Tabela 9.

Tabela 9: Exemplos de soluções encontradas para a linguagem 2

Execução	Solução Encontrada
1	A → 1 ; B → 0 ; S → A B ; S → S S
2	A → 1 ; B → 0 ; P1 → A B ; P3 → A B ; P4 → P3 P1 ; S → P1 P4 ; S → P4 P4 ; P4 → P1 S ; S → P1 P1 ; S → A B
3	A → 1 ; B → 0 ; P1 → A B ; P2 → A B ; P3 → A B ; P4 → P3 P2 ; S → P1 P3 S → P1 P4 ; P3 → P3 P4 ; P1 → P3 S ; S → P4 P1 ; P2 → P4 B ; B → P4 A ; S → A B

A primeira solução encontrada contém apenas quatro (4) produções enquanto que a segunda contém onze (11) e a terceira, quatorze (14). Todas as três gramáticas são equivalentes, isto é, por simplificação todas levam a uma conformação semelhante.

5.1.2 Análise dos Resultados

Os valores mínimos, médio e máximo de taxa de acerto para cada geração para as linguagens 3 a 6 estão nas Figuras 30 a 33. A obtenção destes valores foi feita da seguinte forma: para cada geração, identificou-se dentre as 50 execuções, qual foi o valor mínimo, médio e máximo da taxa de acerto obtido. As linguagens 1 e 2 não estão representadas, pois a inferência delas foi muito rápida (no máximo, duas gerações para convergir).

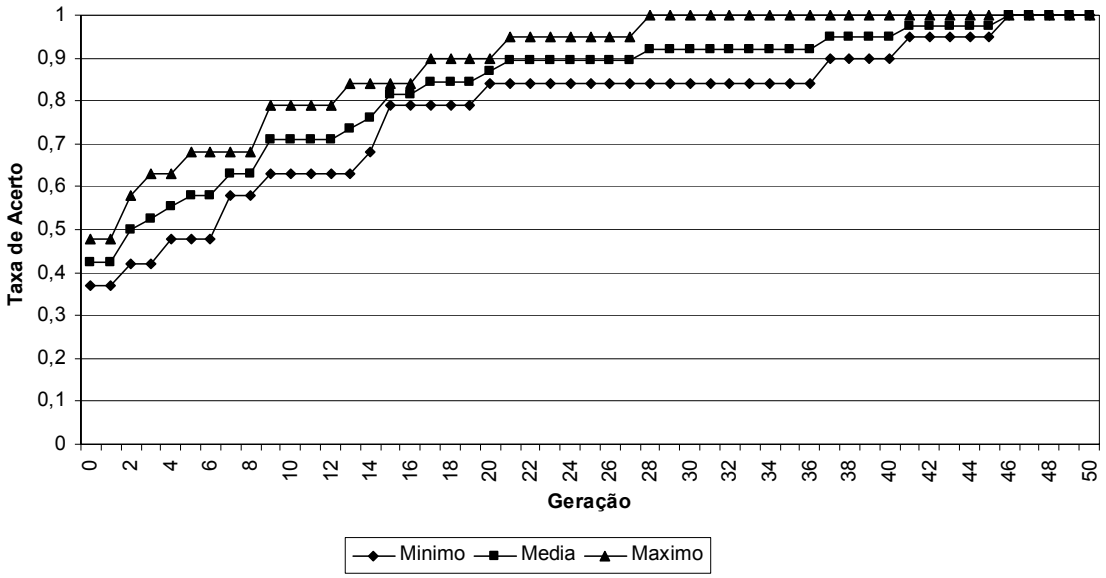

Figura 30: Valores de taxa de acerto por geração para a linguagem 3.

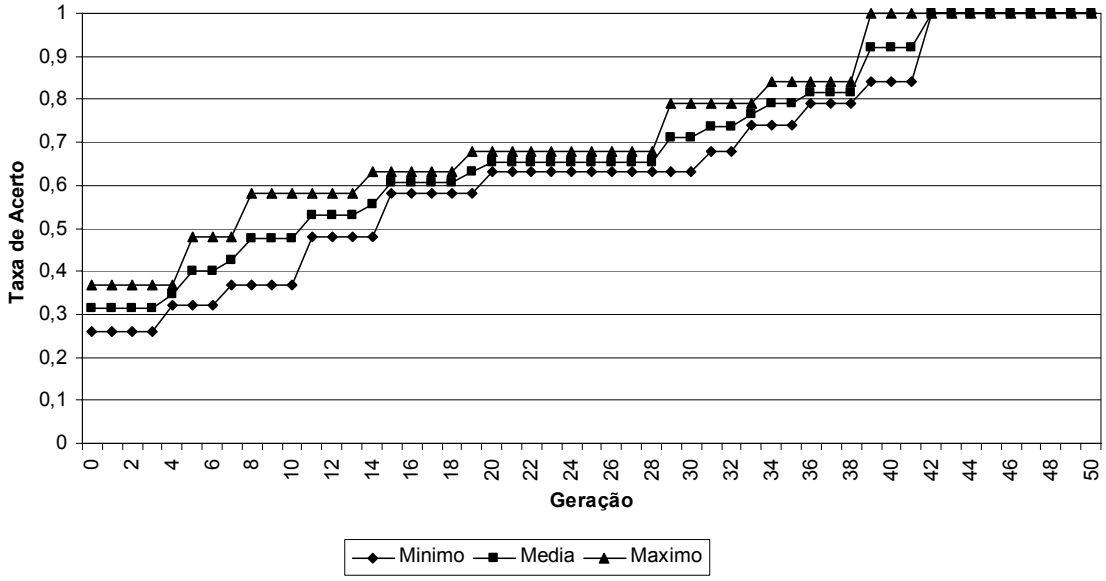

Figura 31: Valores de taxa de acerto por geração para a linguagem 4.

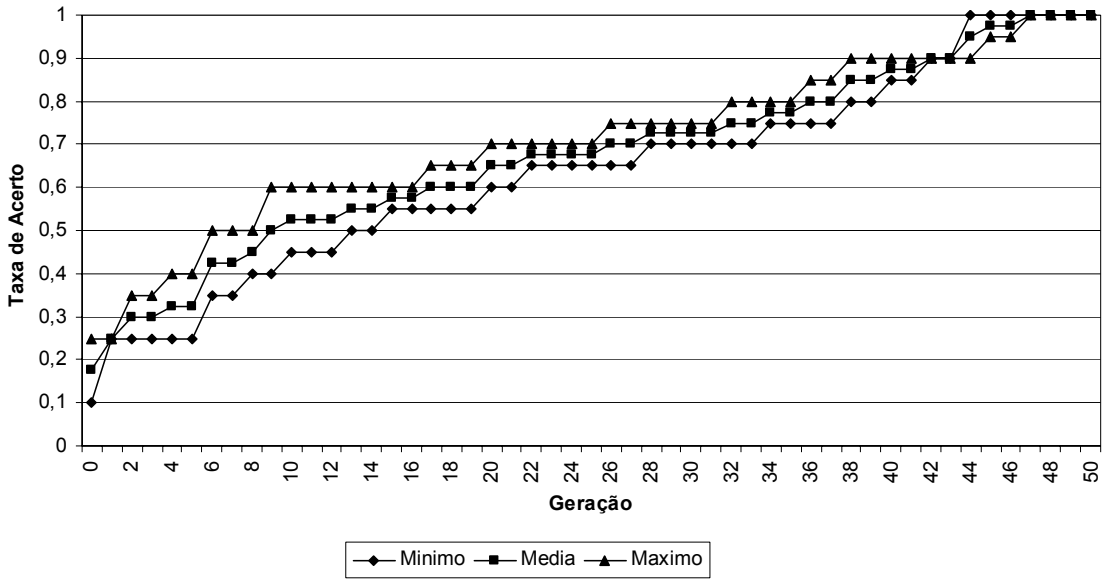

Figura 32: Valores de taxa de acerto por geração para a linguagem 5.

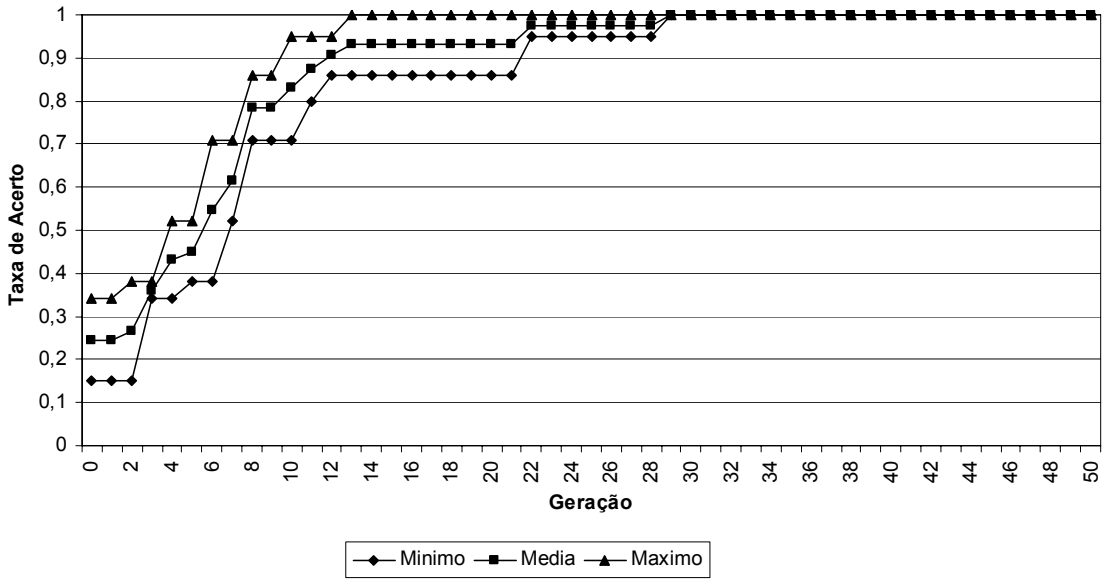

Figura 33: Valores de taxa de acerto por geração para a linguagem 6.

É possível perceber que houve pouca variação nas taxas de acerto para cada geração entre as execuções, isto é, o comportamento foi semelhante independentemente da execução. Isto demonstra que o processo evolutivo não é muito dependente da população inicial.

Com o objetivo de entender como o processo evolutivo está inferindo a gramática, a variação dos valores de sensibilidade (*Sens*), especificidade (*Spec*) e taxa de acerto (*Tacc*) durante o processo evolutivo estão apresentados na Figura 34. Os valores se referem a uma

das rodadas com sucesso para a linguagem 6. Para as outras linguagens, o comportamento foi semelhante.

Nesta figura, é possível perceber que durante as três primeiras gerações, o método procura melhorar a sensibilidade (*Sens*). A especificidade (*Spec*) começa no seu valor máximo (1,0) nas primeiras gerações porque as gramáticas tendem a rejeitar facilmente todos os exemplos negativos. Na terceira geração há uma melhoria significativa na taxa de sensibilidade devido à atuação dos operadores, causando uma queda no valor da especificidade. Isto se deve ao fato de que a melhor gramática começa a cobrir mais exemplos, tanto positivos quanto negativos. Depois da sexta geração, a sensibilidade atinge seu valor máximo (1,0), significando que todos os exemplos positivos agora estão sendo cobertos pela melhor gramática. Porém, ainda há exemplos negativos incorretamente sendo cobertos (especificidade menor que 1,0). A partir deste momento, os operadores de cruzamento, mutação e expansão tornam-se essenciais para a convergência, pois o operador de aprendizagem incremental apenas tende a melhorar a taxa de sensibilidade. Na nona geração ocorre a convergência.

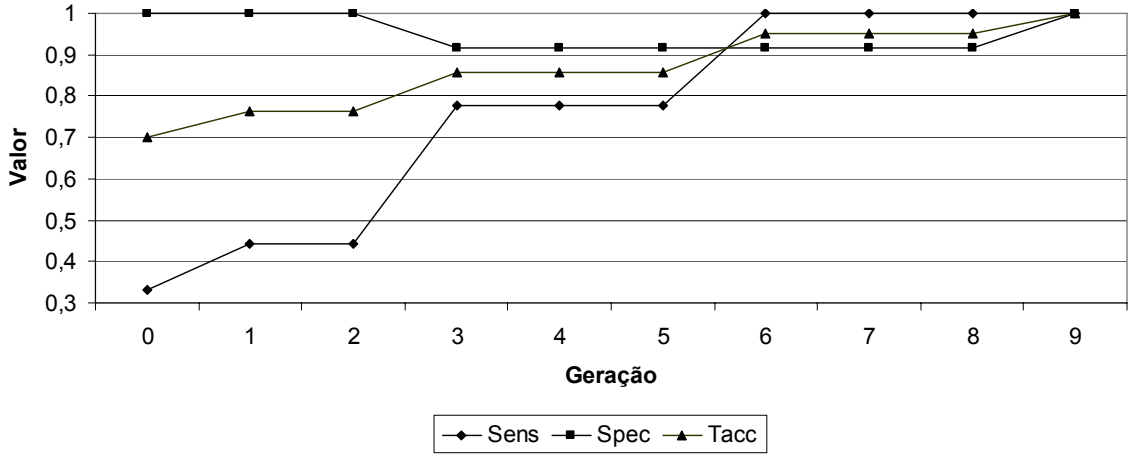

Figura 34: Variação da sensibilidade (*Sens*), especificidade (*Spec*) e taxa de acerto (*Tacc*) da melhor gramática para a linguagem 6.

Para demonstrar a importância dos operadores de aprendizagem incremental e expansão na convergência, a Figura 35 mostra o comportamento do processo evolutivo com e sem estes operadores. A população inicial em todos os casos foi a mesma para permitir a comparação direta.

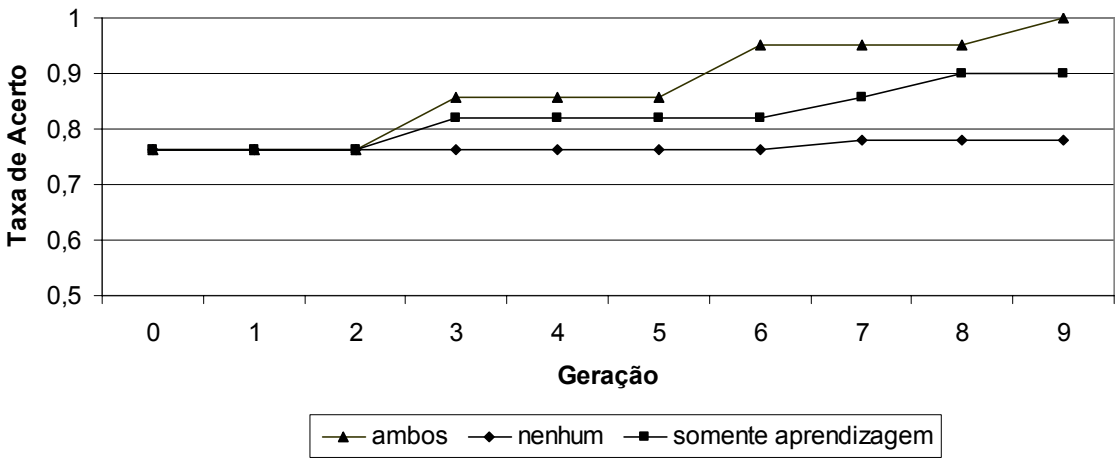

Figura 35: Variação da taxa de acerto para a inferência de uma gramática com e sem os operadores de aprendizagem incremental e expansão.

Sem os operadores de aprendizagem incremental e expansão, isto é, usando apenas os operadores de cruzamento e a mutação, a taxa de acerto não ultrapassa 0,8 nas nove primeiras gerações.

Aplicando apenas o operador de aprendizagem, houve melhorias na terceira, sétima e oitava gerações, contudo a taxa de acerto atinge, no máximo, 0,9 na nona geração. Isto se deve ao fato do operador de aprendizagem ser capaz apenas de melhorar a sensibilidade, sendo necessário que os outros operadores (cruzamento e mutação) melhorarem a especificidade. Portanto, é possível perceber que, a partir da terceira geração, há um trabalho conjunto destes operadores para favorecer a convergência. Utilizando apenas o operador de expansão, o comportamento é idêntico a não utilizar nenhum.

5.2 INFERÊNCIA DE GRAMÁTICAS LIVRES DE CONTEXTO

Basicamente, os parênteses e os palíndromos são usados para validação de inferência de gramáticas livres de contexto. Para caso de palíndromos, quanto maior o alfabeto, mais difícil torna-se a inferência. Para validação do modelo proposto, adotou-se o palíndromo com quatro letras (a, b, c e d) e de dois tipos: o palíndromo comum e o palíndromo disjunto. Na Tabela 10 apresenta-se uma descrição de cada uma delas.

Tabela 10: Gramáticas livres de contexto adotadas.

Linguagem	Descrição
Parênteses	Todas as sentenças que tenham parênteses balanceados tais como (), ()(), (()), (())() e assim por diante.
Palíndromo	$\{vv^R \mid v \in \{ a, b, c, d \}^*\}$ tais como *aa*, *abba*, *baab*, *abccba, bcddcb* e assim por diante.
Palíndromo Disjunto	$\{vw \mid v \in \{a, b, c, d\}^*, w \in z(v) \}$ tais como *abcgfe*, *dh*, *bdhf*. Neste exemplo, a função z mapeia *v* para $\{e, f, g, h\}$.

No caso das gramáticas escolhidas, definir qual o número de exemplos positivos e negativos minimamente necessários para a correta inferência não é uma tarefa trivial. A simples geração aleatória de sentenças não garante um conjunto de exemplos que permita diferenciar a gramática desejada de outra semelhante, porém incorreta.

Exemplo 5.1

Considere que haja a necessidade de encontrar um conjunto de produções da gramática G capaz de reconhecer sentenças onde o número de ocorrências do terminal "a" seja idêntico ao número de ocorrências do terminal "b". Dados os conjuntos de exemplos positivos E^+ = {ab, ba, aabb, abba baab} e negativos E^- = {a, b, aa, bb, aab}, um conjunto possível de produções inferidas para G pode ser:

$$P_1 \rightarrow a \qquad P_2 \rightarrow b \qquad P_3 \rightarrow P_1\ P_1 \qquad P_3 \rightarrow P_2\ P_2$$
$$P_3 \rightarrow P_3\ S \qquad S \rightarrow P_2\ P_1 \qquad S \rightarrow P_1\ P_2 \qquad S \rightarrow P_3\ P_3$$
$$S \rightarrow S\ S$$

Este conjunto de produções corretamente cobre os exemplos em E^+ e rejeita E^-. Inclusive é capaz de cobrir outros exemplos positivos não vistos, tais como "bbaa", o que é desejado. Porém, a sentença "aaab" também é coberta e não pertence à gramática correta. Obviamente, se esta sentença estivesse presente em E^-, a gramática inferida seria outra. Mas a sua presença não garante que a gramática inferida seja a correta.

Assim, o problema é se determinar quais são os exemplos necessários para a correta inferência gramatical, dado que os conjuntos de exemplos são finitos. Este problema também foi encontrado pelos organizadores da Competição Omphalos de inferência gramatical (STARKIE, COSTE e ZAANEN, 2004).

Decidiu-se, então, usar o conjunto de exemplos disponibilizados para a referida competição. Além disto, há a necessidade de se comparar os resultados com abordagens conhecidas para verificar o desempenho do método proposto.

Recentemente, Clark, Florencio e Watkins (2006) apresentaram uma abordagem de inferência gramatical baseada em *String Kernels* (SK) e usaram a base de dados da competição Omphalos. Eles fizeram também comparações com outras abordagens conhecidas, tais como Gramáticas Estocásticas Livres de Contexto (*Stochastic Context Free Grammars*, SCFG) e Modelo Oculto de Markov (*Hidden Markov Models*, HMM).

Neste trabalho optou-se por usar o mesmo conjunto de exemplos e comparações são feitas com os resultados obtidos por Clark, Florêncio e Watkins (2006). O conjunto de exemplos para parênteses tem 537 exemplos positivos e 463 negativos. O conjunto de palíndromos tem 510 exemplos positivos e 490 negativos para ambos os casos (Palíndromos e Palíndromos Disjuntos). Na Tabela 10 estão os parâmetros usados. A alta probabilidade de mutação (25%) foi necessária e será justificada na Seção 4.2.2. A avaliação de *fitness* adotada foi média da sensibilidade pela especificidade (Equação 17).

Tabela 11: Parâmetros usados para inferência de gramáticas livres de contexto.

Parâmetro	**Valor**
Número de execuções independentes	50
Número de partições	10
Número máximo de gerações	50
Tamanho da população	100
População Inicial	
Número Mínimo de Produções	50
Número Máximo de Produções	300
Tamanho do Torneio	7
Probabilidade de cruzamento	70%
Probabilidade de mutação	25%
Probabilidade de Expansão	5%
Elitismo	Sim

Uma dificuldade encontrada na inferência usando o modelo proposto foi o efeito de *overfitting*, isto é, a gramática inferida cobria apenas os exemplos positivos de treinamento, não generalizando como deveria. Este efeito não ocorreu nos experimentos com gramáticas regulares.

Para evitar este efeito, adotou-se uma abordagem de treinamento e teste por validação cruzada denominada *k-fold stratified cross-validation* (WITTEN e FRANK, 2005). Nesta abordagem, o conjunto de todos os exemplos é dividido em k partições mutuamente exclusivas (*folds*) de tamanho aproximadamente igual a n/k exemplos com distribuição proporcional de exemplos positivos e negativos idêntica ao conjunto completo (*stratified*). Os exemplos nas k-1 partições são usados para treinamento e a gramática inferida é testada na partição remanescente. Este processo é repetido k vezes, cada vez considerando uma partição diferente para teste. A taxa de acerto é a média das taxas de acerto em cada uma das k partições usadas. Nos experimentos, adotou-se o valor de $k = 10$, gerando, assim, dez partições.

5.2.1 Resultados Obtidos

A Tabela 12 apresenta os resultados obtidos nas cinqüenta execuções, identificando o número mínimo, máximo e a média de gerações até obter a gramática desejada.

Tabela 12: Número de execuções com sucesso e número de gerações necessário.

Linguagem	**Execuções com Sucesso**	**Número de Gerações Necessárias**		
		Mínimo	Média	Máximo
Parênteses	50	1	2,2	4
Palíndromo	50	21	24	25
Palíndromo Disjunto[19]	42	22	24,5	25

A Tabela 13 apresenta os melhores resultados obtidos em cinqüenta execuções para cada conjunto de exemplos, exceto para a primeira linguagem. Para facilitar uma comparação direta com os resultados apresentados por Clark, Florêncio e Watkins (2006), adotaram-se as mesmas métricas que são Taxa de Erro Negativa (*Negative Error Rate*, NER) e Taxa de Erro Positivo (*Positive Error Rate,* PER), apresentadas nas Equações 22 e 23, respectivamente.

$$NER = \frac{FP}{FP + VN} \qquad (22)$$

$$PER = \frac{FN}{VP + FN} \qquad (23)$$

[19] Os valores de mínimo, média e máximo se referem às 42 execuções de sucesso, isto é, exclui-se as 8 execuções que não convergiram até a 50ª geração.

Para cada conjunto de exemplos, os valores de NER e PER estão apresentados separadamente. Quanto menor os valores, melhor. As colunas SCFG, HMM e SK se referem a resultados obtidos por Clark, Florêncio e Watkins (2006). É importante frisar que apenas a abordagem proposta neste trabalho (GPGI) utilizou a validação cruzada.

Tabela 13: Resultados obtidos na inferência dos três tipos de linguagem.

Problema	**SCFG**		**HMM**		**SK**		**GPGI**	
	NER	PER	NER	PER	NER	PER	NER	PER
Parênteses	0	0	3	1	10	0	0	0
Palíndromos	6	0	84	3	16	0	0	0
Palíndromos Disjuntos	1	0	4	8	0	0	5	0

O método proposto (representado como GPGI na tabela) obteve a gramática correta para o conjunto de exemplos de parênteses em todas as execuções em, no máximo, quatro gerações.

Para os exemplos de palíndromos, a gramática correta foi obtida em, no máximo, 25 gerações. As gramáticas obtidas variam de 8 a 32 produções. A razão pela qual a abordagem proposta foi superior às outras se deve ao fato da população inicial ter sido construída parcialmente com base nos exemplos positivos. Sem isto, o valor de NER atinge 14% de erro (NER), conforme apresentado em Rodrigues e Lopes (2007) onde a abordagem usada foi com a população inicial criada de forma totalmente aleatória.

Com relação aos palíndromos disjuntos, a gramática correta foi obtida em 42 das 50 execuções em, no máximo, 44 gerações. Nas 8 execuções que não obtiveram sucesso, a taxa de acerto variou de 94% a 96%. Nas execuções com sucesso, as gramáticas obtidas variaram entre 20 e 42 regras de produção. Uma das gramáticas com apenas 20 regras de produção é apresentada a seguir:

$P_1 \rightarrow a$	$P_2 \rightarrow b$	$P_3 \rightarrow c$	$P_4 \rightarrow d$
$P_5 \rightarrow e$	$P_6 \rightarrow f$	$P_7 \rightarrow g$	$P_8 \rightarrow h$
$P_{11} \rightarrow S\ P_6$	$P_{18} \rightarrow S\ P_8$	$P_{22} \rightarrow S\ P_5$	$P_{31} \rightarrow S\ P_7$
$S \rightarrow P_1\ P_5$	$S \rightarrow P_4\ P_8$	$S \rightarrow P_3\ P_7$	$S \rightarrow P_2\ P_6$
$S \rightarrow P_1\ P_{22}$	$S \rightarrow P_2\ P_{11}$	$S \rightarrow P_3\ P_{31}$	$S \rightarrow P_4\ P_{18}$

Deste conjunto, as regras de produção $S \rightarrow P_1\ P_5$, $S \rightarrow P_2\ P_6$, $S \rightarrow P_3\ P_7$ e $S \rightarrow P_4\ P_8$ fazem o reconhecimento das seqüências "ae", "bf", "cg" e "dh", A regra de produção $P_{22} \rightarrow S\ P_5$ é atingível unicamente pela regra de produção $S \rightarrow P_1\ P_{22}$, provocando o casamento do símbolo "a" no início com o símbolo "e" no final, tendo um número par de símbolos entre eles[20]. O mesmo ocorre com todas as regras de produção que começam pelo símbolo inicial S, isto é, procura-se o mapeamento $a \rightarrow e$, $b \rightarrow f$, $c \rightarrow g$ e $d \rightarrow h$ nas seqüencias para caracterizá-las como palíndromo disjunto. Sendo assim, reconhece, por exemplo, a seqüência "abfe" como válida e recusa a seqüência "abba".

5.2.2 Análise dos Resultados

Para a inferência de gramáticas para a linguagem Parênteses, o método foi extremamente rápido, convergindo em menos de cinco gerações. Isto se deve ao fato de ser uma gramática simples e que com a população inicial sendo composta de 50% de regras de produção oriundas dos exemplos positivos, fica fácil convergir com auxílio do operador de aprendizagem incremental. Com a população inicial completamente aleatória, o método leva de 20 a 35 gerações para obter sucesso[21].

Nas gramáticas para palíndromos, o comportamento tanto para palíndromo comum quanto para o disjunto diferenciou-se em menos de 5% nos seus valores médios.

A variação dos valores médios de sensibilidade (*Sens*), especificidade (*Spec*) e taxa de acerto (*Tacc*) em cada geração para as vinte e cinco primeiras gerações estão representados na Figura 36, pois em todas as cinqüenta execuções, a solução foi encontrada antes da 25ª geração. Os valores se referem à média entre as melhores gramáticas obtidas em cada

[20] Isto se deve ao fato da Forma Normal Chomsky adotada não suportar a produção $\alpha \rightarrow \varepsilon$ onde ε é o símbolo nulo.

[21] Em 50 execuções, utilizando a mesma semente aleatória, isto é, mantendo-se as mesmas condições de atuação probabilística dos operadores genéticos.

geração. Durante as primeiras sete gerações, o método proposto procura melhorar tanto a sensibilidade quanto a especificidade. Os valores de especificidade são maiores, pois é mais fácil uma gramática rejeitar exemplos negativos do que aceitar os positivos. Entre a oitava e a décima- primeira geração ocorre uma alternância entre os valores de sensibilidade e especificidade. Este comportamento se deve ao efeito do operador de aprendizagem incremental que, ao fazer uma gramática cobrir um exemplo positivo, provoca a cobertura de um ou mais exemplos negativos.

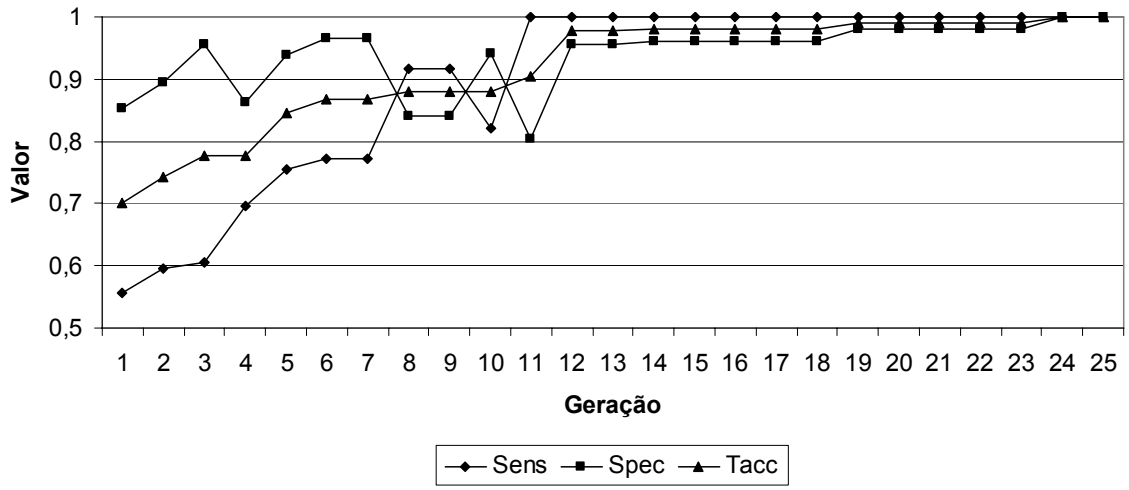

Figura 36: Variação da sensibilidade (*Sens*), especificidade (*Spec*) e taxa de acerto (*Tacc*) da melhor gramática durante a inferência de palíndromos.

Na Figura 37 tem-se a variação do menor, da média e do maior valor de *fitness* encontrados nas cinqüenta execuções durante as vinte e cinco primeiras gerações para o conjunto de palíndromos. O método apresenta uma convergência lenta, o que é desejável em inferência, pois permite escapar de ótimos locais.

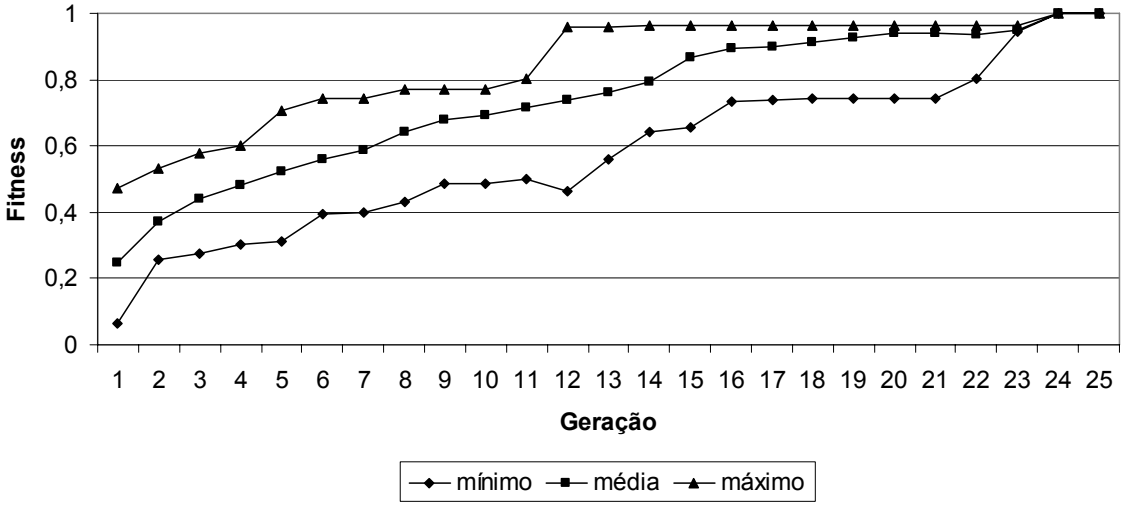

Figura 37: Variação do valor de *fitness* durante a inferência de palíndromos.

Pelo fato da PG usar uma representação de indivíduos de tamanho variável, um problema freqüentemente encontrado é o crescimento descontrolado de partes inúteis (*introns*) no indivíduo que prejudicam a convergência, além de torná-los grandes demais. Este problema recebe o nome de efeito *bloat* (inchaço) e deve ser evitado. Neste trabalho, um *intron* representa uma produção inútil, isto é, inalcançável pelo símbolo inicial S. Graças à atuação do operador de correção (Seção 4.3.3), as produções inúteis são sempre eliminadas.

Infelizmente, nada impede que uma gramática cresça indevidamente com produções alcançáveis pelo símbolo inicial S que pouco ou nada contribuem para a convergência do método. O operador de expansão pode provocar este crescimento. A Figura 38 mostra o crescimento dos indivíduos, em termos de número de produções, nas primeiras quinze gerações em uma das execuções da inferência de palíndromos. Nesta figura são apresentados o tamanho da menor e da maior gramática em cada geração, além do tamanho da melhor gramática (maior *fitness*) durante as cinqüenta execuções.

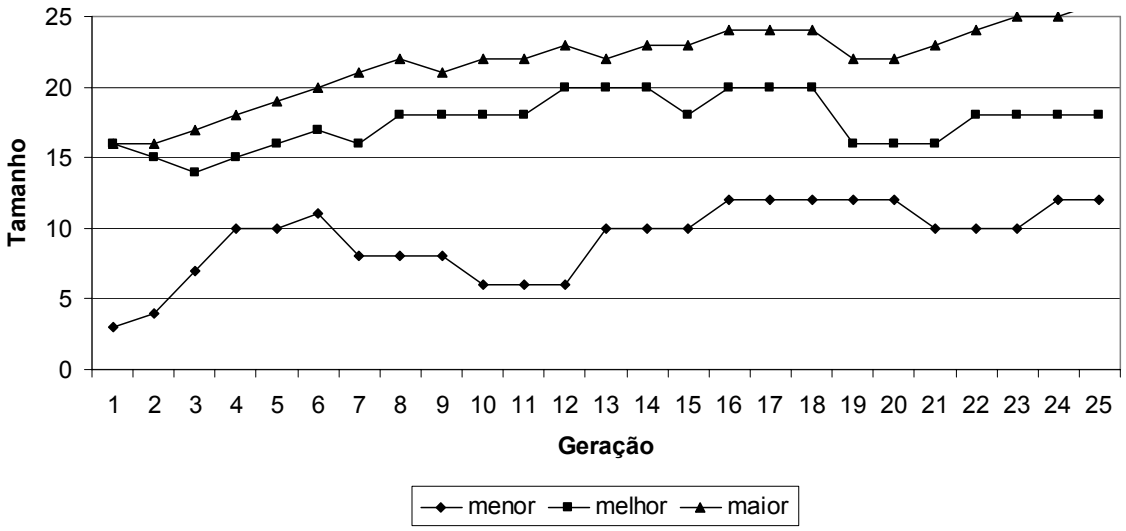

Figura 38: Variação do tamanho das gramáticas durante inferência de palíndromos.

Na figura apresentada, o tamanho das gramáticas cresce rapidamente nas primeiras oito gerações. Ao comparar esta figura com a Figura 36, é possível notar que este crescimento é devido ao operador de aprendizagem incremental, pois nestas oito primeiras gerações o valor da sensibilidade cresce rapidamente. Após a oitava geração, o crescimento é lento, pois o operador de aprendizagem não atua mais, dando lugar ao operador de expansão. Além disto, após cada avaliação, todas as produções que apresentam os escores positivo e negativo iguais a zero são sempre eliminadas. Sendo assim, não houve um crescimento acentuado das gramáticas.

Para entender melhor o comportamento dos operadores de aprendizagem incremental e expansão, a Figura 39 apresenta a variação da taxa de acerto sem o uso dos operadores, com o uso de apenas o operador de aprendizagem incremental e com ambos. As três execuções foram feitas com a mesma população inicial para permitir uma correta comparação.

Sem os operadores, o valor de *fitness* aumenta levemente e não atinge o máximo. Com o uso do operador de aprendizagem incremental, o valor de *fitness* cresce rapidamente, mas se estabiliza em 0,9 após a 17ª geração. Com o uso conjunto do operador de expansão e de aprendizagem incremental, o valor de *fitness* atinge o máximo, pois ele é capaz de produzir espontaneamente as produções que faltavam para a convergência do método.

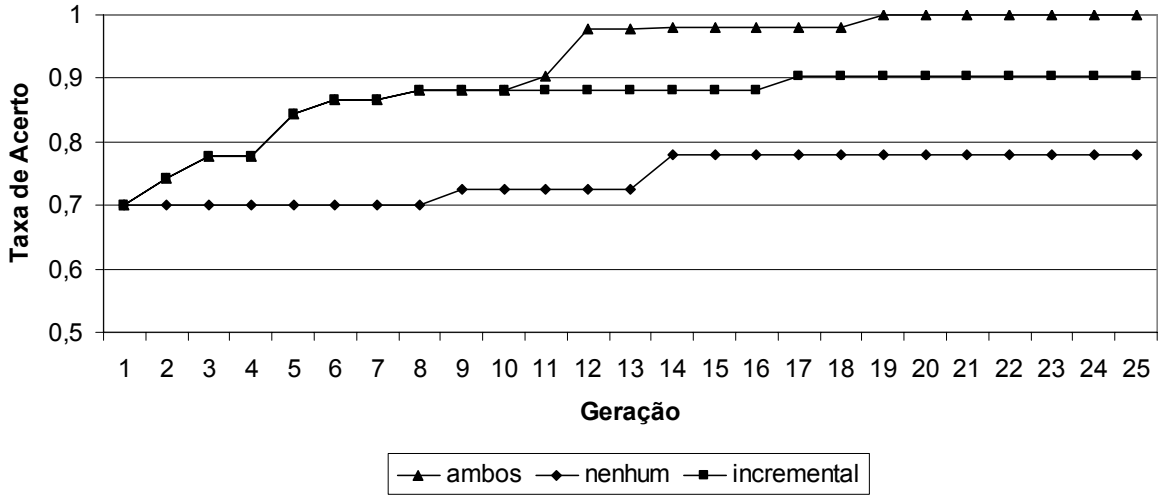

Figura 39: Variação da taxa de acerto em relação ao uso ou não dos operadores propostos.

Um questionamento que se faz ao se usar abordagens evolucionárias se refere aos valores dos parâmetros do modelo, principalmente as probabilidades de cruzamento e mutação. Habitualmente em PG usa-se alta probabilidade de cruzamento (em torno de 90%) e nenhuma de mutação ou eventualmente não mais que 10% (BANZHAF, NORDIN, KELLER *et al*, 1998). Durantes os testes, foi possível perceber que uma alta probabilidade de cruzamento e uma baixa probabilidade de mutação não favoreciam a convergência do método. Sendo assim, fez-se um conjunto de experimentos na inferência de palíndromos com as probabilidades de cruzamento e mutação variando de 0% a 100%. As Figuras 40 e 41 mostram os resultados obtidos. É possível perceber que valores altos de probabilidades são preferíveis, significando que a atuação de ambos os operadores de cruzamento de mutação são importantes para a convergência do método. Com base nos valores obtidos, optou-se pelas probabilidades de 70% de cruzamento e 25% de mutação que representam os menores valores possíveis para a obtenção do *fitness* máximo.

É importante lembrar que a utilidade do operador de cruzamento na abordagem apresentada está na troca de produções inicialmente ruins entre gramáticas. Desta forma, as produções que cobrem mais exemplos negativos do que positivos (valor de c(p) negativo) são substituídas nas gramáticas. Além disto, na nova gramática, estas produções podem vir a colaborar na melhoria da taxa de acerto. De forma semelhante, o mesmo corre com o operador de mutação que, substituindo uma produção ruim por outra completamente nova, traz diversidade ao conjunto de produções.

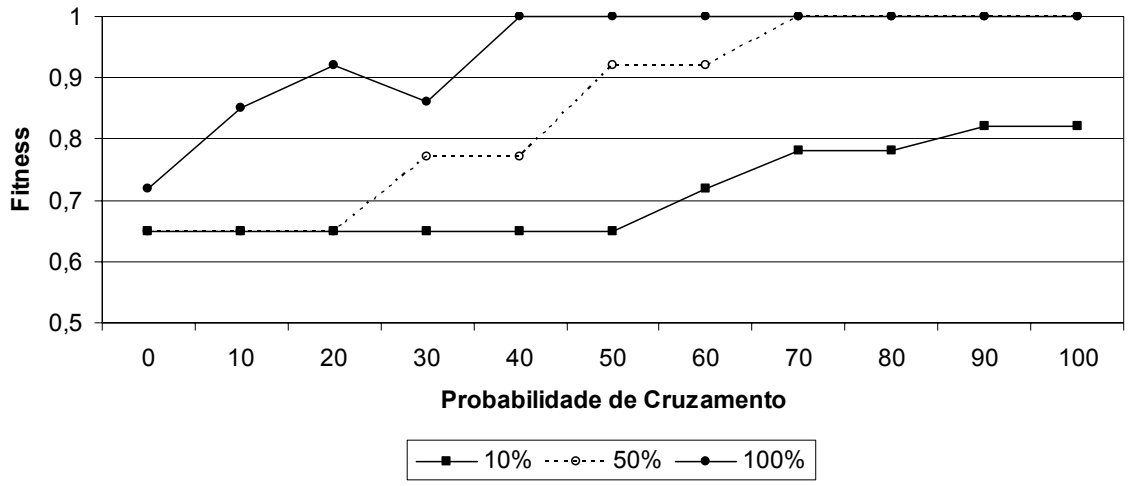

Figura 40: Melhor valor de *fitness* obtido para probabilidades de mutação de 10%, 50% e 100% para valores de cruzamento de 0% a 100%.

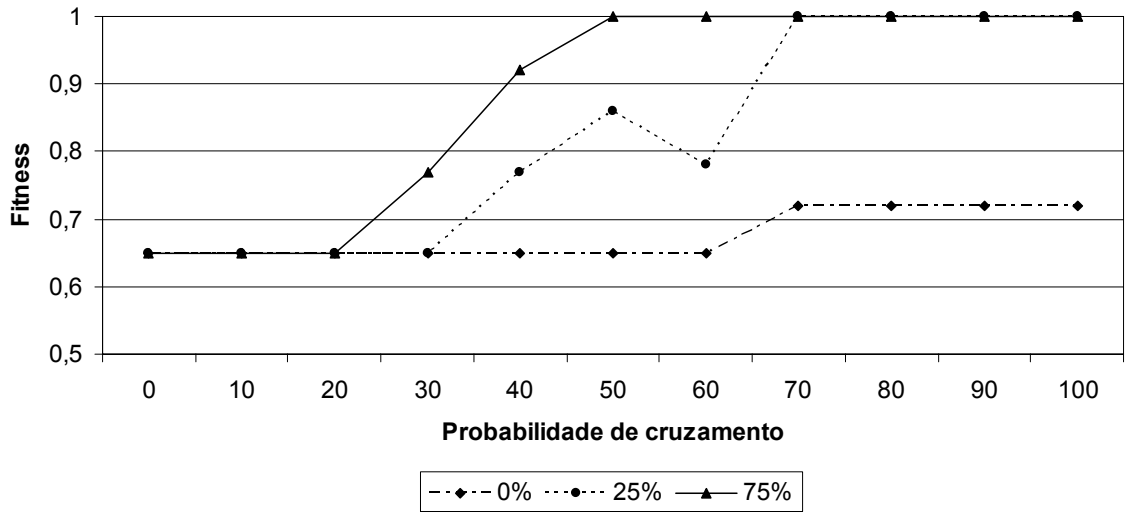

Figura 41: Melhor valor de *fitness* obtido para probabilidades de mutação de 0%, 25% e 75% para valores de cruzamento de 0% a 100%.

5.3 INFERÊNCIA DE GRAMÁTICA PARA SEQUÊNCIAS DE DNA

Tanto o conjunto de parênteses quanto o de palíndromos são exemplos de seqüências que podem, comprovadamente, ser geradas por gramáticas livres de contexto.

Para analisar a capacidade de inferência do método em problemas nos quais não se sabe se os exemplos disponíveis podem ser gerados por uma gramática livre de contexto, dois problemas foram escolhidos: reconhecimento de região promotora e de *splice-junction*.

5.3.1 Conceitos Básicos

O código genético, na forma de unidades conhecidas como genes, está no DNA, no interior das células. Segundo Nelson e Cox (2004), o ácido desoxirribonucléico (DNA) e o ácido ribonucléico (RNA) são substâncias químicas constituintes dos genes. Estas macromoléculas estão envolvidas na transmissão dos caracteres hereditários e na produção de proteínas, que são produzidas constantemente pelas células.

Basicamente, uma molécula de ácido desoxirribonucléico (DNA) consiste de duas fitas entrelaçadas em forma de dupla hélice. Cada fita é composta por uma seqüência de quatro diferentes nucleotídeos (bases): Adenina (A), Guanina (G), Citosina (C) e Timina (T). Cada nucleotídeo de uma fita se liga a outro complementar da segunda, conforme os pares: A-T, T-A, C-G e G-C.

A propriedade mais importante dos genes que compõem um genoma está no fato de que eles codificam proteínas. As proteínas exercem um papel importante no metabolismo, participando praticamente de todas as atividades celulares.

A identificação de genes em seqüências de DNA constitui uma tarefa muito custosa se realizada por meios laboratoriais. A obtenção de um algoritmo que automatize esta tarefa também é inviável, pois o conhecimento acerca da natureza dos genes ainda é incompleto.

No reconhecimento de genes por meios computacionais, duas abordagens são usualmente empregadas: por sinal e por conteúdo (PAVLIDS, FUREY, LIBERTO *et al*, 2001). Estas abordagens diferem nas características das seqüências em que se concentram. A busca por sinal envolve a localização de sítios presentes na molécula de DNA que participam

do processo de expressão gênica[22]. Na busca por conteúdo procuram-se padrões nas seqüências genômicas que indiquem a presença de um gene.

Como exemplos de busca de sinais têm-se: identificação de sítios de início de tradução, identificação de região promotora e identificação de sítios de *splicing*, sendo estes dois últimos apresentados a seguir.

5.3.2 Identificação de Região Promotora

Neste problema, tem-se um conjunto de seqüências de DNA de tamanho fixo com regiões promotoras conhecidas e seqüências sem a sua presença. O objetivo é gerar um classificador capaz de identificar se uma janela de tamanho fixo de uma seqüência de DNA possui ou não um promotor.

Para este experimento, optou-se por utilizar seqüências de genes de *Escherichia Coli*. As seqüências foram obtidas de uma base de dados compilada por Harley e Reynolds (1987) da Universidade de Wisconsin. Neste caso, o conjunto de dados possui 106 exemplos, dos quais 53 são exemplos positivos (contém regiões promotoras) e 53 são negativos (trechos de regiões intragênicas e, portanto, não possuindo as características das regiões promotoras). Cada exemplo possui 57 nucleotídeos que são seqüencias do alfabeto {A,C,T,G}, sendo que aquelas que correspondem a regiões promotoras se referem às posições -50 até a +7 do início de um gene. Isto é, os 50 nucleotídeos que precedem um gene e os 7 primeiros nucleotídeos de um gene. Neste trabalho foram removidos os nucleotídeos das posições +1 a +7, sendo apenas consideradas as posições -50 a -1 (a região que precede efetivamente a informação gênica). Isto foi feito para se possível uma comparação direta com os resultados obtidos por Towell, Shavlik e Noordewier (1990) e por Tavares, Lopes e Lima (2007) em diversos métodos de aprendizado de máquina, conforme mostrado na Tabela 14, onde o índice 1 representa os resultados obtidos pelo primeiro e o índice 2, pelo segundo. O método proposto está identificado como GPGI.

É importante frisar que pelo baixo número de exemplos (106), qualquer novo exemplo positivo coberto causa um aumento de praticamente 1% na taxa de acerto.

[22] O termo Expressão Gênica se refere ao processo em que a informação codificada por um determinado gene é decodificada em uma proteína.

Tabela 14: Comparação entre os diversos métodos de aprendizagem de máquina.

Método	**Taxa de Acerto**
GPGI usando exemplos positivos para criar metade da população inicial	**98%**
KBANN[1]	96%
Perceptron Multicamada[1]	93%
HNB[2]	93%
HMM[2]	92%
NaiveBayes[2]	92%
NaiveBayesSimple[2]	92%
NaiveBayesUpdateable [2]	92%
SMO[2]	92%
Árvore de Decisão com LBR[2]	92%
AODE [2]	91%
RBFNetwork[2]	90%
LogitBoost[2]	90%
Árvore de Decisão com NBTree [2]	90%
O'Neill[1]	89%
Árvore de Decisão com LMT [2]	89%
k-NN[1]	88%

Logistic[2]	88%
MultiBoostAB[2]	88%
MultiClassClassifier[2]	88%
ThresholdSelector[2]	87%
Árvore de Decisão com ADTree[2]	87%
ADABoost[2]	86%
VotedPerceptron[2]	84%
Árvore de Decisão com ID3[1]	82%
IB1[2]	82%
Kstar [2]	82%
GPGI com população inicial aleatória	**82%**
Árvore de Decisão com J48[2]	81%
IBk[2]	81%
PART[2]	81%
DecisionTable[2]	78%
Ridor[2]	78%
JRip [2]	77%
NNge[2]	77%

O resultado obtido usando a população inicial completamente aleatória se equivale à abordagem baseada em Árvores de Decisão, sendo inferior aos outros métodos. A gramática obtida neste caso apresenta apenas 21 produções que são as seguintes:

$P_1 \rightarrow a \mid P_1\ P_1 \mid S\ P2 \mid S\ S$ $\quad$ $P_2 \rightarrow c \mid P_1\ S \mid P_3\ P_1 \mid P_4\ S$

$P_3 \rightarrow g \mid P_3\ P_2 \mid P_3\ P_3 \mid S\ S$ $\quad$ $P_4 \rightarrow t \mid P_1\ P_1 \mid P_2\ S \mid P_4\ P_1$

$S \rightarrow P_4\ S \mid S\ P_1 \mid S\ P_3 \mid S\ P_4 \mid S\ S$

Porém, ao se usar a abordagem de criar metade da população inicial usando os exemplos positivos, conforme apresentado na Seção 4.3.2, o método é capaz de encontrar uma gramática com 98% de acerto (reconhecendo indevidamente dois exemplos negativos) com 78 produções, apresentadas a seguir:

$P_1 \rightarrow a$ $\quad$ $P_2 \rightarrow c$ $\quad$ $P_3 \rightarrow g$ $\quad$ $P_4 \rightarrow t$

$P_5 \rightarrow P_1\ P_1$ $\quad$ $P_6 \rightarrow P_1\ P_2$ $\quad$ $P_7 \rightarrow P_1\ P_3$ $\quad$ $P_8 \rightarrow P_1\ P_4$

$P_9 \rightarrow P_2\ P_1$ $\quad$ $P_{10} \rightarrow P_2\ P_2$ $\quad$ $P_{11} \rightarrow P_2\ P_3$ $\quad$ $P_{12} \rightarrow P_2\ P_4$

$P_{13} \rightarrow P_3\ P_1$ $\quad$ $P_{14} \rightarrow P_3\ P_2$ $\quad$ $P_{15} \rightarrow P_3\ P_3$ $\quad$ $P_{16} \rightarrow P_3\ P_4$

$P_{17} \rightarrow P4\ P_1$ $\quad$ $P_{18} \rightarrow P_4\ P_2$ $\quad$ $P_{19} \rightarrow P_4\ P_3$ $\quad$ $P_{20} \rightarrow P_4\ P_4$

$S \rightarrow P_1\ P_1 \mid P_1\ P_{15} \mid P_1\ P_{19} \mid P_1\ P_{20} \mid P_1\ P_5 \mid P_1\ P_8 \mid P_1\ P_9 \mid P_{12}\ P_3 \mid P_{14}\ P_5 \mid$
$P_{16}\ P_6 \mid P_{17}\ P_1 \mid P_{17}\ P_{12} \mid P_{18}\ P_{18} \mid P_{18}\ P_{19} \mid P_{19}\ P_{12} \mid P_2\ P_{13} \mid P_2\ P_{19} \mid$
$P_2\ P_2 \mid P_{20}\ P_{16} \mid P_{20}\ P_{18} \mid P_3\ P_{17} \mid P_3\ P_2 \mid P_3\ P_6 \mid P_3\ P_8 \mid P_4\ P_{11} \mid$
$P_4\ P_{14} \mid P_4\ P_2 \mid P_4\ P_{20} \mid P_4\ P_3 \mid P_4\ P_4 \mid P_4\ P_6 \mid P_5\ P_{20} \mid P_5\ P_3 \mid P_5\ P_8 \mid$
$P_6\ P_2 \mid P_8\ P_3 \mid P_9\ P_3 \mid P_9\ P_4 \mid S\ P_1 \mid S\ P_{10} \mid S\ P_{11} \mid S\ P_{12} \mid S\ P_{13} \mid$
$S\ P_{14} \mid S\ P_{15} \mid S\ P_{16} \mid S\ P_{17} \mid S\ P_{18} \mid S\ P_{19} \mid S\ P_2 \mid S\ P_{20} \mid S\ P_3 \mid S\ P_4 \mid$
$S\ P_5 \mid S\ P_6 \mid S\ P_7 \mid S\ P_8 \mid S\ P_9$

A variação do valor de *fitness* durante as vinte primeiras gerações está mostrada na Figura 42, com a população inicial criada de forma aleatória e com o método proposto na Seção 4.3.2. É possível perceber que, com a população inicial aleatória, o crescimento do valor de *fitness* é rápido nas três primeiras gerações e se torna lento nas seguintes até estabilizar em 0,82 após a décima-oitava geração. Com o método proposto, há um comportamento dinâmico semelhante, porém as gramáticas começam com uma taxa de acerto maior.

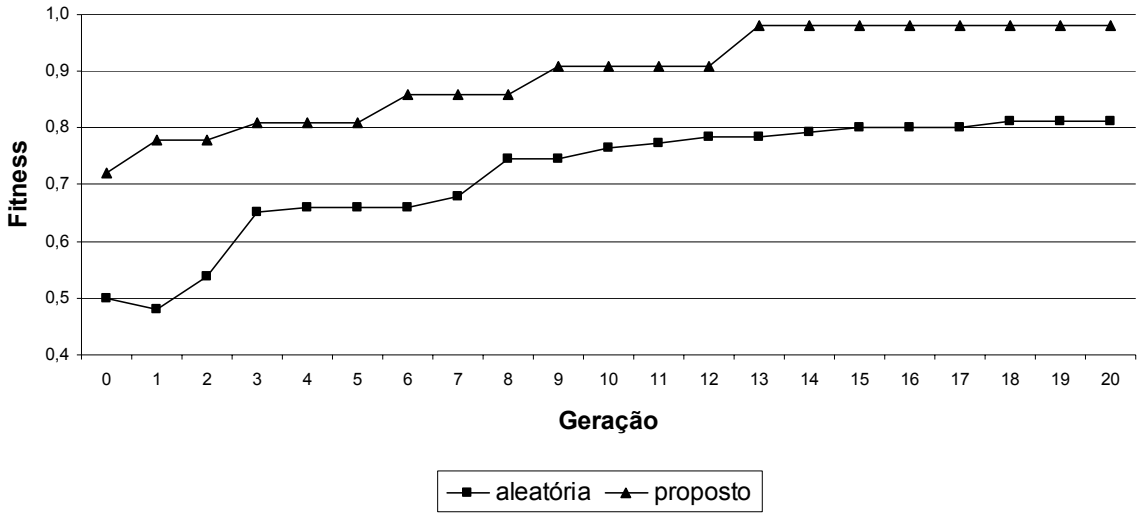

Figura 42: Variação do valor de *fitness* da melhor gramática usando população inicial aleatória e usando o método proposto.

A variação dos valores de sensibilidade (*Sens*), especificidade (*Spec*) e taxa de acerto (*Acc*) para as vinte primeiras gerações estão representados na Figura 43. Os valores se referem a melhor gramática obtida em cada geração em uma das execuções com a população inicial aleatória. Para o método proposto de criação da população inicial, o comportamento é idêntico, exceto pelo fato de começar com valores maiores de sensibilidade e taxa de acerto. Ao analisar as curvas apresentadas, há um crescimento rápido na sensibilidade nas primeiras três gerações devido ao efeito do operador de aprendizagem. Em seguida, ocorre uma estabilidade até a sexta geração. A partir de então, começa uma alternância entre os valores de sensibilidade e especificidade e estabiliza após a décima-sétima geração. O comportamento é semelhante ao apresentado na inferência de palíndromos.

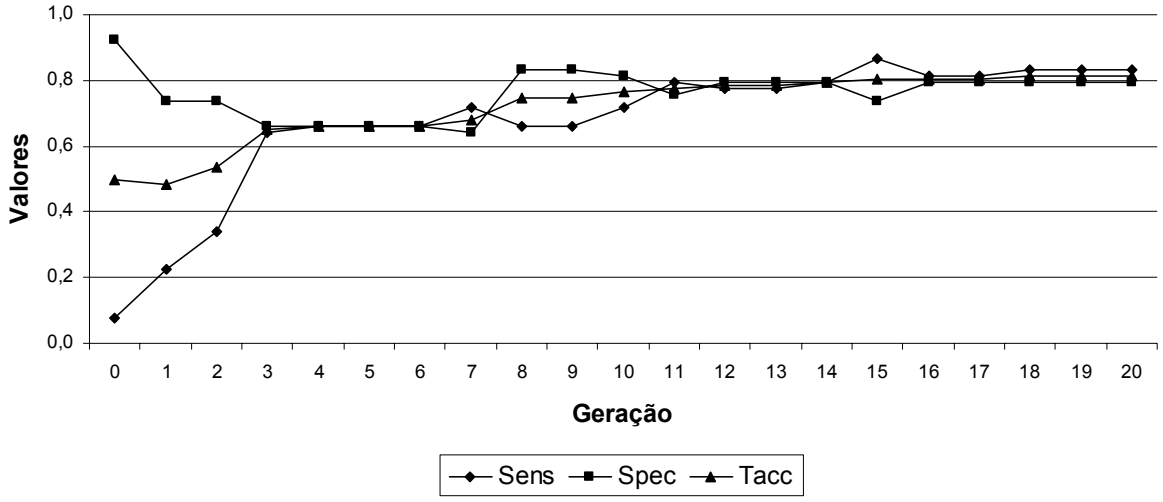

Figura 43: Variação da sensibilidade (*Sens*), especificidade (*Spec*) e taxa de acerto (*Acc*) da melhor gramática obtida nas vinte primeiras gerações.

5.3.3 Reconhecimento de *splice-junction*

Neste problema, um conjunto de seqüências de DNA de tamanho fixo com fronteiras do tipo *intron-exon*, *exon-intron* e sem nenhuma delas é fornecido. O objetivo é obter um classificador capaz de determinar se uma janela de tamanho fixo de uma seqüência de DNA possui uma fronteira *intron-exon*, *exon-intron* ou nenhuma.

Para este experimento, optou-se por utilizar dois conjuntos de seqüências de genes de primatas. O primeiro conjunto foi obtido do Genbank 64.1 por Noordewier, Towell e Shavlik (1992). O segundo foi extraído de conjunto de dados da Universidade da Califórnia usados na validação do NNSPLICE[23], uma rede neural destinada a reconhecer sítios de *splicing*. Tal conjunto de dados foi utilizado em um comparativo recente entre algoritmos de inferência gramatical estocástica (KASHIWABARA, VIEIRA, LIMA *et al*, 2007).

5.3.3.1 Resultados para o primeiro conjunto de seqüências

O conjunto contém 3190 seqüências sendo 767 contendo transições EI, 768 com transições IE e 1655 com nenhuma delas.

O objetivo é classificar uma seqüência em uma das três classes possíveis. Uma abordagem possível para este problema é obter dois classificadores distintos.

[23] http://www.fruitfly.org/seq_tools/splice.html

O primeiro deve reconhecer se existe uma transição EI e outro, se existe uma transição IE. Se nenhum dos dois classificadores reconhecer uma transição, então ela não existe (classe N). Esta abordagem foi usada por Noordewier, Towell e Shavlik (1992) e foi adotada neste trabalho para permitir uma comparação direta. Sendo assim, para o treinamento dos classificadores, foi usado *k-fold stratified cross-validation* com k = 10 usando conjuntos de 1000 exemplos extraídos aleatoriamente dos 3190 exemplos. Os resultados estão mostrados na Tabela 15.

Tabela 15: Taxas de acerto dos classificadores no reconhecimento de *splice-junction*

	Classe		
Método	**N**	**EI**	**IE**
KBANN	95%	93%	92%
Backprop	95%	95%	80%
PEBLS	93%	92%	93%
Perceptron	96%	84%	83%
ID3	92%	90%	87%
COBWEB	89%	85%	90%
Near. Neighbor.	69%	89%	91%
GPGI com população inicial aleatória	58%	53%	52%
GPGI usando exemplos positivos para criar metade da população inicial	73%	72%	62%

Apesar do ganho que se obtém usando os exemplos positivos para criar metade da população inicial (a média de acertos sobe de 54% para 69%), ainda assim permanece abaixo dos outros métodos reportados. Porém, diferentemente de GPGI, os outros métodos são aproximados, isto é, não buscam uma representação exata de classificação. Na Seção 5.4 propõe-se a adoção de gramáticas *fuzzy* como mecanismo de ajuste do modelo.

5.3.3.2 Resultados para o segundo conjunto de seqüências

Neste caso, são fornecidos dois conjuntos: um para treinamento e outro para teste. O conjunto de treinamento consiste em 1116 seqüências com transição EI e 1110 com transição IE. Além disto, são fornecidas 4139 seqüências de falsas transições EI e 4658 seqüências de falsas transições IE. O conjunto de teste contem 208 seqüências com transição EI, 208 seqüências com transição IE, 782 seqüências de falsas transições EI e 881 seqüências de falsas transições de IE.

Para o treinamento dos classificadores, foi usado *k-fold stratified cross-validation* com $k = 10$. Os resultados obtidos pela melhor gramática encontrada estão mostrados nas Tabelas 16 (para transições EI) e 17 (para transições IE), juntamente com os resultados reportados por Kashiwabara, Vieira, Lima e Durham (2007).

Tabela 16: Taxas de erro no reconhecimento de EI[24]

Método	**Erro**		
	% Total	**% Falso-Positivos**	**% Falso-Negativos**
Alergia	10,9	4,7	34,1
Amnésia	22,3	22	23,6
Labfa	5,7	3,8	12,5
NNSPLICE	5,2	6,8	5,5
RPNI	23,5	23,9	22,4
GPGI com população inicial aleatória	51	46	45
GPGI usando exemplos positivos para criar metade da população inicial	48	44	45

[24] Seqüências com transição *exon-intron*.

Tabela 17: Taxas de erro no reconhecimento de IE[25]

Método	Erro		
	% Total	% Falso-Positivos	% Falso-Negativos
Alergia	29,8	1,8	76,4
Amnésia	37,4	34	51,4
Labfa	5,3	4,68	8,2
NNSPLICE	7,9	3	26,2
RPNI	31,5	28,1	45,5
GPGI com população inicial aleatória	58	36	65
GPGI usando exemplos positivos para criar metade da população inicial	56	36	62

Neste caso, não houve ganho significativo usando os exemplos positivos para criar metade da população inicial. Os resultados obtidos pelo método proposto foram bem inferiores aos outros métodos de inferência reportados. Isto se deve ao fato dos outros métodos serem aproximados e não exatos, permitindo um ajuste no modelo para uma melhor adequação aos dados de treinamento.

5.4 INFERÊNCIA DE GRAMÁTICAS FUZZY

O objetivo do uso do refinamento *fuzzy* é conseguir ajustar a gramática inferida pela abordagem em PG quando não se sabe se existe uma gramática livre de contexto capaz de diferenciar os exemplos positivos e negativos ou se o método de inferência for incapaz de obter uma ótima taxa de acerto.

Neste trabalho, propõe-se uma variação no método de inferência gramatical baseado em PG apresentado para suportar gramática livre de contexto *fuzzy* (Seção 3.6). Tal variação

[25] Seqüências com transição *intron-exon*.

se baseia na estimação dos valores de μ(p) para cada produção da gramática e na escolha do limiar *c* para reconhecimento.

Há duas maneiras possíveis de se aplicar a variação *fuzzy* as gramáticas em PG: (a) aplicar ao término do processo evolutivo, quando não é possível mais melhorar a taxa de acerto da melhor gramática e (b) aplicar desde a primeira geração a cada melhor gramática encontrada.

Na Tabela 18 estão os valores de taxa de acerto obtidos sem o uso de *fuzzy* (as duas primeiras linhas da tabela) e com o uso de *fuzzy* (as duas últimas). Neste caso, a variação *fuzzy* apresenta melhor resultado se atuar na primeira geração, onde a melhor gramática tem uma taxa de acerto levemente superior a 50%. Quando a gramática tem uma alta taxa de acerto (98%), a variação tende a piorar o resultado.

Tabela 18: Comparação entre o uso ou não de *fuzzy* para o reconhecimento de região promotora.

Método	**Taxa de Acerto**
GPGI com população inicial aleatória	82%
GPGI usando exemplos positivos para criar metade da população inicial	98%
GPGI com *fuzzy* aplicado na primeira geração	84%
GPGI com *fuzzy* aplicado ao final do processo evolutivo	72%

No caso de reconhecimento de *splice-junction*, a capacidade de reconhecimento da melhor gramática obtida na primeira geração mostrou-se muito superior a abordagem sem *fuzzy*. No caso do primeiro conjunto de dados, os resultados se aproximam do melhor método reportado (KBANN). Os resultados estão na Tabela 19.

Tabela 19: Comparação entre o uso ou não de *fuzzy* para o reconhecimento de *splice-junction*

	Classe		
Método	**N**	**EI**	**IE**
GPGI com população inicial aleatória	58%	53%	52%
GPGI usando exemplos positivos para criar metade da população inicial	73%	72%	62%
GPGI com *fuzzy* aplicado na primeira geração	90%	86%	88%

Para o segundo conjunto de dados, os resultados estão nas Tabelas 20 (para transições EI) e 21 (para transições IE). Apesar de não serem melhores que as outras abordagens, os resultados são muito melhores aos obtidos quando não se usa a extensão *fuzzy*.

Tabela 20: Taxas de erro no reconhecimento de EI[26]

Método	**Erro**		
	% Total	**% Falso-Positivos**	**% Falso-Negativos**
Alergia	10,9	4,7	34,1
Amnésia	22,3	22	23,6
Labfa	5,7	3,8	12,5
NNSPLICE	5,2	6,8	5,5
RPNI	23,5	23,9	22,4
GPGI com *fuzzy*	25	22	33

[26] Seqüências com transição *exon-intron*.

Tabela 21: Taxas de erro no reconhecimento de IE[27]

Método	**Erro**		
	% Total	**% Falso-Positivos**	**% Falso-Negativos**
Alergia	29,8	1,8	76,4
Amnésia	37,4	34	51,4
Labfa	5,3	4,68	8,2
NNSPLICE	7,9	3	26,2
RPNI	31,5	28,1	45,5
GPGI com *fuzzy*	33	29	58

Portanto, o uso da variação *fuzzy* proposta se torna útil quando o processo de inferência proposto de PG não consegue obter uma gramática com boa taxa de acerto. Uma possível melhoria do método seria incluir uma forma de evoluir os valores das funções *g* e *h* através de algum método heurístico de ajuste de valores contínuos representados em forma de vetor, tal como evolução diferencial (PRICE, STORN e LAMPINEN, 2005).

[27] Seqüências com transição *intron-exon*.

CAPÍTULO 6

DISCUSSÃO DOS RESULTADOS E CONCLUSÕES

6.1 DISCUSSÃO DOS RESULTADOS

Nos experimentos realizados com gramáticas previamente conhecidas, o método proposto apresentou resultados excelentes na inferência de gramáticas livres de contexto para parênteses e palíndromos de quatro letras, inclusive para o palíndromo disjunto. No caso dos parênteses, nas 50 execuções efetuadas, todas inferiram a gramática correta em um número pequeno de gerações (entre um e quatro), apesar do tamanho da população ser de apenas 100 gramáticas[28]. No caso dos palíndromos, todas as 50 execuções obtiveram sucesso em, no máximo, 25 gerações. Nos palíndromos disjuntos, em 84% das execuções houve sucesso. Nas restantes, a taxa de acerto foi próxima de 96%.

Estes resultados são superiores a abordagens conhecidas se aplicadas ao mesmo conjunto de treinamento e teste usado, tais como SCFG e HMM e, inclusive, superior a uma abordagem recente baseada em *String Kernels* (CLARK; FLORENCIO e WATKINS, 2006).

O modelo proposto também foi aplicado a duas tarefas de reconhecimento de padrões em seqüência de DNA: identificação de região promotora e de *splice-junction*. Para o primeiro, o método apresentou resultado superior a diversos métodos de aprendizagem de máquina anteriormente utilizados, obtendo uma taxa de acerto de 98%. No reconhecimento de *splice-junction*, buscou-se classificar as seqüências em três classes[29]: N (nenhuma transição), EI (transição *exon-intron*), IE (transição *intron-exon*) e N (nenhuma transição). Para tal, usaram-se dois conjuntos de dados, um obtido do Genbank 64.1 e o segundo, de um conjunto mais recente de dados da Universidade da Califórnia usados na validação do NNSPLICE.

Para o primeiro conjunto, as taxas de acerto obtidas foram 73%, 72% e 62% respectivamente, contra 95%, 93% e 92% obtidas, respectivamente, pelo método KBANN. Para o segundo, os resultados foram inferiores aos obtidos por métodos de inferência gramatical estocástica. O problema reside no fato das seqüências apresentarem muita similaridade e possivelmente ruído, tornando inadequada inferência de um modelo exato, tal como uma gramática livre de contexto.

---

[28] Normalmente em PG trabalha-se com populações de, no mínimo, 500 indivíduos.

[29] Mais detalhes, consulte a Seção 5.3.3

O modelo, então, foi estendido para suportar gramáticas *fuzzy*, mais adequadas para a classificação quando não há a certeza de que os exemplos possam ser reconhecidos por uma gramática livre de contexto, quando há muita semelhança entre os exemplos ou presença de ruídos. A extensão *fuzzy* usou uma abordagem baseada em gramáticas fracionárias, mais adequadas para reconhecimento de padrões em seqüências. Ao se aplicar no reconhecimento de *splice-junction*, as taxas de acerto obtidas foram maiores. Para o primeiro conjunto, obteve-se 90% para a classe N, 86% para a classe EI e 88% para a classe IE, muito próximas às obtidas pelo KBANN. Para o segundo, apesar de melhores, os resultados foram inferiores aos obtidos pelos outros métodos baseados em gramáticas regulares estocásticas. Isto significa que a abordagem baseada em gramáticas livres de contexto não é adequada para reconhecimento de sítios de *splicing*.

6.2 CONCLUSÕES

A inferência gramatical é uma poderosa ferramenta de reconhecimento de padrões, pois permite perceber como as seqüências podem ser formadas a partir de um conjunto de símbolos. Isto é feito com o uso de regras de produção que estabelecem a forma como as seqüências podem ser formadas.

A dificuldade surge na determinação de quais regras de produção são necessárias para compor a gramática desejada.

Neste trabalho, buscou-se um método evolucionário baseado em PG capaz de gerar e evoluir as regras de produção da gramática. Mas a simples aplicação de PG não surtiu os resultados esperados. Modificações na forma da criação da população inicial, juntamente com modificações no funcionamento dos operadores genéticos e a criação de dois novos operadores foram propostas.

Os resultados obtidos e apresentados no Capítulo 5 demonstram que:

a) A população inicial não deve ser gerada de forma totalmente aleatória;
b) Os operadores de cruzamento e mutação devem afetar preferencialmente as regras de produção menos úteis;
c) Há a necessidade de um operador que seja capaz de gerar as produções que estão faltando para que a gramática cubra os exemplos positivos;
d) A diversidade populacional deve ser mantida durante a evolução.

A solução encontrada para o item (a) está descrita na Seção 4.3, onde se propôs um método para construir um conjunto inicial de produções úteis com base nos exemplos de treinamento. Com relação ao item (b), na Seção 4.5 propôs-se o uso de uma função de crédito para cada produção de forma a direcionar os operadores de cruzamento e mutação. Dois novos operadores genéticos foram propostos nas Seções 4.5.3 e 4.5.4, denominados Aprendizagem Incremental e Expansão respectivamente, para atender as necessidades dos itens (c) e (d).

Portanto, conclui-se que o modelo proposto mostrou-se adequado para inferência de gramáticas livres de contexto conhecidas, tendo obtido resultados excelentes. Além disto, a sua aplicabilidade para problemas de bioinformática foi demonstrada em dois problemas de identificação de padrões em seqüências de DNA. A extensão *fuzzy* proposta também se mostrou adequada quando o modelo de inferência não consegue uma boa taxa de acerto. Resultados no reconhecimento de *splice-junction* demonstram a sua viabilidade.

6.3 TRABALHOS FUTUROS

Para continuidade desta pesquisa, sugerem-se as seguintes tarefas:

- No caso de inferência de gramáticas *fuzzy*, buscar uma forma de evoluir o vetor de valores de pertinência das produções através de algum método heurístico adequado para valores contínuos, tal como Evolução Diferencial (PRICE, STORN e LAMPINEN, 2005).
- Apesar dos bons resultados, a estimação do valor do limiar para as gramáticas *fuzzy* baseada na média dos valores de pertinência pode não ser o valor mais adequado para cada problema tratado.
- Encontrar um modelo híbrido que permita evoluir conjuntamente a estrutura das gramáticas juntamente com os valores de pertinência de cada produção. Neste caso, a heurística para atuação dos operadores deve ser feita de tal forma a maximizar a pertinência da gramática para exemplos positivos e minimizar para os negativos.
- Adequar o modelo para inferência de gramáticas sensíveis a contexto. Para tanto, modificações na representação das gramáticas devem ser feitas juntamente com a forma de avaliação de *fitness*.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. **Compilers: Principles, Techniques & Tools**. Boston: Pearson Addison Wesley, 2007. 1009 p.

ALMEIDA, P. E. M.; EVUSUKOFF, A. G. Sistemas Fuzzy. In: REZENDE, S. O. (Coord) **Sistemas Inteligentes: Fundamentos e Aplicações**, 1ª. Ed. São Paulo: Manole, 2003. p. 89-114.

ASHLOCK, D. **Evolutionary Computation for Modelling and Optimization**. Ontario: Springer, 2006. 571 p.

ASVELD, P. R. J. Fuzzy context-free languages - part 1: generalized fuzzy context-free grammars. **Theoretical Computer Science**, v. 347, n. 1, p. 167-190. Elsevier, 2005.

ASVELD, P. R. J. Generating all permutations by context-free grammars in Chomsky Normal Form. **Theoretical Computer Science**, v. 354, n. 1, p. 118-130. Elsevier, 2006.

BACK, T.; FOGEL, D. B.; MICHALEWICZ, Z. (Editors) **Evolutionary Computation 1: Basic Algorithms and Operators**. Philadelphia: IOP Publishing Ltd, 2000. 339 p.

BALDI, P.; BRUNAK, S. **Bioinformatics: The Machine Learning Approach**. London: MIT Press, 2001. 452 p.

BANZHAF, W.; NORDIN, P.; KELLER, R. E.; FRANCONE, F. D. **Genetic Programming: An Introduction**. San Francisco: Morgan Kaufmann, 1998. 470 p.

BIANCHI, D. Learning grammatical rules from examples using a credit assignment algorithm. In: **Proceedings of the 1st Online Workshop on Soft Computing** (WSC1), p. 113-118. Nagoya University, 1996.

BOHM, W.; GEYER-SCHULZ, A. Exact unifrom initialization for genetic programming. In: **Foundations of Genetic Algorithms IV** (FOGA IV), p. 379-407. Morgan Kaufmann, 1996.

BOJARCZUK, C. C.; LOPES, H. S.; FREITAS, A. A. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. **Artificial Intelligence in Medicine**, v. 30, n. 1, p. 27-48. Elsevier, 2004.

BRAMEIER, M.; HAAN, J.; KRINGS, A.; MACCALLUM, R. M. Automatic discovery of cross-family sequence features associated with protein function. **BMC Bioinformatics**, v. 7, n. 16. BioMed Central Ltd, 2006.

BRAZMA, A.; JONASSEN, I.; VILO, J.; UKHONEN, E. P. Pattern discovery in biosequences. In: Proceedings of 4th Colloquium on Gramatical Inference (ICGI-98). **Lecture Notes in Computer Science**, v. 1433, p. 257-270. Springer-Verlag, 1998.

BRICKMANN, J.; KEIL, M.; EXNER, T. E. Fuzzy logic strategies for the treatment of the molecular recognition problem. **Reviews in Modern Quantum Chemistry**, p. 1735-1766. World Scientific Publications, 2002.

BROWN, P. F.; MERCER, R. L.; PIETRA, V. J. D.; LAI, J. C. Class-based *n*-gram models of natural language. **Computational Linguistics**, v.18, n. 4, p. 467-479. MIT Press, 1992.

CAI, L., MALMBERG, R. L.; WU, Y. Stochastic modeling of RNA pseudoknotted structures: a grammatical approach. **Bioinformatics**, v. 19, p. 166-173. Oxford University Press, 2003.

CHARIKAR, M.; LEHMAN, E.; LIU, D.; PANIGRAHY, R.; PRABHAKARAN, M.; SAHAI, A.; SHELAT, A. The smallest grammar problem. **IEEE Transactions on Information Theory**, v. 51, n. 7, p. 2554-2576. IEEE Press, 2005.

CHELLAPILLA, K. Evolving computer programs without sub-tree crossover. **IEEE Transactions on Evolutionary Computation**, v. 1, n. 3, p. 209-216. IEEE Press, 1997.

CHOMSKY, N. On certain formal properties of grammars. **Information and Control**, v. 2, p. 137-167, 1959.

CHOMSKY, N. **Syntatic Structures**. 2nd Ed. Berlin: Mouton de Gruyter, 2002. 117 p.

CLARK, A.; FLORENCIO, C. C.; WATKINS, C. Languages as hyperplanes: grammatical inference with string kernels. In: Proceedings of the 17th European Conference on Machine Learning (ECML). **Lecture Notes in Artificial Intelligence**, v. 4212, p. 90-101. Springer-Verlag, 2006.

COHEN, J. Bioinformatics – an introduction for computers scientists. **ACM Computing Surveys**, v. 36, n. 2, p. 122-158. ACM Inc., 2004.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 2nd Ed. Cambridge: MIT Press, 2001. 1216 p.

DAIDA, J. M. Challenges with verification, repeatability and meaningful comparisons in genetic programming. In: **Proceedings of the 4th Annual Conference in Genetic Programming** (GECCO 99), p. 1069-1076. Morgan Kaufmann, 1999.

DARWIN, C. **On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life**. London: John Murray, 1859.

DE PALMA, G. F.; YAU, S. S. Fractionally fuzzy grammars with application to pattern recognition. **Fuzzy Sets and their Application to Cognitive and Decision Processes**, p. 329-351. New York Academic Press, 1975.

DUNAY, B. D. Context free language induction with genetic programming. In: **Proceedings of the 6th International Conference on Tools with Artificial Intelligence** (ICTAI '94), p. 828-831. IEEE Press, 1994.

DUPONT, P. Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Proceedings of the 2nd International Colloquium in Grammatical Inference (ICGI-94). **Lecture Notes in Computer Science**, v. 862, p. 236-245. Springer-Verlag, 1994

DURBIN, R.; EDDY, S. R.; KROGH, A. L.; MITCHISON, G. **Biological Sequence Analysis - Probabilistic Models of Proteins and Nucleic Acids.** New York: Cambridge University Press, 1998. 356 p.

FOGEL, D. B. **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**. 3rd Ed. New York: Wiley-IEEE Press, 2005. 296 p.

GANAPATHIRAJU, M.; BALAKRISHNAN, N.; REDDY, R.; KLEIN-SEETHARAMAN, J. Computational biology and language. **Lecture Notes in Computer Science**, v. 3345, p. 25-47. Springer-Verlag, 2005.

GARCIA-GOMEZ, J. M.; BENEDI, J. M. Corpus based learning of stochastic context-free grammar combined with Hidden Markov models for tRNA modelling. In: **Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society** (EMBS'04), p. 2785-2788. IEEE Press, 2004.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**, Reading: Addison-Wesley, 1989. 412 p.

GRUNALD, P. D. **The Minimum Description Length Principle**. Cambridge: MIT Press, 2007. 703 p.

HAYKIN, S. **Neural Networks - A Comprehensive Foundation**. 2nd Ed. New York: Prentice-Hall, 1998. 842 p.

HIGUERA, C. A. Bibliographical study of grammatical inference. **Pattern Recognition**, v. 38, n. 9, p. 1332-1348. Elsevier, 2005.

HOAI, N. X. **A Flexible Representation for Genetic Programming: Lessons from Natural Language Processing**. 2004. 277 p. PhD Thesis - School of Information Technology and Electrical Engineering. University of New South Wales, Australian Defense Academy, 2004.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor: Michigan Press, 1975.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to Automata Theory, Languages, and Computation**. 2nd. Ed. Boston: Addison-Wesley, 2001. 521 p.

HU, Y. Biopattern discovery by genetic programming. In: **Proceedings of the 3rd Annual Conference in Genetic Programming**, p. 152-157. Morgan Kaufmann, 1998.

HUANG, X.; ACERO, A.; HON, H. **Spoken Language Processing: A Guide to Theory, Algorithm and System Development**. New York: Prentice Hall, 2001. 1008 p.

HULO, N.; BAIROCH, A.; BULLIARD, V.; CERUTTI, L.; DE CASTRO, E.; LANGENDIJK-GENEVAUX, P. S.; PAGNI, M.; SIGRIST, C. J. A. The Prosite Database. **Nucleic Acids Research**, v. 34, p. D227-D230, 2006.

JIANG, T.; LI, M.; RAVIKUMAR, B.; REGAN, K. W. Formal grammars and languages. In: ATALLAH, M. J. (Editor) **Handbook on Algorithms and Theory of Computation**. Boca Raton: CRC Press, 1998. 1312 p.

JOHNSON, C. M.; FARREL, J. Evolutionary induction of grammar systems for multi-agent cooperation. In: Proceedings of the 7th European Conference in Genetic Programming. **Lecture Notes in Computer Science**, v. 3003, p. 101-112. Springer-Verlag, 2004.

KAMMEYER, T. E.; BELEW, R. K. Stochastic context-free grammar induction with a genetic algorithm using local search. In: **Foundations of Genetic Algorithms IV**, p. 409-436. Morgan Kauffman, 1996.

KASHIWABARA, A. Y.; VIEIRA, D.; LIMA, A. M.; DURHAM, A. M. Splice site prediction using stochastic regular grammars. **Genetics and Molecular Research**, v. 6, p. 105-115. Fundação de Pesquisas Científicas de Ribeirão Preto (FUNPEC-RP), 2007.

KASSAMI, T. An efficient recognition and syntax analysis algorithm for context-free languages. **Technical Report AFCRL-65-758**. Air Force Cambridge Research Laboratory, 1963.

KELLER, B.; LUTZ, R. Evolutionary induction of stochastic context free grammars. **Pattern Recognition**, v. 38, p. 1393-1406. Elsevier, 2005.

KNUDSEN, B.; HEIN, J. Pfold: RNA secondary structure prediction using stochastic context-free grammars. **Nucleic Acids Research**, v. 31, n. 13, p. 3423-3428. Oxford University Press, 2003.

KNUTH, D. E. **The Art of Computer Programming**. Massachusetts: Addison-Wesley, 1998. v. 3, 780 p.

KORKMAZ, E. E.; UCOLUK, G. Genetic programming for grammar induction. In: **Proceedings of 2001 Genetic and Evolutionary Computation Conference**. Late Breaking Papers, p. 245-251. Morgan Kaufmann, 2001.

KOZA, J. R. **Genetic Programming: on the programming of computers by means of natural selection**. Cambridge: MIT Press, 1992. 819 p.

KOZA, J. R. **Genetic Programming II: Automatic Discovery of Reusable Programs**. Cambridge: MIT Press, 1994. 746 p.

KOZA, J. R.; BENNETT III, F. H.; ANDRE, A.; KEANE, M. A.; **Genetic Programming III: Darwinian Invention and Problem Solving**. San Francisco: Morgan Kaufmann, 1999. 1154 p.

KOZA, J. R.; KEANE, M. A.; STREETER, M. J.; MYDLOWEC, W.; YU, J.; LANZA, G. **Genetic Programming IV: Routine Human-Competitive Machine Intelligence**. New York: Springer-Verlag, 2003. 624 p.

KUBICEK, V.; ZENDULKA, J. Construction of a fuzzy grammar from a set of sentences. In: **Proceedings of the 5th Joint Conference on Knowledge-Based Software Engineering**, p. 108-115. IOS Press, 2002.

LANKHORST, M. M. A genetic algorithm for the induction of context-free grammars. In: **Proceedings of Computational Linguistics in The Netherlands**, p. 87-100, 1994.

NELSON, D. L.; COX, M. M. **Lehninger Principles of Biochemistry**, 4th Ed. New York: W. H. Freeman: 2004. 1100 p.

LOBO, F. G.; LIMA, C. F.; MICHALEWICZ, Z. (Editors) Parameter setting in evolutionary algorithms. In: **Studies in Computational Intelligence**, v. 54, p. 185-204. Springer, 2007.

LORENA, A. C.; BATISTA, G. E. A. P. A.; CARVALHO, A. C. P. L. F.; MONARD, M. C. Splice junction recognition using machine learning techniques. In: **I Workshop Brasileiro de Bioinformática**, p. 32-39. Sociedade Brasileira de Computação (SBC), 2002.

LUCAS, S. Structuring chromosomes for context-free grammar evolution. In: **Proceedings of the 1st IEEE International Conference on Evolutionary Computation**, p. 130-135. IEEE Press, 1994.

LUKE, S., HAMAHASHI, S.; KITANO, H. "Genetic" programming. In: **Proceedings of the Genetic and Evolutionary Computation Conference** (GECCO 99), p. 1098-1105. Morgan Kaufmann, 1999.

LUKE, S. Two fast tree-creation algorithms for genetic programming. **IEEE Transactions in Evolutionary Computation**, v. 4, n. 3, p. 274-283. IEEE Press, 2000.

LUSCOMBE, N. M.; GREENBAUM, D.; GERSTEIN, M. What is bioinformatics? a proposed definition and overview of the field. **Methods in Medical Informatics**, v. 4, p. 346-358. Schattauer, 2001.

MARUO, M. H.; LOPES, H. S.; DELGADO, M. R. Self-adapting evolutionary parameters: encoding aspects for combinatorial optimization problems. In: Proceedings of the 5th European Conference in Evolutionary Computation in Combinatorial Optimization (EvoCOP). **Lecture Notes in Computer Science**, v. 3448, p. 154-165. Springer-Verlag, 2005.

MENEZES, P. B. **Linguagens Formais e Autômatos**. 4ª Ed. Porto Alegre: Sagra Luzzato, 2001. 165 p.

MENON, A. (Editor) **Frontiers of Evolutionary Computation**. New York: Kluwer Academic Publishers, 2004. 271 p.

MERNIK, M.; GERLIC, G.; ZUMER, V.; BRYANT, B. Can a parser be generated from examples? In: **Proceedings of the 2003 ACM Symposium on Applied Computing**, p. 1063-1067. ACM Inc., 2003.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge: MIT Press, 1996. 226 p.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. In: REZENDE, S. O. (Coord) **Sistemas Inteligentes: Fundamentos e Aplicações**, 1ª. Ed. São Paulo: Manole, 2003. p. 89-114.

MONTANA, D. J. Strongly typed genetic programming. **Evolutionary Computation**, v. 3, n. 2, p. 199-230. MIT Press, 1995.

MORDESON, J. N.; MALIK, D. S. **Fuzzy Automata and Languages: Theory and Applications**. Boca Raton: Chapman & Hall/CRC, 2002. 576 p.

NAKAMURA, K.; MATSUMOTO, M. Incremental learning of context free grammars. In: Proceedings of the 6th International Colloquium on Gramatical Inference (ICGI 2002). **Lecture Notes in Computer Science**, v. 2484, p. 749-753. Springer-Verlag, 2002.

NAKAMURA, K. Incremental learning of context free grammars by extended inductive CYK algorithm. In: Proceedings of the 7th International Colloquium on Grammatical Inference (ICGI 2004). **Lecture Notes in Computer Science**, v. 3264, p. 281-282. Springer-Verlag, 2004.

NAKAMURA, K. Incremental learning of context free grammars by bridging rule generation and search for semi-optimum rule sets. In: Proceedings of the 8th International Colloquium on Grammatical Inference (ICGI 2006). **Lecture Notes in Computer Science**, v. 4201, p. 72-83. Springer-Verlag, 2006.

NAUR, P. Revised report on the algorithmic language ALGOL 60. **Communications of ACM**, v. 6, n. 1, p. 1–17. ACM Inc., 1963.

NERBONNE, J.; BELZ, A.; CANCEDDA, N.; DEJEAN, H.; HAMMERTON, J.; KOELING, R.; KONSTANTOPOULOS, S.; OSBORNE, M.; THOLLARD, F.; SANG, E. T. K. Learning computational grammars. In: **Proceedings of 5th Workshop on Computational Language Learning**, p. 97-104. Morgan Kaufmann, 2001.

NEVADO, F.; SANCHEZ, J. A.; BENEDI, J. M. Combination of estimation algorithms and grammatical inference techniques to learn stochastic context-free grammars. In: Proceedings of the 5th International Colloquium on Grammatical Inference. **Lecture Notes in Computer Science**, v. 1891, p. 196-206. Springer-Verlag, 2000.

NEVILL-MANNING, C. G.; WITTEN, I. H. Compression and explanation using hierarchical grammars. **The Computer Journal**, v. 40, p. 103-116. Oxford Journals, 1997.

NOORDEWIER, M. O.; TOWELL, G. G.; SHAVLIK, J. W. Training knowledge-based neural networks to recognize genes in DNA sequences. In: **Advances in Neural Information Processing Systems**, v. 3, p. 530–536. Morgan Kaufmann, 1991.

O'REILLY, U.; YU, T.; RIOLO, R.; WORZEL, B. (Editors) **Genetic Programming Theory and Practice II**. Boston: Springer, 2005. 320 p.

PAVLIDS, P.; FUREY, T. S.; LIBERTO, M.; HAUSSLER, D.; GRUNDY, W. N. Promoter region-based classification of genes. In: **Proceedings of the Pacific Symposium on Biocomputing**, p. 151-163, 2001.

PEDERSEN, C. N. S. **Algorithms in Computational Biology**. 2000. 210 p. PhD Thesis - Department of Computer Science, University of Aarhuns, Denmark. BRICS Dissertation Series, 2000.

PEDRYCZ, W.; GOMIDE, F. **An Introduction to Fuzzy Sets**. Cambridge: MIT Press, 1998. 465 p.

PETASIS, G.; PALIOURAS, G.; SPYROPOULOS, C. D.; HALATSIS, C. eg-GRIDS: context free grammatical inference from positive examples using genetic search. In: 7th International Colloquium on Grammatical Inference (ICGI 2004). **Lecture Notes in Computer Science**, v. 3264, p. 223-234. Springer-Verlag, 2004.

POLI, R.; LANGDON, W. B.; MCPHEE, N. F.; KOZA, J. R. Genetic programming: an introductory tutorial and a survey of techniques and applications. **Technical Report CES-475**. Department of Computing and Electronic Systems, University of Essex, UK, 2007.

PRICE, K. V.; STORN, R. M.; LAMPINEN, J. A. **Differential Evolution: A Practical Approach to Global Optimization**. Springer, 2005. 566 p.

QUAMMEN, D. **The Kiwi's Egg: Charles Darwin and Natural Selection**. London: Weidenfeld & Nicolson, 2007. 286 p.

RATLE, A.; SEBAG, M. Grammar-guided genetic programming and dimensional consistency: application to non-parametric identification in mechanics. **Applied Soft Computing**, v. 1, n. 1, p. 105-118. Elsevier, 2001.

REYNOSOL, J. C. S.; MONTANO, M. A. J. Análisis de secuencias de ADN con WinGramm 2 Avances en la Ciencia de la Computación. In: **VI Encuentro Internacional de Computación (**ENC'05), p. 159-162. Sociedad Mexicana de Ciencias de la Computación (SMCC), 2005.

RODRIGUES, E.; LOPES, H. S. Genetic programming with incremental learning for grammatical inference. In: **Proceedings of 6th International Conference on Hybrid Intelligent Systems** (HIS'06), p. 47. IEEE Press, 2006.

RODRIGUES, E.; LOPES, H. S. Genetic programming for induction of context-free grammars. In: **Proceedings of 7th International Conference on Intelligent Systems Design and Applications** (ISDA'07), p. 297-302. IEEE Press, 2007.

RODRIGUES, E.; LOPES, H. S. Inferência de gramáticas livres de contexto usando programação genética. In: **I Simpósio Brasileiro de Inteligência Computacional**, 2007. CD-ROM.

ROSS, B. J. The evolution of stochastic regular *motifs* for protein sequences. **New Generation Computing**, v. 20, n. 2, p. 187-213. Ohmsha, 2002.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2nd Ed. New York: Prentice Hall, 2003. 1132 p.

SAKAKIBARA, Y. Grammatical inference in bioinformatics. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 27, n.7, p. 1051-1062. IEEE Press, 2005.

SCHWEHM, M.; OST, A. Inference of stochastic regular grammars by massively parallel genetic algorithms. In: **Proceedings of the 6th International Conference on Genetic Algorithms**, p. 528-535. Morgan Kaufmann, 1995.

SEARLS, D. B. The language of genes. **Nature**, v. 420, p. 211-217, 2002.

SEEHUUS, R.; TVEIT, A.; EDSBERG, O. Discovering biological motifs with genetic programming. In: **Proceedings of the Conference on Genetic and Evolutionary Computation** (GECCO 2005), p. 401-408. ACM Inc, 2005.

SETUBAL, J. C.; MEIDANIS, J. **Introduction to Computational Molecular Biology**. San Francisco: PWS Publishing Company, 1997. 296 p.

SIGRIST, C. J. A.; CERUTTI, L.; HULO, N.; GATTIKER, A.; FALQUET, L.; PAGNI, M.; BAIROCH, A.; BUCHER, P. PROSITE: a documented database using patterns and profiles as motif descriptors. **Briefings in Bioinformatics**, v. 3, n. 3, p. 265–274. Oxford Journals, 2002.

SMITH, T. C.; WITTEN, I. H. A genetic algorithm for the induction of natural language grammars. In: **Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)**. Workshop on New Approaches to Learning for Natural Language Processing, p. 17-24. Morgan Kaufmann, 1995.

SOLTO, M. C. P.; LORENA, A. C.; DELBEM, A. C. B.; CARVALHO, A. C. P. L. F. Técnicas de aprendizado de máquina para problemas da biologia molecular. In: **Anais do XXIII Congresso da Sociedade Brasileira de Computação**, III MCIA, v. VIII, p.103-152. Sociedade Brasileira de Computação (SBC), 2003.

SPECTOR, L. C. **Automatic Quantum Computer Programming: A Genetic Programming Approach**. New York: Kluwer Academic Publishers, 2004. 153 p.

STARKIE, B.; COSTE, F.; ZAANEN, M. The Omphalos context-free grammar learning competition. In: Proceedings of the 7th International Colloquium on Grammatical Inference (ICGI 2004). **Lecture Notes in Computer Science**, v. 3264, p. 281-282. Springer-Verlag, 2004.

TAN, A. C.; GILBERT, D.; DEVILLE, Y. Multi-class protein fold classification using a new ensemble machine learning approach. In: Proceedings of the 14th International Conference on Genome Informatics. **Genome Informatics**, v. 14, p. 206–217. Universal Academy Press, 2003.

TAVARES, L. G.; LOPES, H. S.; LIMA, C. R. E. Estudo comparativo de métodos de aprendizado de máquinas na detecção de regiões promotoras de genes de *Escherichia Coli*. In: **Anais do I Simpósio Brasileiro de Inteligência Computacional**, I SBIC, 2007. CD-ROM.

TOMITA, M. Dynamic construction of finite automata from examples using hill climbing. In: **Proceedings of the 4th Annual Cognitive Science Conference**, p. 105- 108, 1982.

TORRES, A.; NIETO, J. J. Fuzzy logic in medicine and bioinformatics. **Journal of Biomedicine and Biotechnology**, v. 2006, p. 1-7. Hindawi Publishing Corporation, 2006.

TOWELL, G.; SHAVLIK, J.; NOORDEWIER, M. Refinement of aproximate domain theories by knowledge-based neural networks. In: **Proceedings of the 8th National Conference on Artificial Intelligence**, p. 861-866. MIT Press, 1990.

TSUNODA, D. F.; LOPES, H. S. Feature discovery in a protein database using a memetic algorithm-based solution. In: **Proceedings of the 1st International Conference on Bioinformatics and Computational Biology** (ICoBiCoBi), 2003. CD-ROM.

UNOLD, O. Grammar-based classifier system for recognition of promoter regions. In: Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA). **Lecture Notes in Computer Science**, v. 4432, p. 798-805. Springer-Verlag, 2007.

VAPNIK, V. N. **Statistical Learning Theory**. New York:Wiley-Interscience, 1998. 736 p.

WEINERT, W. R.; LOPES, H. S. GEPCLASS: a classification rule discovery tool using gene expression programming. In: Advanced Data Mining and Applications. **Lecture Notes in Computer Science**, v. 4093, p. 871-880. Springer-Verlag, 2006.

WEISS, S. M.; KULIKOWSKI, C. A. **Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems**. San Francisco: Morgan Kauffmann, 1991. 223 p.

WHIGHAM, P. A. Grammatically based genetic programming. In: **Proceedings of Machine Learning Workshop on Genetic Programming: from theory to real world applications**, p. 33-41. Morgan Kaufmann, 1995.

WINSTON, P. H.; HORN, B. K. P. **LISP**. 3rd Ed. Reading: Addison-Wesley, 1989. 611 p.

WINTNER, S. On the semantics of unification grammars. **Grammars**, v. 6, n. 2, p. 145-153. Springer, 2003.

WITTEN, I. H.; FRANK, E. **Data Mining - Practical Machine Learning Tools and Techniques**. 2nd Ed. San Francisco: Elsevier, 2005. 525 p.

WYARD, P. Context free grammar induction using genetic algorithms. In: **Proceedings of the IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives**, p. P11/1-P11/5. IEE, 1993.

WYARD, P. Representational issues for context-free grammar induction using genetic algorithms. In: Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications. **Lecture Notes in Computer Science**, v. 862, p. 222-235. Springer-Verlag, 1994.

YOUNGER, D. H. Recognition and parsing of context-free languages in time n^3. **Information and Control**, v. 10, n. 2, p. 189-208, 1967.

YU, T.; BENTLEY, P. Methods to evolve legal phenotypes. In: Proceedings of the Parallel Problem Solving from Nature (PPSN V). **Lecture Notes in Computer Science**, v. 1498, p. 280–291. Springer-Verlag, 1998.

ZADEH, L. A. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338–353, 1965.

ZHOU, H.; GREFENSTETTE, J. J. Induction of finite automata by genetic algorithms. In: **Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics**, p. 170–174. IEEE Press, 1986.

ZHOU, W.; ZHU, H.; LIU, G.; HUANG, Y.; WANG, Y.; HAN, D.; ZHOU, C. A novel computational based method for discovery of sequence *motifs* from coexpressed genes. **International Journal of Information Technology**, v. 11, n. 8. Information Communication Institute of Singapore, 2005.

RESUMO

A inferência gramatical lida com o problema de aprender um classificador capaz de reconhecer determinada construção ou característica em um conjunto qualquer de exemplos.

Neste trabalho, um modelo de inferência gramatical baseado em uma variante de Programação Genética é proposto. A representação de cada indivíduo é baseada em uma lista ligada de árvores representando o conjunto de produções da gramática. A atuação dos operadores genéticos é feita de forma heurística. Além disto, dois novos operadores genéticos são apresentados. O primeiro, denominado Aprendizagem Incremental, é capaz de reconhecer, com base em exemplos, quais regras de produção estão faltando. O segundo, denominado Expansão, é capaz de prover a diversidade necessária. Em experimentos efetuados, o modelo proposto inferiu com sucesso seis gramáticas regulares e duas gramáticas livres de contexto: parênteses e palíndromos de quatro letras, tanto o comum quanto o disjunto, sendo superior a abordagens recentes.

Atualmente, modelos de inferência gramatical têm sido aplicados a problemas de reconhecimento de seqüências biológicas de DNA. Neste trabalho, dois problemas de identificação de padrão foram abordados: reconhecimento de promotores e *splice-junction*. Para o primeiro, o modelo proposto obteve resultado superior a outras abordagens. Para o segundo, o modelo proposto apresentou bons resultados.

O modelo foi estendido para o uso de gramáticas *fuzzy*, mais especificamente, as gramáticas *fuzzy* fracionárias. Para tal, um método de estimação adequado dos valores da função de pertinência das produções da gramática é proposto. Os resultados obtidos na identificação de *splice-junctions* comprovam a utilidade do modelo de inferência gramatical fuzzy proposto.

PALAVRAS-CHAVE:

Inferência Gramatical, Programação Genética, Gramáticas *fuzzy*, Bioinformática.

ÁREA/SUB-ÁREA DE CONHECIMENTO:

1.03.03.04-9 - Sistemas de Informação
1.03.01.02-0 - Linguagens Formais e Autômatos

2007

Nº 30