INDUÇÃO DE GRAMÁTICAS LIVRES DE CONTEXTO UTILIZANDO *DATA MINING*

Rene Kultz
Sandra Mara Guse Scós Venske
Universidade Estadual do Centro-Oeste
Departamento de Ciência da Computação
Guarapuava, Paraná
renekultz@yahoo.com.br, ssvenske@unicentro.br

Resumo

O desenvolvimento de máquinas de aprendizado de gramáticas é um dos problemas mais desafiadores do estudo de inferência gramatical, possuindo diversas aplicações. Neste artigo, é proposto um algoritmo que, a partir de um conjunto de cadeias de caracteres representando exemplos positivos e de uma cadeia de caracteres representando exemplos negativos, identifique padrões sintáticos e crie gramáticas livres de contexto que aceitem todos os exemplos positivos e rejeitem todos os exemplos negativos. Para atingir este objetivo, o algoritmo é apoiado por um processo chamado Data Mining.

1. Introdução

O desenvolvimento de máquinas de aprendizado de gramáticas é um dos problemas mais desafiadores do estudo de inferência gramatical, possuindo diversas aplicações, tais como o reconhecimento de padrões sintáticos, computação biológica, reconhecimento de linguagem natural, validação de documentos XML, dentre outros [3]. O aprendizado através de um conjunto de exemplos positivos e negativos tem sido estudado por mais de duas décadas. Entretato, a maior parte dos estudos envolveu o reconhecimento de gramáticas regulares, que constituem a classe de gramáticas mais simples na hierarquia de linguagens formais de Chomsky [5]. Outros estudos envolvendo gramáticas livres de contexto tiveram bons resultados apenas com exemplos simples, não sendo eficientes em exemplos extensos, especialmente porque se basearam em técnicas de computação natural [3].

Este trabalho propõe um algoritmo que recebe como entrada um conjunto de cadeias representando exemplos positivos e um conjunto de cadeias representando exemplos negativos. A partir desta entrada, o algoritmo identifica padrões sintáticos nas cadeias e produz um novo conjunto de gramáticas livres de contexto que aceite todos os exemplos positivos e que rejeite todos os exemplos negativos.

Para atingir o objetivo de realizar a inferência, o algoritmo é apoiado por um processo chamado *Data Mining*[2], para procurar conjuntos de caracteres que apareçam em seqüência com certa freqüência no conjunto de exemplos positivos.

O escopo de aplicação do algoritmo proposto envolve problemas que utilizem reconhecimento de padrões. Como exemplo específico, pode-se citar a validação de documentos XML utilizando XML *Schema* ou DTD (*Document Type Definition*).

Este artigo está organizado em 6 seções. Na seção 2 são apresentados alguns trabalhos correlatos ao proposto. Na seção 3 é mostrado o funcionamento detalhado do algoritmo desenvolvido, seguida da seção 4 que apresenta descrição de testes de funcionamento. Por fim, na seção 5 têm-se as conclusões obtidas e os trabalhos futuros.

2 Trabalhos Relacionados

Nas últimas décadas, diversos trabalhos envolvendo inferência gramatical têm sido desenvolvidos [3]. Em [5] é proposto o aprendizado de autômatos finitos determinísticos que representem uma linguagem regular utilizando vários algoritmos e fazendo uma comparação entre estes.

Quanto à inferência de gramáticas livres de contexto, em [4] é apresentado um algoritmo evolucionário que apresenta o aprendizado incremental de gramáticas livres de contexto, baseado no algoritmo de *parser* **CYK**. Da mesma forma, [3] propõe um algoritmo de aprendizado de gramáticas livres de contexto utilizando um algoritmo evolucionário. A partir de um conjunto de regras de produção aleatórias, um algoritmo evolucionário evolui as gramáticas até que estas

aceitem os exemplos positivos e rejeitem os exemplos negativos. Além disso, o algoritmo procura encontrar subpadrões regulares nos exemplos positivos, a fim de uni-los na gramática principal. Ainda são utilizadas técnicas de *Data Mining* para melhorar a qualidade da população inicial do algoritmo genético.

Além deste, [8] apresenta uma aplicação de algoritmos genéticos para a inferência de gramáticas livres de contexto. Este algoritmo se mostrou promissor para gramáticas mais simples, como um número igual de `a`'s e `b`'s em uma cadeia, mas não se mostrou eficaz ao tratar de gramáticas mais complexas.

Outros trabalhos têm como objetivo a evolução, e não o aprendizado de gramáticas. Em [7] é proposto o algoritmo LR(0) Indutivo (**ILR(0)**), uma extensão do algoritmo LR(0) que recebendo como entrada uma gramática livre de contexto e uma cadeia inicialmente não aceita por ela, adiciona novas regras de produções à gramática, gerando como saída um vetor de gramáticas que aceitem a cadeia e outras sintaticamente semelhantes a ela.

O problema da inferência gramatical motivou ainda uma competição de desenvolvimento de algoritmos de indução, como é mostrado em [6]. Esta competição propôs o desenvolvimento de algoritmos de aprendizado de gramáticas tendo como base um conjunto de exemplos positivos e de exemplos negativos. [6] mostra a maneira como foi avaliada a dificuldade dos exemplos, o número mínimo de cadeias utilizadas na indução e a maneira de avaliação das gramáticas geradas pelos competidores.

O algoritmo aqui apresentado contribui para as pesquisas nesta área de inferência gramatical. Ele combina aprendizado através de exemplos positivos e negativos com a técnica inteligente de *Data Mining*.

3. Algoritmo de Aprendizado de Gramáticas

O algoritmo objeto deste artigo tem como objetivo realizar a inferência de gramáticas livres de contexto, com base em cadeias de caracteres. Têm-se como entrada um conjunto de cadeias de caracteres representando exemplos positivos e um conjunto de cadeias de caracteres representando exemplos negativos. A partir destes, procura-se gerar gramáticas que aceitem os exemplos positivos e rejeitem os exemplos negativos.

O conjunto de cadeias de exemplos positivos precisa abranger, necessariamente, todas as estruturas possíveis definidas na linguagem que define as cadeias, visto que as gramáticas são construídas a partir dos padrões sintáticos encontrados. O número de vezes que essas estruturas aparecem nos exemplos positivos irão influenciar, diretamente, a construção das gramáticas. Quanto mais freqüente for a ocorrência das estruturas sintáticas neste conjunto, maior a chance de serem criadas regra de produção que as representem corretamente. Desta maneira, as estruturas que o usuário julga mais importantes precisam ser repetidas com mais freqüência, a fim de dar suporte ao algoritmo na geração de regras de produção.

O conjunto dos exemplos negativos, por outro lado, não precisa necessariamente conter todas as estruturas que devem ser rejeitadas, pois nesse caso, seria necessário considerar todo o universo de combinações aleatórias de símbolos terminais e não-terminais da gramática, o que, obviamente, é inviável. Entretanto, ele precisa ser grande o suficiente para definir as estruturas que, caso apareçam, desvirtuem a linguagem que define as cadeias positivas. Um exemplo disso é uma linguagem que defina parênteses balanceados. Dentre as cadeias do conjunto de exemplos negativos, é necessário haver cadeias onde essa regra seja quebrada, ou seja, cadeias com um ou mais parênteses sem fechar, ou cadeias com fecha-parênteses que não foram abertos. A inserção de cadeias que não têm absolutamente nenhuma relação com os exemplos positivos pode ser feita, mas estas pouco, ou mesmo nada, irão contribuir para a definição da linguagem.

3.1 Estrutura Básica do Algoritmo

Na Figura 1, é mostrado um fluxograma que define o funcionamento geral do algoritmo.

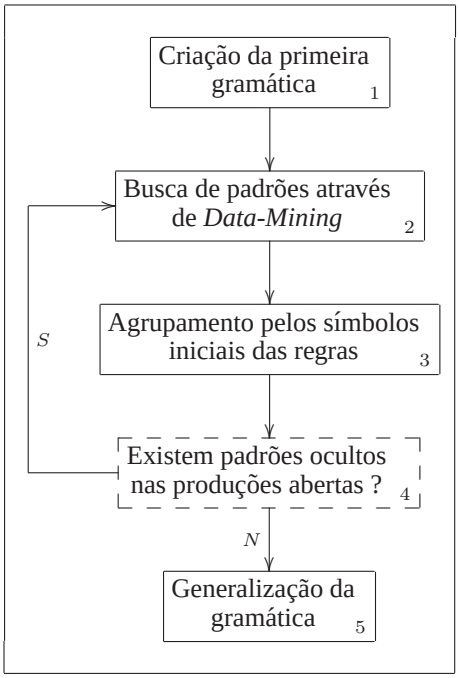

Figura 1. Fluxograma da estrutura básica do funcionamento do algoritmo

Inicialmente, é dado o conjunto V, que representa o con-

junto cadeias de exemplos positivos, e o conjunto `F`, que representa o conjunto de exemplos negativos. A primeira gramática gerada (Fig. 1 - quadro 1) deriva imediatamente do conjunto de exemplos positivos, resultando numa gramática do tipo

```
G = {{NT1}, T, P, NT1}
P = { NT1 -> V[0] | V[1] | ...  | V[n]}
```

Sendo `NT1` o símbolo inicial da gramática, `T` o conjunto de terminais da gramática, `P` o conjunto de regras de produção da gramática e `V[n]` os exemplos positivos de entrada do algoritmo. Esta gramática atende parcialmente o objetivo do algoritmo, pois aceita todos os exemplos positivos e rejeita todos os exemplos negativos. Entretanto, está longe de ser a gramática ideal, pois nenhum padrão foi descoberto, visto que a gramática se limita a criar regras de produção que derivam diretamente para o conjunto de exemplos positivos. A partir deste momento, as regras de produção passam a ser divididas em duas listas distintas:

- **Produções Abertas (PA)**: Engloba as regras de produção onde ainda podem existir padrões a ser encontrados. A busca de novos padrões é feita apenas nos elementos deste conjunto.
- **Produções Fechadas(PF)**: Engloba as regras de produção resultantes da busca de padrões através do *Data Mining* ou através do agrupamento de símbolos. Não são procurados novos padrões nas regras de produção fechadas.

O uso das lista de produções abertas e fechadas é importante porque limita a busca de padrões apenas às regras de produção onde existe real probabilidade de se encontrar novos padrões. Quando a primeira gramática é criada, a lista de produções fechadas é inicializada vazia, sendo preenchida à medida em que são encontrados os padrões na lista de produções abertas.

O preenchimento da lista de produções fechadas é feito através da aplicação do algoritmo *A Priori* de *Data Mining* (Fig. 1 - quadro 2), seguido do agrupamento das regras de produção pelos símbolos de início e fim da parte direita da regra (Fig. 1 - quadro 3). Esses dois procedimentos simplificam as regras de produção abertas e vão incrementando a lista de regras de produções fechadas, até o momento em que a lista de produções abertas fique vazia. Quando isso acontece, considera-se que não existe nenhum padrão novo a ser encontrado (Fig. 1 - quadro 4). Para finalizar, chega-se à etapa de generalização da gramática (Fig. 1 - quadro 5), onde são criadas as recursões que permitirão que ela aceite cadeias diferentes dos exemplos positivos.

3.2 Busca de Padrões Através do *Data Mining*

O uso do algoritmo *A Priori* do *Data Mining*[2] é o primeiro passo de busca de padrões, onde são encontrados padrões relativos à ocorrência de dois ou mais símbolos no conjunto de produções abertas da gramática.

O funcionamento da busca de padrões por *Data Mining* é feito em dois passos: *(i)* busca de relações entre símbolos com um grau de confiança e suporte mínimo e *(ii)* substituição dos símbolos nas regras de produção. Estes passos são detalhados nas subseções 3.3 e 3.4.

3.3 Busca de relações com grau de confiança e suporte mínimo

Nesta fase, pretende-se encontrar nas regras de produções abertas um par de símbolos que possuam uma relação direta entre eles, ou seja, sempre que for encontrado o primeiro símbolo, exista um certo grau de certeza que este é seguido pelo segundo, medido pelos valores de suporte e confiança.

Dado o conjunto de regras de produção a seguir:

```
NT1 → a b a c
NT1 → b a c d
NT1 → d a b a
```

A aplicação do *Data Mining* sobre o conjunto de regras de produção considera cada caracter como uma tupla, relacionando cada um dos símbolos da parte direita das regras de produção abertas aos símbolos que aparecem imediatamente depois destes. A relação é descrita em uma matriz $\mathtt{a}_{ij}$, onde `i` e `j` representam os elementos do alfabeto encontrado na lista de produções abertas, e a relação $\mathtt{a}_{ij}$ representa o número de vezes em que, no conjunto de produções abertas, o elemento `j` aparece imediatamente depois do elemento `i`. A Tabela 1 representa a matriz de ocorrências do conjunto de regras de produção mostrado anteriormente.

	a	**b**	**c**	**d**
a	0	2	2	0
b	3	0	0	0
c	0	0	0	1
d	1	0	0	0

Tabela 1. Matriz de Ocorrências

A matriz de ocorrências é a base para o cálculo dos valores de suporte e confiança. Considerando dois símbolos do alfabeto das regras de produção abertas, representados por α e β, as fórmulas relativas aos valores de suporte e confiança são:

$$\textbf{Suporte (Sup.)} = \frac{N^o \ de \ vezes \ em \ que \ \alpha \ e \ \beta \ aparecem \ nesta \ ordem}{N^o \ total \ de \ caracteres}$$

$$\textbf{Confiança (Conf.)} = \frac{N^o \ de \ vezes \ em \ que \ \alpha \ e \ \beta \ aparecem \ nesta \ ordem}{N^o \ de \ registros \ de \ \alpha}$$

O valor de suporte é o número de ocorrências de α e β, nesta ordem, dividido pelo número total de símbolos encontrados na parte direita das regras de produção abertas. Da mesma forma, o valor de confiança é o número de ocorrências de α e β, nesta ordem, dividido pelo número de ocorrências de α nos exemplos positivos.

Considerando a matriz de ocorrências da Tabela 1, os valores de suporte e confiança para $\alpha =$ "b" e $\beta =$ "a" são:

$$\textbf{Suporte(Sup.)} = \frac{3}{12} = 0,25$$

$$\textbf{Confiança(Conf.)} = \frac{3}{3} = 1$$

A aplicação do algoritmo *A Priori* considerando cada caracter como tupla é mais vantajoso do que se considerada cada regra de produção como uma tupla, pois no primeiro caso, um valor de confiança igual a 1 indica, sem sombra de dúvidas, um padrão válido. Em relação ao exemplo anterior, pode-se saber, com certeza, que sempre que aparecer um caracter "b" no conjunto de regras analisado, este será seguido por um caracter "a". O mesmo não pode ser dito quando considerada cada regra de produção como tupla.

Após o cálculo dos valores de suporte e confiança, seleciona-se dentre as relações aquela com os maiores valores. Se estes valores estiverem acima de um esperado, que o usuário deve definir de acordo com a precisão esperada no reconhecimento dos padrões, é feita a substituição dos símbolos nas regras de produção através do processo descrito na subseção 3.4

3.4 Substituição dos símbolos nas regras de produção

Do processo de busca de relações de símbolos com maiores valores de suporte e confiança, é feita a substituição dos conjuntos nas regras de produção, de maneira a simplificar as produções abertas.

Inicialmente, é criada uma nova regra de produção, com um novo não-terminal e derivando para a seqüência de caracteres encontrada pelo *Data Mining*. Em seguida, são procuradas nas regras de produção abertas todas as subocorrências desta seqüência e estas devem ser substituídas pelo não-terminal da regra de produção criada.

Tendo como base o mesmo exemplo citado na subseção 3.3:

```
NT1 → a b a c
NT1 → b a c d
NT1 → d a b a
```

Neste conjunto, o cálculo dos valores de suporte e confiança para a seqüência "b a" resulta nos valores 0,25 e 1, respectivamente. Fazendo a substituição desta seqüência nas regras de produção abertas, obtém-se o seguinte conjunto de regras:

```
NT1 → a NT2 c
NT1 → NT2 c d
NT1 → d a NT2
NT2 → b a
```

Após a substituição, a regra de produção originária pelo processo de *Data Mining* é adicionada à lista de produções fechadas, enquanto que as produções que sofreram a substituição continuam na lista de produções abertas. No exemplo anterior, as regras que contém como não-terminal o símbolo "NT1" continuam na lista de produções abertas, enquanto que a regra de produção com o símbolo "NT2" é adicionada à lista de produções fechadas.

Sobre este novo conjunto de produções abertas, é feita novamente a busca de padrões por *Data Mining*. A matriz de ocorrência deve ser refeita, pois a substituição tende a mudar os valores de suporte e confiança para os símbolos restantes. Este processo continua até o momento em que as seqüências de caracteres encontradas tenham valores de suporte e confiança menores do que os valores mínimos definidos pelo usuário.

3.5 Agrupamento pelos Símbolos Iniciais das Regras

Após a busca de padrões pelo *Data Mining*, é feito outro processo, o agrupamento das regras de produção. O processo de busca por *Data Mining* considera todas as regras como se fossem um único objeto, pois os valores de suporte e confiança são calculados com base no número de ocorrências de símbolos em todo o conjunto de produções abertas. Por outro lado, a busca de padrões pelo agrupamento pelos símbolos iniciais da regra considera cada regra como um único objeto, procurando ocorrências que se repitam entre duas ou mais regras.

A primeira ação a ser realizada é dividir as regras de produção abertas utilizando dois critérios: o não-terminal e o primeiro símbolo da parte direita. Dentro de cada grupo contendo estes elementos em comum, existe grande possibilidade de se encontrar outros padrões em comum.

Após a divisão, é gerada uma nova regra de produção, englobando todos os elementos iniciais e finais compartilhados por todas as regras de produção do grupo, da maneira como é mostrado no exemplo a seguir. Do grupo:

```
NT1 → a b c
NT1 → a d c
```

obtém-se:

```
NT1 → a NT2 c
NT2 → b | d
```

Todos os elementos comuns ao conjunto de regras de produção, tanto no início como no fim das regras, são agrupados em uma nova regra de produção, contendo o não-terminal antigo e derivando para um novo não-terminal. Os elementos que não são compartilhados pelo conjunto dão origem a uma nova regra de produção com o não-terminal gerado pelo agrupamento.

A regra de produção que possui o não-terminal antigo é adicionada à lista de produções fechadas, enquanto que as regras com o novo não-terminal passam a substituir as regras originais na lista de produções abertas. Desta forma, a lista de produções abertas é simplificada, tornando possível encontrar outros tipos de padrões, tanto através do agrupamento como através do *Data Mining*. No exemplo dado, a regra que contém como não-terminal o símbolo "`NT1`" é adicionada à lista de produções fechadas, enquanto que as regras de produção que contém como não-terminal o símbolo "`NT2`" permanecem na lista de produções abertas.

3.6 Generalização da Gramática

O processo de criação da árvore de generalizações ocorre imediatamente após a busca de padrões através de *Data Mining* e do agrupamento de símbolos, busca esta que é encerrada quando a lista de regras de produções abertas fica vazia. Neste novo processo, as listas de regras de produções abertas e fechadas deixam de ter utilidade. A busca passa a ser feita com o auxílio de uma árvore.

A gramática resultante do processo de busca de padrões anterior é, na maior parte dos casos, uma gramática com muitos não-terminais, com a desvantagem de não possuir nenhuma recursão. A recursão é quando um não-terminal chama, direta ou indiretamente, a si mesmo durante a derivação. Um exemplo de recursão é dado pela seguinte gramática:

$$S \rightarrow a\ S$$

São as recursões que permitem às gramáticas aceitarem um número infinito de cadeias. Como a gramática gerada pelo *Data Mining* e pelo agrupamento de símbolos não possui nenhuma recursão, ela aceita somente os exemplos positivos e nenhum outro. Obviamente, este não é o objetivo do algoritmo, pois deseja-se que a gramática final reconheça não apenas os exemplos positivos, mas também outras cadeias semanticamente semelhantes, ou seja, cadeias que sigam as mesmas regras sintáticas obedecidas pelas cadeias dos exemplos positivos.

A inserção de recursões em alguns pontos da gramática, de modo que esta possa aceitar novas cadeias além do conjunto de exemplos positivos, é chamada neste trabalho de **generalização**. No processo de generalização, o conjunto de exemplos negativos passa a ter uma importância crucial, pois é este conjunto que vai guiar todo o processo, evitando que apareçam padrões indesejáveis na gramática final.

A generalização é feita em três passos: *i)* construção da árvore de generalização; *ii* inserção das recursões e *iii* teste das generalizações nas cadeias de exemplos negativos. Cada um dos passos será detalhado nas subseções seguintes.

3.6.1 Construção da árvore de generalização

O maior problema encontrado no tocante à inserção de generalizações na gramática livre de contexto é definir o lugar exato onde seriam inseridas as recursões. Dependendo do lugar onde é inserida a "semente", um resultado distinto é alcançado. Para auxiliar na escolha na escolha do melhor lugar onde inserir uma recursão, criou-se uma estrutura chamada **árvore de generalização**, onde a gramática é representada na forma de uma árvore.

Conforme [1], uma gramática livre de contexto não pode ser representada em forma de um autômato. Em conseqüência, também não é possível representá-la em forma de árvore. Entretanto, no caso específico da gramática gerada no processo descrito nas seções 3.2 e 3.5, o fato de ainda não existirem recursões na gramática torna possível a representação em forma de árvore.

A árvore de generalização não possui o mesmo objetivo da árvore de derivação, cuja definição está relacionada à validação de cadeias usando gramáticas regulares e livres de contexto. Enquanto a árvore de derivação representa os passos que foram utilizados para se chegar do símbolo inicial da gramática até a cadeia, a árvore de generalização representa todos os lugares possíveis onde pode-se inserir uma recursão. Desta forma, o objetivo da árvore de generalização não é verificar se uma cadeia pertence a uma gramática, mas sim identificar os pontos onde pode-se inserir uma recursão em uma gramática.

Seja a gramática `G`, que tenha sido originada pelo processo de busca por *Data Mining* e agrupamento pelo símbolo inicial, definida por:

```
NT1 → a NT4 d
NT4 → a NT2 | c NT3 NT2
NT2 → d e
NT3 → f NT2
```

Para esta gramática, a árvore de generalização criada é apresentada na Figura 2.

O nó raiz da árvore de generalização não está ligado a nenhuma regra de produção, para que o símbolo inicial possa derivar duas ou mais regras de produção. Os outros nós possuem armazenados uma regra de produção e uma informação que define uma posição na parte direita da regra. A posição sempre refere-se a um não-terminal, indicando uma posição candidata para ser inserida uma recursão. As regras de produção que não possuam nenhum não-terminal, (como o nó 5 da Figura 2), não armazenam

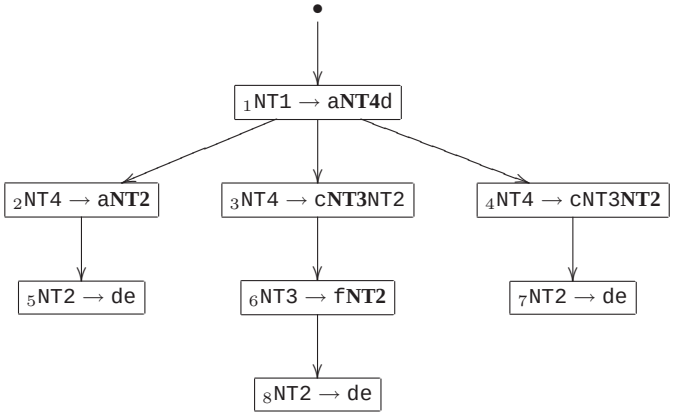

Figura 2. Exemplo de árvore de generalização

nenhuma informação quanto à posição, e em conseqüência, não dão origem a recursões.

Quando a regra de produção possui dois ou mais não-terminais, é gerado um nó para cada não-terminal, como ocorre com os nós 3 e 4 na Figura 2, onde é criado um nó para o não-terminal "NT3" e um nó para o não-terminal "NT2".

Uma vez construída a árvore de generalização, pode-se realizar a inserção das recursões na gramática, através de um processo descrito na subseção 3.6.2.

3.6.2 Inserção das recursões

A generalização da gramática é feita efetivamente pelo processo de inserção de recursões. Para tanto, utiliza-se da árvore de generalização para a definição exata do ponto onde deve ser inserida a recursão.

A cada nó da árvore de generalização está associado um valor booleano, que indica se este é um nó aberto ou fechado. Neste trabalho, definiu-se como nós abertos todos aqueles que são passíveis de originar uma recursão, enquanto os nós fechados são os nós onde já foi feita uma tentativa, bem sucedida ou não, de inserir uma recursão.

A inserção das recursões parte, inicialmente, dos nós abertos mais profundos da árvore de generalização até o nó raiz. Caso a regra de produção deste nó contenha um não-terminal na parte direita da regra, sobre esta é feita uma modificação, de modo que a regra de produção possa aceitar novas cadeias, sem que as cadeias dos exemplos positivos deixem de ser aceitas.

A generalização da gramática foi feita utilizando-se duas abordagens distintas, que neste trabalho serão chamadas de **substituição simples** e **repetição do não-terminal**, descritas nas subseções 3.6.2.1 e 3.6.2.2, respectivamente. As duas abordagens foram testadas separadamente.

3.6.2.1 Substituição simples

Neste processo, ocorre uma substituição simples do não-terminal destacado na parte direita da regra de produção. Na gramática original, é inserida uma cópia da regra de produção antiga, com a diferença que o não-terminal destacado é substituído pelo não-terminal da parte esquerda da regra.

Considerando a regra de produção a seguir, que representa o nó de número 6 na Figura 2:

NT3 → f **NT2**

Esta regra é expandida da seguinte maneira:

NT3 → f NT3 | f NT2

Neste exemplo, o não-terminal destacado, representado por NT2, é substituído pelo não-terminal da parte esquerda da regra, no caso, NT3. O efeito desta recursão é acrescentar à linguagem da gramática original um *loop* envolvendo caracter "f". A menor cadeia deste caso é o próprio caracter "f", que corresponde ao exemplo positivo de entrada do algoritmo.

Após a inserção da recursão, o nó é fechado, para que não sejam feitas novas tentativas de inserção neste ponto da árvore.

3.6.2.2 Repetição de não-terminal

A generalização por repetição de não-terminal é, de certa forma, o oposto da generalização por substituição simples. Enquanto a generalização por substituição simples cria uma recursão que envolve todos os símbolos da regra, com exceção do não-terminal selecionado, a recursão por repetição de não-terminal cria uma recursão que envolve apenas o não-terminal selecionado.

Em primeiro lugar, é criado um não-terminal novo, e este substitui na regra de produção original o não-terminal selecionado. Em seguida, este não-terminal origina duas regras de produção: uma derivada para o próprio não-terminal e outra com uma recursão para ele.

Considerando a mesma regra de produção do exemplo apresentado na subseção 3.6.2.1, que representa o nó de número 6 na Figura 2:

NT3 → f **NT2**

Esta regra é expandida da seguinte maneira:

NT3 → f NT5
NT5 → NT5 NT2 | NT2

Ao contrário da generalização por substituição simples, a generalização por repetição de não-terminal gera a recursão apenas no símbolo NT2 e não no símbolo f. Da mesma forma, o nó é fechado após a inserção da generalização.

3.6.3 Teste nos exemplos negativos

O processo de inserção de recursões não faz com que a gramática deixe de aceitar qualquer uma das cadeias de exemplos positivos, mas pode causar o surgimento de algum padrão indesejável. Para evitar que isso aconteça, a gramática generalizada é testada sobre todo o conjunto de exemplos negativos. Neste trabalho, o teste foi feito com o auxílio do analisador sintático **SLR(1)**.

Caso a generalização modifique a gramática de modo que esta aceite qualquer um dos exemplos negativos ou produza gramáticas que não sejam SLR(1), a generalização é desfeita e o processo de inserção de recursões continua.

O ponto de parada da inserção de recursões ocorre quando todos os nós da árvore de verificação tiverem sido testados.

4. Testes

Nesta seção são apresentados dois testes da aplicação do algoritmo detalhado na seção 3 para algumas cadeias exemplo. Como esta é uma proposta inicial do algoritmo, os testes realizados tiveram como objetivo a descoberta das limitações deste, a fim de que, futuramente, seja possível a realização de testes mais complexos, como os testes realizados em [6].

4.1 Teste 01

Considerando o conjunto de exemplos positivos V, definido por:

V = {“a*a”; “a-a”; “a+a.a”; “a.a”}

E o conjunto de exemplos negativos F, definido por:

F = {“a+a”}

Este é um exemplo de um conjunto pequeno de cadeias, com poucos padrões repetidos. A cadeia de exemplos negativos também oferece pouca restrição quanto aos possíveis padrões.

A última gramática gerada antes da fase de generalização é mostrada a seguir:

```
NT1 → a NT8
NT2 → * a
NT3 → - a
NT4 → + a
NT5 → NT4
NT6 → NT5 a
NT7 → .  a
NT8 → NT2 | NT3 | NT6 | NT7
```

Após a generalização pelo processo de substituição simples, obteve-se a seguinte gramática:

```
NT1 → a NT8 | a NT1
NT2 → * a
NT3 → - a
NT4 → + a
NT5 → NT4 .  | NT5
NT6 → NT5 a | NT6 a
NT7 → .  a
NT8 → NT2 | NT3 | NT6 | NT7
```

A generalização através do processo de repetição de não-terminal resultou na seguinte gramática:

```
NT1 → a NT8
NT2 → * a
NT3 → - a
NT4 → + a
NT7 → .  a
NT8 → NT11 | NT12 | NT13 | NT14
NT5 → NT9
NT9 → NT9 NT4 | NT4
NT6 → NT10 a
NT10 → NT10 NT5 | NT5
NT11 → NT11 NT2 | NT2
NT12 → NT12 NT3 | NT3
NT13 → NT13 NT6 | NT6
NT14 → NT14 NT7 | NT7
```

Na gramática gerada pelo processo de substituição simples, a cadeia “a a - a”, inicialmente não aceita, passa a ser reconhecida.

Em nenhum dos exemplos positivos duas letras “a”s” aparecem consecutivas. Caso o teste seja repetido com o mesmo conjunto de exemplos positivos e esta cadeia adicionada ao conjunto de exemplos negativos, obtém-se a seguinte gramática através do processo de substituição simples:

```
NT1 → a NT8
NT2 → * a
NT3 → - a
NT4 → + a
NT5 → NT4 .  | NT5
NT6 → NT5 a | NT6 a
NT7 → .  a
NT8 → NT2 | NT3 | NT6 | NT7
```

4.2 Teste 02

Considere-se o conjunto de exemplos positivos V, definido por:

V = {“c [a ; a ; a ; i B [w B [a ; a ;]] e [a ; a ;]”;
“c [a ; a i B [w B [a ; a ;]] e [a ; a ;] ; a ;]”;
“c [a ; w B [a ; a ;] a ;]”; “c [a ; i B [a ; a ;] a ;]”;
“c [i B [a ;] a ;]”; “c [a ; w B [a ;]]”; “c [w B [a ; a ;]]”;
“c [a ; i B [a ; a ;] e [w B [a ;]] a ;]”; “c [i B [a ; a ;] a ;]”;

“c [a ; a ;]”; “c [i B [a ;]]”; “c [w B [a ;]]”;
“c [i B [a ;] e [a ;]]”; “c [a ;]”; “c [a ; w B [a ; a ;]]”;
“c [a ; a ; a ; a ;]”; “c [w B [i B [a ; a ;] e [i B[a ; a ;]] a ;]]”;
“c [i B [a ; a ;] e [a ; a ;]]”; “c [a ; i B [a ;] e [a ;] a ;]”;
“c [i B [a ; w B [a ; a ;]] e [a ; a ;]]”;
“c [i B [w B [a ; a ;]] e [w B [a ; a ;]]]”}

E o conjunto de exemplos negativos F, definido por:

F = {“c [a a ;]”; “c [i [a ;]]”; “c [w B [a ; a ;] e [a ; a ;]]”;
“c [i B [a ;]”; “c [e [a ; a ;] a ;]”;
“c [w B [i B [a ; a ;] e[i B[a ; a ;]]] a ;]]”}

Este conjunto de exemplos representa a estrutura básica de uma linguagem de programação. A linguagem que definiria este conjunto de exemplos é mais extensa do que a linguagem que define o exemplo 01, além de existir um maior número de exemplos positivos no conjunto V. O objetivo deste teste é demonstrar o comportamento do algoritmo com um conjunto maior de exemplos positivos e uma gramática mais complexa. Esta gramática trabalha com, basicamente, quatro estruturas distintas:

- Uma estrutura que representa um `if` seguido por um `else`.
- Uma estrutura que representa um `if` sem ser seguido por um `else`.
- Uma estrutura que representa um `while`.
- Os caracteres “`a ,`”, que representa outros padrões.

Após a execução do algoritmo até o passo anterior à generalização, a gramática gerada é mostrada a seguir. Considerou-se como valor de confiança igual a 1 e valor de suporte igual a 0,1.

Como este exemplo é mais extenso, as limitações do algoritmo ficam mais evidentes. O objetivo principal para este exemplo seria que a gramática reconhecesse as estruturas básicas de que é formado, e estas repetidas em qualquer ordem e em qualquer número de vezes. Entretanto, a gramática gerada não foi capaz de reconhecer este padrão.

Outros padrões foram desconsiderados pelo simples fato de não existirem outros paralelos no mesmo lugar, como ocorreu com uma regra de produção associada ao não-terminal `NT3`, onde toda uma estrutura de `if` ficou representada como se fosse um padrão único na cadeia, o que resultou na simples representação dele na regra de produção.

```
NT1 → c [ NT2 ]
NT2 → a ; NT3 | i B [ NT4 | w B [ NT5 a ; ]
NT3 → a NT6 a ; | w B [ a ; NT7 |
       i B [ a ; NT8 ] a ; | λ
NT4 → a ; NT9 | w B [ a ; a ; ] ]
       e [ w B [ a ; a ; ] ]
NT5 → a ; | i B [ a ; a ; ] e [ i B [ a ; a ; ] ] |
       λ
NT10 → a ;
NT6 → ; NT10 NT11 | i B [ w B [ NT10 NT10 ] ]
       e [ NT10 NT10 ] ; | λ
NT7 → NT10 ] NT12 | ]
NT8 → NT10 NT13 | ] e [ NT10
NT9 → NT10 ] NT14 | ] NT15 |
       w B [ NT10 NT10 ] ] e [ NT10 NT10 ]
NT16 → i B
NT17 → NT16 [
NT18 → NT17 w
NT19 → NT18 B
NT20 → NT19 [
NT21 → NT20 NT10
NT22 → NT21 NT10
NT23 → NT22 ]
NT24 → NT23 ]
NT25 → NT24 e
NT26 → NT25 [
NT27 → NT26 NT10
NT28 → NT10 ]
NT29 → e [
NT11 → NT27 | λ
NT12 → λ | NT10
NT13 → λ | ] NT29 w B [ NT28
NT14 → NT10 | NT29 NT10 NT28
NT15 → λ | NT10 | NT29 NT28
```

Para finalizar, a gramática cresceu de tal maneira que ambos os processos de generalização não produziram nenhuma nova gramática, fazendo com que esta gramática gerada aceite apenas os exemplos positivos, da mesma maneira que a primeira gramática gerada pelo algoritmo.

Entretanto, cabe ressaltar que alguns padrões foram encontrados na gramática, como é o caso do não-terminal `NT1`, que deriva para uma estrutura presente em todos os exemplos positivos.

5. Conclusões

Este artigo descreveu o desenvolvimento de um algoritmo que, a partir de um conjunto de cadeias de caracteres representando exemplos positivos de cadeias e um conjunto representando exemplos negativos, crie gramáticas que validem os exemplos positivos e rejeitem os exemplos negativos.

O algoritmo de *Data Mining* foi aplicado no projeto no reconhecimento de padrões relativos à ocorrêcia de dois ou mais símbolos em seqüência na cadeia com certa freqüência. O símbolos substituídos tornaram as regras de produção mais simples, simplificando também a busca de padrões por agrupamento pelo símbolo inicial.

A busca de padrões por agrupamento mostrou-se eficaz em encontrar padrões típicos de linguagens livres de contexto, tais como a repetição de símbolos no início e no fim de várias cadeias simultaneamente. Entretanto, em algumas situações esta busca teve desempenho abaixo do esperado, como no caso de identificar estruturas de repetição aleatórias nas cadeias, identificação de estruturas opcionais e no caso de estruturas típicas de linguagens regulares. Nestes casos específicos, as gramáticas geradas não criaram estruturas que permitissem o reconhecimentos dos padrões regulares. Além disso, nenhuma das regras de produção destas gramáticas pode ser expandida de maneira que fosse possível a generalização das mesmas.

O algoritmo se mostrou mais eficaz em linguagens mais simples do que em linguagens mais complexas e que envolvam muitos exemplos, como é o caso do conjunto de testes na seção 4.2. Em linguagens mais complexas, exige-se um número muito grande de exemplos positivos para que o algoritmo reconheça o padrão.

O fato do algoritmo não ser baseado em processos estocásticos permite que alguns parâmetros sejam estabelecidos de acordo com o desempenho da gramática final gerada, como é o caso dos valores de suporte e confiança. Além disso, o conjunto de exemplos negativos pode ser criado à medida em que o usuário descobre padrões indesejáveis aceito pelas gramáticas depois da generalização.

Como trabalhos futuros, pretende-se adicionar novos processos entre a aplicação do *Data Mining* e o agrupamento pelos símbolos iniciais da gramática, de maneira que os problemas encontrados sejam solucionados ou ao menos tenham o impacto reduzido. Um destes processos é o desenvolvimento ou a adaptação de estratégias que permitam o reconhecimento de padrões regulares nas gramáticas livres de contexto. Outro ponto a ser considerado é estabelecer de maneira automática os valores ótimos de suporte e confiança para a aplicação do *Data Mining*. Sobre a generalização, pretende-se estudar formas de minimizar o impacto da ambigüidade gerada no processo de inserção de recursões. Finalmente, pretende-se realizar conjuntos de testes clássicos, como os testes descritos em [6].

Referências

[1] A. Aho, R. Sethi, and J. D. Ullman. *Compiladores: Princípios, Técnicas e Ferramentas*. Editora: LTC, 1986.

[2] F. C. N. do Amaral. *Data Mining: Técnicas e Aplicações para o Marketing Direto*. Editora: Berkley Brasil, 1st edition, 2001.

[3] M. Mernik, M. Crepinsek, G. Gerlic, V. Zumer, B. R. Bryant, and H. Sprague. *Learning Context-Free Grammars Using an Evolutionary Approach*. Editora: Maribor, 2000.

[4] K. Nakamura. Incremental learning of context-free grammars by extended inductive cyk algorithm. In *Workshop and Tutorial at ECML/PKDD: Learning Context-Free Grammars*. 2003.

[5] R. Parekh and V. Honavar. Learning DFA from simple examples. In *Algorithmic Learning Theory, 8th International Workshop, ALT '97, Sendai, Japan, October 1997, Proceedings*, volume 1316, pages 116–131. Springer, 1997.

[6] C. B. Starkie, B. Starkie, F. Coste, and M. V. Zaanen. The omphalos context-free grammar learning. In *in Y Sakakibara (ed.), Grammatical Inference: Algorithms and Applications; 7th International Colloquium, ICGI 2004*, pages 16–27. Springer, 2004.

[7] S. M. Venske, S. B. Boss, M. A. Musicante, J. M. Hauagge, I. W. Soares, L. T. Agner, and A. M. Ré. Inductive inference of lr(o) grammars. In *Revista Hífen*, volume 30, pages 17–24. 2006.

[8] P. J. Wyard. Context free grammar induction using genetic algorithms. In R. K. Belew and L. B. Booker, editors, *ICGA*, pages 514–519. Morgan Kaufmann, 1991.