INFERÊNCIA DE GRAMÁTICAS LIVRES DE CONTEXTO USANDO PROGRAMAÇÃO GENÉTICA

ERNESTO RODRIGUES, HEITOR SILVÉRIO LOPES

Laboratório de Bioinformática, CPGEI, Universidade Tecnológica Federal do Paraná
Av. Sete de Setembro, 3165-CEP: 80230-901- Curitiba - PR- Brasil
ernesto@cpgei.cefetpr.br, hslopes@pesquisador.cnpq.br

Abstract— We proposed an evolutionary algorithm for the inference of context-free grammars from positive and negative examples. The algorithm is based on genetic programming and uses a local optimization operator that is capable of improving the learning task. Ordinary genetic operators were modified so as to bias the search and a new operator was proposed. The system was evaluated using benchmark problems and results were compared with another recent approach. Results show that the proposed approach is very promising.

Keywords— Grammatical Inference, Genetic Programming, Context free grammars

Resumo— Neste artigo propõe-se um algoritmo evolucionário para a inferência de gramáticas livres de contexto a partir de exemplos positivos e negativos. O algoritmo é uma variação da Programação Genética e usa um operador de busca local capaz de melhorar sua convergência. Alterações nos operadores genéticos são propostas para facilitar a busca e um novo operador é apresentado. Para a validação do método, dois tipos de gramáticas foram usados: parênteses e palíndromos. Os resultados foram comparados com uma recente abordagem. Os resultados obtidos comprovam que a abordagem é promissora.

Palavras-chave— Inferência Gramatical, Programação Genética, Gramáticas livres de contexto

1 Introdução

A inferência gramatical, também conhecida como indução gramatical, é a tarefa de aprender uma gramática a partir de um conjunto de exemplos. Além disso, a gramática inferida deve ser capaz de reconhecer também outros exemplos não apresentados durante sua aprendizagem.

Existem vários algoritmos conhecidos para inferência gramatical, porém somente para as gramáticas regulares a exatidão da inferência é garantida. Para as gramáticas livres de contexto, alguma abordagem recente tem apresentado relativo sucesso dado que o espaço de gramáticas possíveis é infinito. Desta forma, inferir gramáticas livres de contexto ainda é um desafio para a inferência gramatical.

As aplicações da inferência gramatical são muitas tais como reconhecimento de padrões, análise de seqüências biológicas dentre outras.

Neste artigo, propõem-se uma abordagem para inferência gramatical livre de contexto baseada em Programação Genética (PG) usando um mecanismo de busca local. O uso de PG em inferência de gramáticas livres de contexto não é novo, mas apenas recentemente tem apresentado resultados promissores (Rodrigues e Lopes, 2006 Javed, Bryant, Crepinsek, Mernik e Sprague, 2004).

Para validar a abordagem proposta, foi utilizado um conjunto de exemplos de gramáticas livres de contexto conhecido e usado em uma publicação recente (Clark, Florêncio e Watkins, 2006).

A técnica de PG se baseia na evolução de um conjunto de programas com o objetivo de aprendizagem por indução. A idéia é ensinar computadores a se programar, isto é, a partir de especificações de comportamento, o computador deve ser capaz de induzir um programa que as satisfaça (Koza, 1992). A cada programa é associado um valor de mérito (*fitness*) representando o quanto ele é capaz de resolver o problema. Basicamente, a PG mantém uma população de programas de computador, usa métodos de seleção baseados na capacidade de adaptação (*fitness*) de cada programa (escolha dos "melhores"), aplica operadores genéticos para modificá-los e convergir para uma solução. O objetivo é encontrar uma solução, no espaço de todos os programas possíveis, usando apenas um valor de *fitness* como auxílio no processo de busca.

A próxima seção define a classe de gramáticas livres de contexto que são objeto do método proposto. Em seguida, a abordagem de PG usada é descrita. Os experimentos feitos demonstrando o potencial do método proposto são então apresentados seguidos da discussão dos resultados. Ao final, conclusões são apresentadas.

2 Gramáticas Livres de Contexto

Uma gramática livre de contexto (GLC) é definida pela quádrupla $G = (N, \sum, P, S)$, onde N é o conjunto de símbolos não-terminais, $\sum$ é o conjunto de símbolos terminais (alfabeto) e P é uma lista finita de produções na forma $A \rightarrow \alpha$ sendo $A \in N$ e $\alpha \in (N \cup \sum)^*$. O símbolo S é um não-terminal denominado símbolo inicial.

Uma sentença da gramática é formada somente por símbolos terminais escolhidos do conjunto Σ. Por outro lado, o conjunto N representa as classes sintáticas da gramática, isto é, os componentes

usados para se descrever a construção das sentenças. O conjunto de todas as sentenças geradas pela gramática *G* forma uma linguagem e é denotada por L(*G*). A geração das sentenças é feita a partir do símbolo inicial *S* aplicando-se as produções existentes em *P*. Portanto, L(*G*) = { $x \mid S \Rightarrow^* x, x \in \Sigma^*$ }. Caso uma sentença contendo símbolos terminais não possa ser gerada por *G*, então ela não pertence à linguagem L(*G*).

Dada uma gramática livre de contexto *G*, é possível determinar se uma determinada sentença pertence ou não à linguagem L(*G*). Um algoritmo eficiente é o CYK que apenas exige que *G* esteja na forma normal de Chomsky.

Uma gramática livre de contexto *G* está na forma normal de Chomsky se e somente se todas as suas produções sejam da forma $A \rightarrow BC$ ou $A \rightarrow \alpha$ onde A, B, C $\in N$ e $\alpha \in \Sigma$. É importante frisar que qualquer gramática livre de contexto pode ser colocada na forma normal de Chomsky através de sucessivas transformações sem perda de expressividade.

2.1 Algoritmo CYK

Para determinar se uma sentença pertence ou não a uma gramática livre de contexto G na forma normal de Chomsky, o algoritmo CYK constrói uma tabela triangular. As posições de cada caractere dentro da sentença são representadas no eixo horizontal. A variável V_{rs} é o conjunto de variáveis $A \in P$ tais que $A \Rightarrow^* a_r\ a_{r+1}\ ...\ a_s$. O objetivo é verificar se o símbolo inicial *S* está presente na variável V_{1n} porque isto seria o mesmo que afirmar $S \Rightarrow^* w$, ou seja, $w \in$ L(*G*).

Para preencher a tabela, avança-se linha a linha de baixo para cima usando os seguintes passos:

- Preencher a linha de baixo da tabela triangular com os símbolos não-terminais que geram diretamente cada um dos caracteres da sentença sendo analisada.

```
for r = 1 to n do
  Vr1 = { A | A → ar ∈ P }
```

- Preencher as outras linhas (de baixo para cima) com as produções que geram os dois não-terminais existentes abaixo.

```
for s = 2 to n do
  for r = 1 to n-s+1 do
    Vrs = ∅
    for k = 1 to s - 1 do
     Vrs = Vrs ∪ {A | A → BC ∈ P,
                  B ∈ Vrk and
                  C ∈ V(r+k)(s-k) }
```

A Figura 1 mostra um exemplo de uma tabela triangular obtida da sentença "aba" usando o conjunto de produções P = { $S \rightarrow AA$; $S \rightarrow AS$; $S \rightarrow b$; $A \rightarrow SA$; $A \rightarrow AS$; $A \rightarrow a$ }. Como o símbolo *S* aparece no topo da tabela triangular (V_{1n}), então a sentença pertence à linguagem gerada pela gramática.

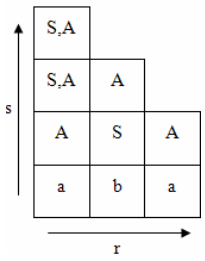

Figura 1. Exemplo de tabela triangular obtida pelo CYK.

3 Programação Genética

A Programação Genética (PG) é uma abordagem para a geração automática de programas de computador desenvolvida por John Koza (Koza, 1992). A técnica se baseia na combinação de idéias da teoria da evolução (seleção natural), genética (reprodução, cruzamento e mutação), inteligência artificial (busca heurística) e teoria de compiladores (representação de programas como árvores sintáticas). Basicamente, a PG é um algoritmo que busca, dentre um espaço relativamente grande, porém restrito de programas de computador, uma solução ou, pelo menos, uma boa aproximação para resolver determinado problema.

3.1 Algoritmo da PG

O algoritmo da PG pode ser resumido como:

- Criar aleatoriamente uma população de indivíduos (programas);
- Executar os seguintes passos até que um critério de término seja satisfeito:
  - Avalie o fitness de cada indivíduo;
  - Aplique um método de seleção baseado em *fitness* para escolher indivíduos;
  - Modifique os indivíduos escolhidos através da aplicação de operadores genéticos, tais como reprodução, cruzamento e mutação.

A representação dos indivíduos em PG tradicionalmente se baseia em árvore de sintaxe abstrata, isto é, os indivíduos são formados pela livre combinação de funções, variáveis e constantes adequados ao domínio do problema.

Para cada indivíduo, a avaliação é feita através do uso de um conjunto de exemplos de treinamento conhecido como *fitness cases* (Koza, 1992). Este conjunto é composto basicamente por um conjunto de valores de entrada e seus respectivos valores desejados de saída. Normalmente, o valor de *fitness* é o desvio entre o valor esperado de saída e o valor retornado pelo indivíduo.

Há dois métodos de seleção tradicionalmente usados em PG: *fitness proportionate* e torneio. No primeiro método, os indivíduos são selecionados aleatoriamente com probabilidade proporcional ao seu *fitness*. Isto é, quanto melhor o *fitness*, mais a

chance de ser escolhido. No segundo método, um número fixo de indivíduos são escolhidos aleatoriamente e quem apresentar o melhor *fitness* é o escolhido. Neste artigo, foi utilizado o torneio.

A reprodução é um operador genético que simplesmente copia o indivíduo para a próxima geração. O cruzamento combina partes de dois indivíduos anteriormente escolhidos para criar dois novos. A mutação altera aleatoriamente uma pequena parte de um indivíduo.

Após a geração dos novos indivíduos, a população é substituída. O processo é interrompido quando uma solução é encontrada ou um número pré-determinado de gerações é alcançado.

4 Inferência Gramatical com PG

Ao aplicar-se PG à inferência de gramáticas livres de contexto (GLC), os indivíduos não são programas de computador e sim gramáticas. Uma representação adequada deve ser feita a fim de permitir a aplicação dos operadores genéticos sem tornar o indivíduo inválido (gramática inconsistente).

Neste artigo, a gramática é representada como uma lista de árvores estruturadas. Cada árvore representa uma produção com seu lado esquerdo como nó raiz e as derivações como folhas. A Figura 2 exibe a gramática G = (N, $\sum$, P, S) sendo $\sum$ = { a, b }, N = { S, A } e P = { $S \rightarrow$ A S ; $S \rightarrow$ b; A $\rightarrow$ SA ; A $\rightarrow$ a }.

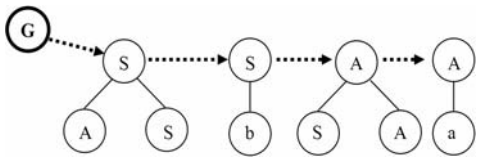

Figura 2. Representação de uma GLC como uma lista de árvores.

Com esta representação, os operadores genéticos atuam isoladamente nas produções, substituindo-as quando possível.

A população inicial pode ser criada com produções aleatórias, porém com a garantia de que todas as produções são alcançáveis, direta ou indiretamente, pelo símbolo inicial S. Para tal, estabelece-se o número de produções desejado e a geração de cada produção referencia os não-terminais criados anteriormente. Ao menos uma das produções criadas tem seu lado esquerdo igual ao símbolo inicial.

Após a criação da população inicial, começa o processo evolutivo conforme o algoritmo apresentado na seção 3.1 e somente termina se for encontrada a gramática desejada ou um número máximo de gerações for alcançado.

Para auxiliar o processo de inferência, cada produção de cada gramática tem associados dois valores: um escore positivo e um escore negativo.

Durante a avaliação de cada gramática, cada produção que for usada para reconhecer um exemplo positivo recebe um incremento em seu escore positivo. Da mesma forma, cada produção que for usada para reconhecer um exemplo negativo, recebe um incremento em seu escore negativo. Os escores são usados para direcionar a atuação dos operadores genéticos.

O operador de cruzamento permite a troca de produções entre duas gramáticas escolhidas previamente. A escolha da produção é feita com base nos escores obtidos durante a avaliação de *fitness*. Na primeira gramática, a produção que tiver o maior escore negativo é escolhida. Se houver mais de uma produção com o mesmo escore, a que tiver menor escore positivo é escolhida. Se a segunda gramática não tiver nenhuma produção com o mesmo símbolo não-terminal como cabeça da produção (lado esquerdo), o cruzamento não é feito e as duas gramáticas são copiadas para a geração seguinte. Caso contrário, trocam-se as suas derivações (lados direitos das produções). A exigência do ponto de cruzamento ser entre produções de mesmo lado esquerdo é para se evitar que as gramáticas geradas sejam inconsistentes.

A mutação é aplicada sobre uma gramática escolhida anteriormente. A escolha da produção a ser mutada segue o mesmo mecanismo do cruzamento. Uma nova produção com o mesmo lado esquerdo, porém com o seu lado direito gerado aleatoriamente, substitui a produção escolhida.

Infelizmente, apenas direcionando a aplicação dos operadores com o uso dos escores, a convergência do algoritmo não é garantida. Recentemente, demonstramos que o uso de um operador de aprendizagem incremental é necessário (Rodrigues e Lopes, 2006). Para isto, informações obtidas pelo algoritmo CYK são usadas para descobrir qual produção está faltando e permitir a convergência do algoritmo. Para fins de completeza deste artigo, o operador de aprendizagem está descrito na seção 4.1.

Nos experimentos, percebeu-se que também é necessário um operador capaz de estender a gramática, ou seja, adicionar novas produções. Este operador proposto está descrito na seção 4.2.

4.1 Operador de Aprendizagem Incremental

Os operadores de cruzamento e mutação são operadores globais e, portanto, incapazes de fazer otimização local tais como adicionar uma produção que está faltando para se obter uma solução. Além disso, com o valor de *fitness* não é possível saber quais produções estão faltando. Com base nisto, em um trabalho anterior, um novo operador denominado *Incremental Learning* (Aprendizagem Incremental) foi proposto para fazer a busca local (Rodrigues e Lopes, 2006). Este operador é aplicado sempre antes da avaliação de cada gramática na população. Ele usa

a tabela triangular obtida pela aplicação do algoritmo CYK durante a análise dos exemplos positivos para permitir a adição de novas, porém úteis, produções. Para cada exemplo positivo, os seguintes passos são feitos:

- Construa a tabela triangular do CYK com os elementos V_{rs} conforme descrito na seção 2.1
- Se o exemplo não é reconhecido:
  - Se V_{1n} não está vazio, copie uma das produções em V_{1n} alterando o seu lado esquerdo para o símbolo inicial S.
  - Se V_{1n} está vazio, adicione uma nova produção com a forma S → AB tal que A referencia a primeira metade e B referencia a segunda metade da tabela triangular CYK. Se A ou B estão vazios, o operador não é aplicado.

Ao término deste processo, vários (possivelmente todos) exemplos positivos serão reconhecidos pela gramática afetada por este operador. Todavia, não há nenhuma garantia de que alguns exemplos negativos continuem a ser rejeitados pela gramática.

4.2 Operador de Expansão

Na abordagem apresentada anteriormente (Rodrigues e Lopes, 2006), o número de produções de cada gramática é fixado durante a criação da população inicial. Exceto pela atuação do operador de aprendizagem, 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, o método é incapaz de criá-la. Em experimentos com gramáticas mais complexas, observou-se que, em alguns casos, a convergência é prejudicada pelo fato da gramática ser incapaz de gerar novas produções espontaneamente.

Sendo assim, propõe-se um novo operador denominado expansão. Ele atua da seguinte forma: inicialmente um novo símbolo não-terminal é criado. Em seguida, gera-se uma nova produção contendo este novo não-terminal como lado esquerdo. Esta nova produção é então adicionada à gramática. Desta forma, a gramática pode aumentar dinamicamente em tamanho.

Para tornar esta recém-criada produção útil, adiciona-se também mais uma produção à gramática cujo lado esquerdo é o símbolo inicial *S* e o lado direito contem o não-terminal recém-criado mais um não-terminal escolhido aleatoriamente. Portanto é importante frisar que o operador de expansão adiciona duas novas produções à gramática.

Este operador é capaz de promover diversidade na população, necessária nas gerações iniciais.

4.3 Avaliação das Gramáticas

Em PG usa-se um conjunto de *fitness cases* para avaliar cada indivíduo na população. Este conjunto é formado por exemplos tipo entrada-saída, isto é, dado um conjunto de valores de entrada, identifica-se qual é o valor desejado para a saída. No caso de inferência gramatical, os *fitness cases* são dois conjuntos: um formado pelas sentenças que pertencem à gramática desejada (exemplos positivos) e outro composto pelas que não pertencem (exemplos negativos).

Para avaliação, adotou-se a matriz de confusão que representa uma ferramenta útil para aprendizagem supervisionada. Nesta matriz, cada coluna representa o número de exemplos classificados como positivo ou negativo pela gramática, enquanto que cada linha representa a real classificação de cada exemplo. O formato da matriz de confusão está na Tabela 1.

Tabela 1. Matriz de Confusão

	Previsto Positivo	**Previsto Negativo**
Exemplo Positivo	Verdadeiro Positivo (VP)	Falso Negativo (FN)
Exemplo Negativo	False Positivo (FP)	Verdadeiro Negativo (VN)

Cada célula da matriz de confusão tem o seguinte significado no contexto do nosso estudo:

- VP é o número de exemplos positivos corretamente aceitos pela gramática;
- VN é o número de exemplos negativos corretamente rejeitados pela gramática;
- FP é o número de exemplos negativos incorretamente aceitos pela gramática;
- FN é o número de exemplos positivos incorretamente rejeitados pela gramática.

Existem várias medidas que podem ser extraídas da matriz de confusão. A mais comum é a taxa de acerto obtida pela razão entre o total de exemplos corretamente classificados e o total de exemplos. Para uma melhor inferência, neste artigo aplicaram-se outras duas medidas: sensitividade (Equação 1) e especificidade (Equação 2). Estas medidas permitem avaliar como os exemplos positivos e negativos são corretamente classificados.

$$sensitividade = \frac{VP}{VP + FN} \qquad (1)$$

$$especificidade = \frac{VN}{VN + FP} \qquad (2)$$

O valor do *fitness* é calculado como o produto da sensitividade pela especificidade, permitindo uma melhor orientação na escolha dos melhores indivíduos. Este produto foi proposto inicialmente por Lopes, Coutinho e Lima (1998) e tem sido usado em vários problemas de classificação.

Antes de serem avaliadas, todas as gramáticas são verificadas quanto a sua consistência e corrigidas

se necessário, ou seja, produções inúteis ou redundantes são eliminadas.

5 Experimentos

O algoritmo proposto foi implementado em C++ cujo código fonte pode ser obtido com os autores por *e-mail*. As execuções foram feitas em uma máquina Pentium D 915 Dual Core de 2.8 GHz com 1 Gbyte de memória rodando Debian Linux 4. A Tabela 2 mostra os parâmetros usados.

Para validar o algoritmo, escolhemos três problemas clássicos de inferência: parêntesis, palíndromo simples e palíndromo intercalado. Neste último caso, a seqüência pode conter internamente um palíndromo, tal como "abcabbab" onde "abba" é um palíndromo interno.

A população inicial foi criada com base em uma distribuição uniforme de produções, variando de 10 a 59. Porém a quantidade real de produções pode ser diferente, pois as gramática são corrigidas quanto a sua consistência antes de serem avaliadas, isto é, produções inúteis são eliminadas.

Tabela 2. Parâmetros usados na abordagem

Execuções	10
Tamanho da população	500
População Inicial	
Número mínimo de produções	10
Número máximo de produções	59
Tamanho do torneio	7
Probabilidade de cruzamento	60%
Probabilidade de mutação	30%
Probabilidade de expansão	10%

Para treinamento e teste, usou-se procedimento de validação cruzada com dez partições com exemplos distribuídos equitativamente (*10-fold stratified cross-validation*). Neste procedimento, os exemplos são aleatoriamente distribuídos em dez partições, preservando-se em cada partição a mesma proporção de exemplos positivos e negativos do conjunto inteiro. A cada execução, uma partição é separada e as nove restantes são usadas para treinamento. A avaliação é feita com exemplos da partição não usada no treinamento. Os resultados apresentados são a média dos valores obtidos em todas as avaliações.

Para evitar que seja inferida uma gramática excessivamente genérica, há a necessidade tanto de exemplos positivos quanto de negativos. Porém, exemplos negativos são difíceis de serem construídos. Gerar simplesmente seqüências aleatórias não produz um conjunto de exemplos suficientemente completo para distinguir-se uma gramática correta (desejada) de outra semelhante, porém errada.

Recentemente, Clark e seus colegas (2006) apresentaram uma nova abordagem de inferência gramatical baseada em *String Kernels* e fez comparações com outras abordagens conhecidas, tais como gramáticas livre de contexto probabilísticas (do inglês, *probabilistic context-free grammars,* PCFG) e modelos ocultos de Markov (do inglês *Hidden Markov Models,* HMM). Eles aplicaram SK à inferência tanto de gramáticas livres de contexto quanto a sensíveis a contexto.

Para permitir uma comparação direta, o mesmo conjunto de exemplos para a inferência das gramáticas para parêntesis e palíndromo simples foi usado. Para a gramática de palíndromo intercalado, o conjunto foi montado com base nos exemplos de palíndromo simples. O conjunto de exemplos para parêntesis é formado por 537 exemplos positivos e 463 negativos. Para o palíndromo, são 510 exemplos positivos e 490 negativos.

A Tabela 3 mostra os resultados obtidos para as duas primeiras gramáticas e sua comparação com os valores de Clark e seus colegas (2006). A taxa de erro negativa (do inglês, *negative error rate*, NER) e a taxa de erro positiva (do inglês, *positive error rate,* PER) são calculadas pelas Equações 3 e 4. Na tabela, estes valores são representados como N e P, respectivamente. O uso destes valores foi feito para facilitar a comparação com os valores apresentados no artigo do Clark e seus colegas (2006). Para cada conjunto, apresentam-se as taxas e quanto menor o valor, melhor.

$$NER = \frac{FP}{FP + TN} \quad (3)$$

$$PER = \frac{FN}{TP + FN} \quad (4)$$

Tabela 3. Comparação dos valores obtidos pela nossa abordagem (GPGI) com as de Clark e seus colegas (2006)

	PCFG		HMM		SK		GPGI	
	N	P	N	P	N	P	N	P
1	0	0	3	1	10	0	0	0
2	6	0	84	3	16	0	14	0

Na abordagem proposta (representada como GPGI na Tabela 3), encontrou-se com sucesso a gramática desejada para parêntesis (representada pela linha 1) em todas as dez execuções em, no máximo, três gerações. No palíndromo (representado pela linha 2), obtivemos uma média de 90% de taxa de acerto.

Na Figura 3 apresentam-se os valores de sensitividade (Sens), especificidade (Esp) e a taxa de acerto (Tac) obtidos pela melhor gramática da inferência de palíndromos simples durante as quinze primeiras gerações. Nesta figura, durante as primeiras sete gerações, o algoritmo tende a melhorar as taxas de sensitividade e especificidade. Entre a oitava e a décima-primeira, os valores de especificidade e sensitividade se alternam. Este comportamento é causado pelo operador de aprendizagem que tenta melhorar a cobertura dos exemplos positivos (aumenta sensitividade) e provoca a cobertura de exemplos negativos (diminui

especificidade). O operador de expansão proposto pode causar o mesmo comportamento.

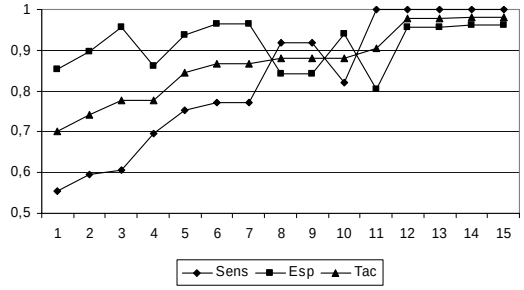

Figura 3. Valores obtidos nas primeiras quinze gerações para a inferência de palíndromos simples.

A Figura 4 mostra a variação dos valores mínimo, médio e máximo do *fitness* durante as primeiras quinze gerações da inferência de palíndromos simples. A lenta convergência auxilia o algoritmo a evitar um ótimo local. Para o caso dos palíndromos intercalados, o comportamento foi semelhante, porém com população oito vezes maior (4000).

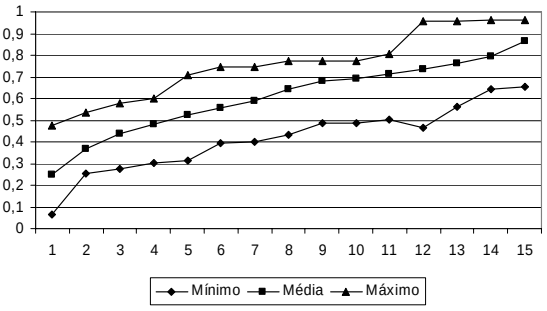

Figura 4. Variação do valor *fitness* durante as quinze primeiras gerações da inferência de palíndromos simples.

Um conhecido problema de PG é denominado *bloat*. Este problema se refere ao crescimento descontrolado do tamanho dos indivíduos de uma população. A Figura 5 mostra a variação do tamanho das gramáticas durante as primeiras quinze gerações da inferência de palíndromos simples. A curva demonstra que o operador de expansão é útil durante as primeiras oito gerações.

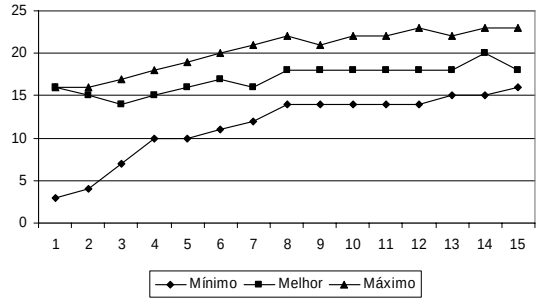

Figura 5. Variação do tamanho das gramáticas para a inferência de palíndromos simples.

Comparando esta figura com as Figuras 3 e 4, é possível perceber que o operador de expansão cria produções úteis nas primeiras gerações, portanto provendo diversidade genética. Após a oitava geração, o tamanho das gramáticas cresce levemente devido ao fato que, antes da avaliação, todas as produções inúteis são eliminadas. Sendo assim, o efeito de *bloat* não foi observado.

6 Conclusões

Neste artigo se propõem uma abordagem da PG para inferência de gramáticas livres de contexto. Cada indivíduo é representado como uma lista estruturada de árvores onde cada árvore representa uma produção. Os operadores genéticos (cruzamento e mutação) têm seu comportamento guiado pelos escores positivo e negativo de cada produção.

Um operador de busca local denominado *Incremental Learning* anteriormente apresentado (Rodrigues e Lopes, 2006) foi aplicado para ajustar cada gramática com base nos exemplos positivos. Além disso, apresenta-se um novo operador denominado Expansão que adiciona novas produções à gramática, permitindo o seu aumento em tamanho. Este novo operador provê a diversidade necessária nas primeiras gerações.

Resultados obtidos na inferência de gramática para parêntesis e palíndromos foram melhores que os obtidos por uma recente publicação que usou HMM e *String Kernels* (Clark, Florêncio e Watkins, 2006). Apenas com relação à PCFG, o método proposto foi levemente inferior. Estes resultados demonstram a viabilidade da abordagem proposta.

Referências Bibliográficas

Clark, A.; Florêncio, C.C. e Watkins, C. (2006). Languages as hyperplanes: grammatical inference with string kernels, *Proceedings of European Conference on Machine Learning*, LNAI Vol. 4212, Springer-Verlag, Berlin, pp. 90–101.Higuera, C. (2005) A bibliographical study of grammatical inference, *Pattern Recognition*, Vol. 38, no. 9, pp. 1332-1348.

Javed, F., Bryant, B., Crepinsek, M., Mernik, M. e Sprague, A. (2004) Context-free grammar induction using genetic programming, *Proceedings of the 42nd Annual ACM Southeast Conference*, Huntsville, AL, pp. 404-405.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Natural Selection, MIT Press, Cambridge, MA.

Rodrigues, E. e Lopes, H.S. (2006) Genetic programming with incremental learning for grammatical inference, *Proceedings of the 6th International Conference on Hybrid Intelligent Systems* (HIS'06), EEE Press, Auckland, pp. 47.