Capítulo 4

Inferência de Modelos Utilizando a Programação Imunológica Gramatical

Heder S. Bernardino* e Helio J. C. Barbosa

Resumo: Apresenta-se neste capítulo a programação imunológica gramatical, uma técnica para evolução de programas que combina um mecanismo de busca inspirado pela teoria da seleção clonal com a representação via evolução gramatical, que faz uma distinção clara entre o espaço de busca e o espaço de soluções, oferecendo portanto mais flexibilidade. A técnica é aplicada ao problema inverso de identificação de modelos – em forma simbólica – a partir de dados. Exemplos de inferência de sistemas de equações diferenciais ordinárias são apresentados.

Palavras-chave: Programação genética, Problemas inversos, Identificação de modelos, Programação imunológica gramatical.

Abstract: *In this chapter we present the grammar-based immune programming, a technique for evolving programs by combining a search mechanism, inspired by the clonal selection theory, with the grammatical evolution representation which makes a clear distinction between the search and the solution spaces, thus offering more flexibility. The technique is applied to the inverse problem of model identification – in symbolic form – from data. Examples of the inference of systems of ordinary differential equations are presented.*

Keywords: *Genetic programming, Inverse problems, Model identification, Grammar-based immune programming.*

Conteúdo

1 Introdução 38
2 Inferência Automática de Modelos 38
2.1 Identificação paramétrica 38
2.2 Regressão simbólica e identificação estrutural 38
2.3 Inferência de modelos evolutivos 39
3 Evolução Gramatical 39
3.1 Gramática formal 39
3.2 Evolução gramatical 40
3.2.1 Coeficientes numéricos 41
3.3 Avaliação das soluções candidatas 42
4 Programação Imunológica Gramatical 42
4.1 CLONALG 43
4.2 Sistemas imunológicos para evolução de programas 44
4.3 Programação imunológica gramatical 44
5 Experimentos Computacionais 44
5.1 Modelo de reação química 46
5.2 Modelo da equação de fertilidade para dois alelos 46
5.3 Modelo de rede de regulação gênica 47
6 Conclusão 49

*Autor para contato: heder@ice.ufjf.br

 DOI: 10.7436/2014.tica.04 ISBN 978-85-64619-15-9

1. Introdução

Problemas inversos ocorrem nas mais variadas áreas da ciência e engenharia. Entretanto, simplificações são frequentemente introduzidas no problema de forma a torná-lo tratável. Por exemplo, a estrutura do modelo pode ser pré-especificada, restando assim a tarefa de determinar seus parâmetros numéricos. Todavia, mesmo com esse tipo de simplificação, o problema de otimização resultante pode ser complexo o suficiente de modo que técnicas numéricas tradicionais não sejam aplicáveis. Nesse tipo de situação, as metaheurísticas em geral, e os algoritmos genéticos em particular, parecem ser alternativas mais adequadas (Mera et al., 2002).

Explora-se nesse capítulo a possibilidade de se ir além da identificação paramétrica. A idéia aqui é apoiar o analista na identificação, a partir de dados observados, não só de parâmetros mas também da estrutura do modelo. As técnicas de inteligência computacional, em particular as de programação genética (Koza, 1992), têm se mostrado promissoras nesta área (Schmidt & Lipson, 2009). O presente trabalho descreve o algoritmo imuno-inspirado para a evolução de programas, guiado por gramáticas formais, denominado Programação Imunológica Gramatical (Bernardino & Barbosa, 2009, 2010a,b, 2011; Bernardino et al., 2011; Bernardino, 2012). Este método é baseado na técnica de seleção clonal conhecida como CLONALG e adota o mapeamento da evolução gramatical (O'Neill & Ryan, 2001, 2003), em que soluções candidatas, codificadas por cadeias binárias, são transformadas em programas utilizando-se uma gramática formal.

A programação imunológica gramatical é aplicada aqui a problemas inversos que visam a identificação de sistemas de equações diferenciais ordinárias –em forma simbólica– a partir de dados. Experimentos computacionais com problemas da literatura são apresentados.

2. Inferência Automática de Modelos

2.1 Identificação paramétrica

Simplificações são muitas vezes introduzidas no problema de encontrar um modelo que melhor se ajuste a um conjunto de observações de modo a torná-lo tratável. Na identificação dita *paramétrica*, a estrutura do modelo é pre-especificada pelo analista; por exemplo, $f(\mathbf{x}) = g(\mathbf{x}, \mathbf{a})$, onde apenas alguns coeficientes $a_1, a_2, \ldots, a_n$ são variáveis a serem ajustadas, ou seja, o espaço de busca reduz-se a $\mathbb{R}^n$. Pode-se então estipular uma medida de erro a ser minimizada, como

$$\Psi(\mathbf{a}) = \left(\sum_{i=1}^{m} |y_i - g(\mathbf{x_i}, a_1, \ldots, a_n)|^p \right)^{1/p}, \tag{1}$$

onde y_i é o valor observado (podendo ser multidimensional) para o registro $\mathbf{x_i}$, $i = 1, \ldots, m$, dos dados fornecidos e $p \geq 1$ um parâmetro.

Observe-se que, ao buscar um modelo que reproduza/explique um conjunto de dados, o que se deseja na realidade é que este seja capaz de oferecer também boas aproximações para novos pontos apresentados. Portanto, além de um modelo acurado em relação aos dados ditos de treinamento, deseja-se encontrar aquele que generalize para outros casos.

Finalmente, nota-se o papel fundamental do especialista nesse processo, por causa da sua responsabilidade na escolha da estrutura do modelo. Modelos com baixa acurácia ou que superajustem os dados podem ser resultados de escolhas inadequadas na sua forma.

2.2 Regressão simbólica e identificação estrutural

Pode-se esperar mais da inferência de modelos, de modo que sua capacidade não esteja limitada à previsão de resultados sobre novos dados, mas que também possibilite a geração de conhecimento sobre o fenômeno modelado. Isto ocorre na identificação dita *estrutural*, onde a forma do modelo não é pré-especificada. A consequência óbvia é o aumento no "tamanho" e na complexidade do conjunto das soluções candidatas do problema.

A inteligência computacional (IC) vem então fornecer ferramentas para a inferência automática de modelos a partir de dados observados de determinado fenômeno. A expressão *regressão simbólica* passa a ser utilizada nos casos em que a estrutura do modelo não é pré-definida pelo analista, mas sim construída, combinando-se estruturas de um conjunto de primitivas. A forma mais comum da regressão simbólica corresponde ao problema de se encontrar a expressão algébrica $f(x_1, x_2, \ldots, x_n)$ que melhor relaciona a quantidade de interesse, y, a um conjunto de variáveis independentes observadas $x_1, x_2, \ldots, x_n$.

A Programação Genética (PG) é a ferramenta de IC capaz de resolver o problema de regressão simbólica, agregando também conhecimento acerca do fenômeno observado. Como será visto, a PG pode fornecer conhecimento de forma bem mais legível, entendível e comunicável; esse tipo de informação é incomum em outras técnicas de aprendizado de máquina (tais como as redes neuronais artificiais).

2.3 Inferência de modelos evolutivos

Um conjunto de equações diferenciais ordinárias (EDOs) é uma forma de descrever matematicamente comportamentos que variam ao longo do tempo nas mais variadas áreas da ciência e das engenharias. Neste capítulo, há o interesse em inferir um modelo evolutivo na forma de uma EDO, ou seja, encontrar $f(x, y)$ em forma explícita tal que $y'(x) = f(x, y)$ seja o que melhor representa os dados observados (x_i, y_i), com $i = 1, \ldots, m$.

A derivação numericamente dos dados observados para determinar um conjunto de aproximações $\bar{y}'_i \approx y'_i$ é uma forma possível de tratamento para esse problema. Uma expressão em diferença central requerendo apenas informações dos pontos (x_{i-1}, y_{i-1}) e (x_{i+1}, y_{i+1}) pode ser utilizada:

$$\bar{y}'_i = \frac{y_{i+1} - y_{i-i}}{x_{i+1} - x_{i-1}}. \tag{2}$$

Recai-se assim no caso usual da regressão simbólica.

Alternativamente, um modelo evolutivo também pode ser avaliado integrando numericamente a EDO $y' = f(x,y)$, que corresponde à solução candidata $f(x,y)$, resultando numa aproximação $\bar{y}$. Assim, a qualidade do modelo pode ser calculada comparando-se $\bar{y}$ com as observações y_i.

Mesmo a diferenciação numérica dos valores observados resultar em menor custo computacional, já que seu cálculo é realizado apenas uma vez, a integração dos modelos candidatos fornece mais precisos para os resultados. Considerando o método apresentado aqui, modelos mais acurados e generalizáveis são encontrados ao adotar a alternativa de integrar os candidatos (Bernardino & Barbosa, 2010a).

Finalmente, não é difícil perceber que pode-se abordar também o caso em que o modelo a ser inferido é um sistema de EDOs. O objetivo passa a ser encontrar as $f_i(x_1, x_2, \ldots, x_n)$, $i = 1, \ldots, n_{eq}$, tais que o sistema de EDOs $\mathbf{y}' = \mathbf{f}(x_1, x_2, \ldots, x_n)$ seja aquele que melhor se ajuste aos valores observados. A abordagem em sistemas de EDOs é importante também por permitir inferir EDOs de ordem superior.

3. Evolução Gramatical

A programação genética é uma poderosa ferramenta para geração automática de programas. Entretanto, os sub-elementos na representação por árvore devem ter o mesmo tipo, restringindo a PG tradicional e, possivelmente, dificultando sua aplicação à problemas reais. Essa limitação pode ser resolvida adotando-se a programação genética fortemente tipada, em que as operações sobre as árvores podem ocorrer desde de que suas sub-estruturas concordem não somente em número, mas também em relação aos tipos de dados dos parâmetros de entrada e saída. Todavia, percebe-se que restrições mais complexas podem ser úteis para o processo de busca, especialmente nos casos em que a introdução de conhecimento prévio sobre o problema diminua convenientemente o espaço de busca e, portanto, aumente as chances de sucesso do método.

A sintaxe da linguagem em que é permitido que os programas evoluam pode ser restringida, de modo a aumentar o controle sobre o espaço de busca. Gramáticas formais podem ser utilizadas para este fim, impedindo que programas ditos inválidos sejam gerados e aumentando a capacidade da PG de evoluir estruturas complexas (um código-fonte, por exemplo), agora delimitado pela sintaxe da linguagem de programação requerida.

3.1 Gramática formal

Pode-se definir uma gramática formal G como (Chomsky, 2002)

$$G = \{N, \Sigma, R, S\}, \tag{3}$$

onde N é um conjunto finito de não-terminais, Σ é um conjunto finito de terminais, R é um conjunto finito de regras (ou produções) e $S \in N$ é o símbolo inicial. Enquanto componentes auxiliares da gramática formam N, Σ é composto por símbolos que podem aparecer na linguagem: em expressões aritméticas, x, sin, 1, e $+$ seriam exemplos de elementos de Σ. Finalmente, no que se refere às

$$
\begin{aligned}
G =& \{N, \Sigma, R, S\} \\
N =& \{\text{<expr>}, \text{<op>}, \text{<uop>}, \text{<var>}\} \\
\Sigma =& \{\sin, \cos, \log, \exp, \sqrt{}, x, y, 1, 2, 3, +, -, \times, \div, \text{pow}, (,)\} \\
S =& \text{<expr>} \\
R:& \\
\text{<expr>} ::=& (\text{<expr>}\ \text{<op>}\ \text{<expr>}) \mid \text{<uop>}\ (\text{<expr>}) \mid \text{<var>} \\
\text{<op>} ::=& + \mid - \mid \times \mid \div \mid \text{pow} \\
\text{<uop>} ::=& \sin \mid \cos \mid \log \mid \exp \mid \sqrt{} \\
\text{<var>} ::=& x \mid y \mid 1 \mid 2 \mid 3
\end{aligned}
$$

Figura 1. Exemplo de gramática formal para expressões aritméticas.

gramáticas livres de contexto (normalmente usadas na evolução de programas), as regras R na forma de Backus-Naur (BNF) podem ser expressas como

$$\text{<nao\_terminal>} ::= \text{expressao}, \tag{4}$$

onde <nao_terminal> $\in N$, expressao $= (\Sigma \cup N)^*$ e * é a estrela de Kleene, um operador que permite a formação de palavras compostas pelos elementos do conjunto de referência. Assim, os não-terminais podem ser expandindos em um ou mais terminais e não-terminais bem como auto-referenciar-se, gerando uma recursão. Na ocorrência de várias sequências em "expressao", as possíveis escolhas são delimitadas pelo símbolo "|".

A Figura 1 apresenta um exemplo de gramática livre de contexto para definir expressões algébricas simples. Algumas possíveis expressões aritméticas representáveis por essa gramática são $((y-3)\times x)$ e $\cos(y+x)$. Vale lembrar que, embora expressões como $\tan(x+3)$ não possam ser representadas pela gramática, pois a sequência de símbolos tan não aparece nos terminais, a expressão $(\sin(x+3)/\cos(x+3))$ é perfeitamente válida, sendo a função seno representada por sin.

3.2 Evolução gramatical

Apesar da adição de gramáticas formais gerar mais generalidade, expressividade e aplicabilidade na PG, é possível perceber um aumento na complexidade de sua implementação e uso. Nota-se também que o método de otimização é limitado a procedimentos similares à PG (ou AG), já que não há distinção entre genótipo e fenótipo nas soluções candidatas. A Evolução Gramatical (EG), proposta por Ryan et al. (1998) e estendida por O'Neill & Ryan (2001, 2003), foi criada como uma forma de desconectar a representação gênica do programa em si, mas mantendo a capacidade restritiva das gramáticas formais.

Na EG, o processo de mapeamento genótipo-fenótipo pode ser decomposto em duas etapas: (i) decodificação de uma lista de binários em um vetor de números inteiros; e (ii) criação de um programa, por meio de uma gramática formal, utilizando o vetor de inteiros previamente decodificado.

Dessa forma, a lista binária é uma representação de um vetor de inteiros codificado utilizando b *bits* (usualmente, $b = 8$). Na realidade, a primeira etapa pode ser suprimida utilizando-se diretamente uma representação inteira das soluções candidatas (Hugosson et al., 2007, 2010).

Um programa pode então ser criado por meio de uma gramática formal (definida pela Equação 3) a partir do vetor de números inteiros. Cada passo desse processo é realizado selecionando-se uma regra no conjunto de produções da gramática através da seguinte expressão:

$$regra = i_p \mod (n_r),$$

onde i_p é o próximo número inteiro na sequência e n_r é o número de regras do não-terminal corrente (o primeiro não-terminal observado numa verificação da esquerda para a direita).

Um exemplo ilustrativo do processo é apresentado a seguir (extraído de (Bernardino, 2012)). A gramática G mostrada na Seção 3.1 será adotada mas suas regras de produção R são re-apresentadas e em outro formato para facilitar o entendimento do processo de mapeamento. Cada uma das produções em R foi colocada em uma linha e à direita dessas são apresentados seus índices em relação ao não-terminal derivador. Assim, as regras de produção R ficam como na Figura 2.

$$
\begin{aligned}
\text{<expr>} &::= (\text{<expr>}\ \text{<op>}\ \text{<expr>}) && (0)\\
&\mid \text{<uop>}\ (\text{<expr>}) && (1)\\
&\mid \text{<var>} && (2)\\
\text{<op>} &::= + && (0)\\
&\mid - && (1)\\
&\mid \times && (2)\\
&\mid \div && (3)\\
&\mid \text{pow} && (4)\\
\text{<uop>} &::= \text{sin} && (0)\\
&\mid \text{cos} && (1)\\
&\mid \text{log} && (2)\\
&\mid \text{exp} && (3)\\
&\mid \sqrt{} && (4)\\
\text{<var>} &::= x && (0)\\
&\mid y && (1)\\
&\mid 1 && (2)\\
&\mid 2 && (3)\\
&\mid 3 && (4).
\end{aligned}
$$

Figura 2. Regras de produção R da gramática ilustrativa.

Nas Figuras 3 e 4, vê-se que a entrada no primeiro passo é o não-terminal inicial definido na gramática corrente, $S =$<expr>. De posse do próximo inteiro (33), verificando-se que o não-terminal base (<expr>) possui 3 possibilidades de derivação e sendo 33 mod (3) = 0, então (<expr> <op> <expr>) é selecionado para substituir o não terminal base, por ser a possibilidade zero dentre as regras de <expr>. O resultado é criado trocando-se o não-terminal base (<expr>) pela produção escolhida na derivação ((<expr> <op> <expr>)). O resultado de um passo é utilizado como entrada para o passo seguinte. Sendo a base o primeiro não-terminal à esquerda encontrado na entrada então, neste caso, <expr> é o não-terminal base no segundo passo. Como o próximo inteiro é 29, sendo 3 as possibilidades de derivação do não-terminal base (<expr>) e 29 mod (3) = 2, então <var> é selecionado para substituir o não terminal base. O procedimento continua enquanto houver não-terminais no resultado; caso contrário o programa estará completo. Seguindo esse processo, o exemplo conclui gerando o programa $x + (\exp(y))$.

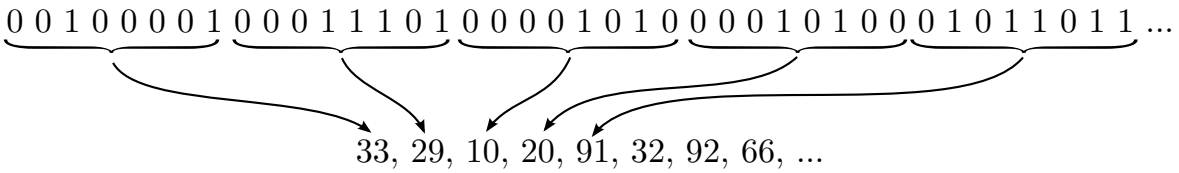

Figura 3. Exemplo de decodificação de um genótipo num vetor de inteiros.

Percebe-se que não é necessário utilizar todos os valores inteiros disponíveis. Apesar disso não gerar complicações, uma outra ocorrência precisa ser tratada: o mapeamento pode esgotar os números inteiros antes que um programa completo seja criado. Uma proposta seria reutilizar os valores inteiros (*wrap-around*) a partir do início do cromossomo (O'Neill & Ryan, 2001). A alternativa adotada aqui é reparar o programa durante o mapeamento, sempre gerando soluções válidas, já que a reutilização de valores não resolve completamente o problema (Bernardino & Barbosa, 2009).

3.2.1 Coeficientes numéricos

Vários programas incluem coeficientes numéricos entre seus componentes, tornando seu processo de criação muito importante para a geração desse tipo de estrutura. A utilização de uma gramática formal incorpora à EG diversas possibilidades de se tratar essa situação. Aparentemente não há consenso sobre qual a técnica mais adequada, apesar de alguns trabalhos analisarem várias das

passo	entrada	regra	resultado
1	<expr>	33 mod (3) = 0	(<expr> <op> <expr>)
2	(<expr> <op> <expr>)	29 mod (3) = 2	(<var> <op> <expr>)
3	(<var> <op> <expr>)	10 mod (5) = 0	(x <op> <expr>)
4	(x <op> <expr>)	20 mod (5) = 0	(x + <expr>)
5	(x + <expr>)	91 mod (3) = 1	(x + <uop>(<expr>))
6	(x + <uop>(<expr>))	32 mod (5) = 2	(x + exp(<expr>))
7	(x + exp(<expr>))	92 mod (3) = 2	(x + exp(<var>))
8	(x + exp(<var>))	66 mod (5) = 1	(x + exp(y))

Figura 4. Ilustração do mapeamento do vetor da Figura 3 num programa.

alternativas que existem atualmente (Augusto et al., 2011). Dois métodos são concomitantemente adotados aqui: o tradicional e o dos mínimos quadrados.

No método tradicional (O'Neill & Ryan, 2001), os coeficientes numéricos são criados através de operações aritméticas sobre os valores inteiros pré-fixados. Por exemplo, pode-se gerar o coeficiente real 2,5 com a expressão (5 ÷ 2).

Outra forma de ajustar os coeficientes numéricos na EG é adotar uma gramática que gere funções-base sobre as quais aplica-se o método dos mínimos quadrados (Bernardino & Barbosa, 2011). Neste caso, a gramática deve ser adaptada para mapear uma combinação linear de expressões. Por exemplo, através da regra de produção

$$\text{<base>} ::= (\text{<base>} \ + \ \text{<base>}) \mid \text{<expr>}$$

e indicando <base> como o não-terminal inicial (S =<base>). Ao avaliar um dado programa, as bases são executadas sobre os dados, formando uma matriz X que é então utilizada para determinar o vetor de coeficientes $\mathbf{a}$ que minimiza o erro quadrático médio em relação aos valores observados $\mathbf{y}$. Apenas os coeficientes lineares são ajustados por meio dessa técnica. As demais constantes são geradas aqui por meio do método tradicional.

3.3 Avaliação das soluções candidatas

A qualidade com a qual os programas candidatos realizam uma determinada tarefa é medida na etapa de avaliação dos indivíduos. Para isso, cada programa candidato é executado sobre os conjuntos de argumentos de entrada e os valores de retorno são comparados ao que se é esperado (conhecido). Dessa forma, uma função objetivo (a ser minimizada) é a média dos erros ao quadrado, ou seja,

$$f(\text{ programa }) = \frac{1}{m} \sum_{i=1}^{m} [\text{programa}(\mathbf{p_i}) - y_i]^2 \,. \tag{5}$$

onde m é o número de pontos, $\mathbf{p_i} \in \mathbb{R}^n$ é o i-ésimo ponto, "programa" é a função/programa codificada por um indivíduo e $y_i \in \mathbb{R}$ é o valor esperado para o i-ésimo ponto.

Como já mencionado na Seção 2.3, existem duas formas de adequar a Equação 5: (i) os valores esperados ($\mathbf{y}$) são definidos como a diferenciação dos valores observados; (ii) a execução do "programa" resulta na integração do modelo dinâmico.

Como em todo processo de aprendizado automático, corre-se o risco de se super-ajustar o modelo aos dados e, dessa forma, duas direções de busca conflitantes ficam evidentes: (i) maximizar a acurácia, ou seja, minimizar a discrepância em relação ao que é esperado; e (ii) minimizar a complexidade do modelo, assumindo que maior complexidade está geralmente ligada à ocorrência de super-ajuste. Nota-se inclusive, que o segundo item também está relacionado à facilidade de interpretabilidade do modelo. Além disso, é importante observar que a forma como os programas são mapeados na evolução gramatical gera um viés na direção de aumentar a probabilidade de criação de programas menores.

4. Programação Imunológica Gramatical

A evolução gramatical dissocia o espaço de busca (binário, inteiro, etc) do de programas, permitindo que diferentes algoritmos de busca possam ser utilizados na criação de estruturas complexas em uma linguagem arbitraria. Apesar de originalmente adotada em conjunto com os algoritmos

genéticos (Ryan et al., 1998), a técnica adotada aqui utiliza um algoritmo imuno-inspirado combinado com ideias da EG para evoluir programas (Bernardino & Barbosa, 2009, 2010b).

4.1 CLONALG

O CLONALG (*CLOnal selection ALGorithm*, ou algoritmo de seleção clonal) foi proposto por Castro & von Zuben (2000, 2002) e imita o princípio de seleção clonal (Burnet, 1957), de modo que as soluções candidatas são melhoradas seguindo as etapas de clonagem dessas soluções, hipermutação das novas células geradas e seleção daquelas com maior afinidade em relação ao antígeno, ou seja, as melhores soluções em relação ao objetivo. É importante ressaltar que as soluções candidatas são representadas por anticorpos (para simplificar, os linfócitos não são considerados) e que a exploração do espaço de busca é feita através mutações aplicadas com uma taxa elevada.

Um pseudo-código do CLONALG é apresentado no Algoritmo 1 (extraído de (Bernardino, 2012)). Seus parâmetros de entrada são: um valor β que define o número de clones gerados por cada anticorpo (veja a Equação 6), um fator ρ responsável por ajustar a intensidade da hipermutação que será aplicada aos clones (veja a Equação 7), o tamanho *tamanhoPopulacao* da população de soluções candidatas *anticorpos*, a quantidade *nAleatorio* de anticorpos de menor afinidade que devem ser substituídos por novas soluções aleatoriamente geradas, e o número *nSelecao* de anticorpos a serem selecionados para a clonagem e re-selecionados na atualização da população corrente. O valor de *nAleatorio* pode ser tomado como uma fração do tamanho da população, apesar de ter sido inicialmente proposto como um parâmetro absoluto.

Algoritmo 1: Pseudo-código do CLONALG.

Entrada: β, ρ, *tamanhoPopulacao*, *nAleatorio*, *nSelecao*
Saída: *anticorpos*

```
1  inicio
2      anticorpos ⟵ inicializaPopulacao(tamanhoPopulacao);
3      afinidades ⟵ avalia(anticorpos);
4      enquanto critério de parada não é atingido faça
5          afinidadesNormalizadas ⟵ normaliza(afinidades);
6          selecionados ⟵ seleciona(anticorpos, afinidadesNormalizadas, nSelecao);
7          clones ⟵ clona(selecionados, afinidadesNormalizadas, β);
8          hipermuta(clones, afinidadesNormalizadas, ρ);
9          afinidadesClones ⟵ avalia(clones);
10         substituiPopulacao(anticorpos, afinidades, clones, afinidadesClones, nSelecao,
           nAleatorio);
```

O pseudo-código utiliza também as funções que seguem: "inicializaPopulacao", responsável por gerar uma nova população de soluções candidatas (normalmente isso é feito aleatoriamente); "avalia", avalia as soluções em relação à função objetivo; "normalize", normaliza as afinidades em $[0; 1]$; "seleciona", seleciona os anticorpos que serão clonados; "clona", clona as soluções candidatas; "hipermuta", aplica o operador de modificação (hipermutação) nos clones gerados; "substituiPopulacao", atualiza a população de anticorpos pela substituição dos piores *nSelection* anticorpos pelos melhores clones gerados e gera *nRandom* novas soluções candidatas aleatoriamente também substituindo-as na população (note que cálculos adicionais da função objetivo são necessários aqui).

O número de clones n_{clones} gerados pelo anticorpo $anticorpo_i$ é definido por

$$n_{clones}(anticorpo_i) = arredonda\left(\frac{\beta.|anticorpos|}{i}\right), \tag{6}$$

onde "arredonda" é um operador que retorna o valor inteiro mais próximo do seu argumento e i é a posição do anticorpo $anticorpo_i$ na lista ordenada (de forma decrescente) pela afinidade normalizada; dessa forma, o primeiro anticorpo da lista corresponde à melhor solução candidata. Segundo Castro & von Zuben (2002), a hipermutação é aplicada com uma taxa α proporcional à afinidade do anticorpo que deu origem ao clone $clone_i$, de forma que

$$\alpha(clone_i) = \exp\left(-\rho.\bar{f}_i\right), \tag{7}$$

com $\bar{f}_i$ sendo a afinidade normalizada do anticorpo i correspondente.

Uma vantagem do CLONALG é a sua adaptabilidade a diversos tipos de problemas. Um algoritmo evolutivo, por exemplo, precisa ser equipado com alguma técnica de nicho quando se deseja manter a diversidade da população, enquanto que no CLONALG apenas precisa-se fazer (Castro & von Zuben, 2002):

- $nSelection = |antibodies|$, ou seja, todos os anticorpos da população são selecionados para o processo de clonagem;
- n_{clones} igual para todas as soluções candidatas;
- a população passa a ser atualizada substituindo-se os anticorpos pelo seu melhor clone, quando uma melhora ocorre.

O CLONALG destaca-se principalmente pela manutenção da diversidade de suas soluções candidatas, facilitando a adaptação (importante em problemas dinâmicos), permitindo uma larga exploração do espaço e disponibilizando um conjunto de soluções ao concluir a busca (relevante no caso de problemas multi-modais).

4.2 Sistemas imunológicos para evolução de programas

As técnicas imuno-inspiradas não vêm sendo muito exploradas como mecanismo de busca para programas, apesar da popularização das técnicas de PG, especialmente a partir do estudo de Koza (1992). Pode-se citar Johnson (2003), que propôs um algoritmo imuno-inspirado para resolver problemas de regressão simbólica em que as soluções eram representadas por árvores e, mais recentemente, Musilek et al. (2006), que elaboraram a chamada Programação Imunológica, posteriormente utilizada por Lau & Musilek (2009). Outras variantes são a chamada programação imunológica elitista, proposta por Ciccazzo et al. (2008) e a programação por seleção clonal proposta por Gan et al. (2009a,b), em que as soluções candidatas são representadas linearmente. Posteriormente, Gan et al. (2010) estenderam a ideia para a representação via grafos. Finalmente, assim como a PIG, a técnica proposta por McKinney & Tian (2008) utiliza gramáticas formais para apoiar a evolução de programas, diferentemente dos outros métodos imuno-inspirados apresentados, que baseiam-se na forma tradicional da PG. É importante destacar que no método proposto por McKinney & Tian (2008), não há separação entre o espaço de busca e o de soluções.

4.3 Programação imunológica gramatical

A evolução de programas auxiliada por gramáticas formais tem se mostrado uma opção interessante na obtenção de bons resultados, principalmente pela facilidade em conectar a experiência do especialista à busca. É possível observar também que o complexo espaço de busca de programas possui características que tornam o CLONALG uma boa alternativa. Em especial, pode-se destacar que o CLONALG oferece as seguintes vantagens na resolução dessa classe de problemas (Bernardino, 2012): (i) adequação a espaços de busca complexos; (ii) disponibilização de um variado conjunto de soluções ao final da busca; (iii) natural navegação em espaços de busca discretos; e (iv) maior capacidade de paralelização, dada a reduzida quantidade de troca de informação entre as soluções. É possível verificar também que, assim como outras metaheurísticas, o CLONALG requer muitos cálculos da função objetivo, o que resulta num alto custo computacional. Todavia, assim como em outras técnicas populacionais, essa desvantagem pode ser aliviada com o uso de computação paralela.

Assim, a programação imunológica gramatical (Bernardino & Barbosa, 2009, 2010b), ou simplesmente PIG, é um método que codifica as soluções candidatas em código de Gray, utiliza o CLONALG como mecanismo de busca e adota o mapeamento da EG para gerar programas por meio de uma gramática formal. O pseudo-código para a PIG é o mesmo apresentado anteriormente para o CLONALG no Algoritmo 1. A diferença reside na avaliação da solução candidata realizada pelo procedimento "avalia" que, nesse caso, engloba o mapeamento da solução candidata para um programa bem como a avaliação de sua qualidade.

5. Experimentos Computacionais

Nessa seção, três problemas encontrados na literatura são utilizados para averiguar a capacidade da PIG em resolver problemas inversos encontrados na literatura. Busca-se com os experimentos computacionais avaliar o desempenho da PIG na inferência de modelos na forma de um sistema de EDOs a partir de um conjunto de dados observados. Os dados $(x_k, \mathbf{y_k})$, $k \in \{1, \ldots, m\}$

dos experimentos são artificialmente gerados pela integração numérica do modelo alvo, com x_k igualmente espaçado em $[x_1, x_m]$. Ruídos foram introduzidos aos dados de modo que cada valor y_{ik} foi alterado fazendo-se $y_{ik} = y_{ik} \times (r_g(0,1) \times p_{ruido} + 1)$, onde $r_g(0,1)$ é um número aleatoriamente gerado com distribuição gaussiana com média zero e desvio padrão unitário e p_{ruido} é a intensidade do ruído. Nos experimentos foram adotados $p_{ruido} = 0.01$, 0.03 e 0.05. As soluções candidatas são avaliadas comparando-se os dados do problema com os valores calculados pela integração numérica dos modelos candidatos utilizando o método de Runge-Kutta de quarta ordem com passo $h = (b-a)/(m-1)$.

De forma geral, a mesma gramática é utilizada para todos os problemas de inferência de modelos em forma de sistemas de EDOs. Para cada equação do sistema, as regras de formação R das expressões são dadas por

$$\begin{aligned} \text{<base>} &::= (+ \text{ <base> } \text{<base>}) \mid \text{<expr>} \\ \text{<expr>} &::= (\text{<op>} \text{ <expr> } \text{<expr>}) \mid \text{<var>} \\ \text{<op>} &::= + \mid - \mid \times \\ \text{<var>} &::= 1 \mid y_1 \mid y_2 \mid y_3. \end{aligned}$$

Nota-se que as regras de produção previamente apresentadas são suficientes para gerar apenas uma EDO, logo ne mapeamentos são necessários para criar o sistema desejado, onde ne é o número de equações do modelo. Além disso, foi definido que cada linha mapeia uma expressão do sistema, ou seja, as soluções candidatas são matrizes binárias (Bernardino & Barbosa, 2011). As gramáticas formais adotadas aqui são as mesmas definidas por Bernardino & Barbosa (2011).

Verifica-se também que a gramática gera programas na forma de uma combinação linear de bases e os coeficientes lineares dos modelos devem ser ajustados através do método dos mínimos quadrados. Para isso, os dados de entrada (com ruído) são diferenciados numericamente (Equação 2) e esse resultado compõe o sistema linear que completa a definição do modelo quando resolvido.

As regras de reparo R_r também precisam ser especificadas e foram definidas como

$$\begin{aligned} (+ \text{ <base> } \text{<base>}) &::= \text{<base>} \mid 1 \mid y_1 \mid y_2 \mid y_3 \\ (\text{<op>} \text{ <expr> } \text{<expr>}) &::= \text{<expr>} \mid 1 \mid y_1 \mid y_2 \mid y_3 \\ \text{<base>} &::= 1 \mid y_1 \mid y_2 \mid y_3 \\ \text{<expr>} &::= 1 \mid y_1 \mid y_2 \mid y_3 \end{aligned}$$

Os parâmetros do algoritmo de busca utilizados aqui são similares aos atribuídos em experimentos anteriores: tamanho da população= 50, tamanho do vetor de inteiros para cada equação do sistema= 100, $n_{clones} = 1$, $\rho = 5$ e $nAleatorio = 10\%$ da população. Cada experimento foi composto por 30 execuções independentes onde as sementes do gerador de números pseudoaleatórios foram inteiros numa sequência a partir do zero. Assim como em Bernardino & Barbosa (2011), um orçamento de 100.000 cálculos da função objetivo foi adotado como critério de parada.

Além do erro de treinamento, que é computado nos limites $[x_1, x_m]$, também são fornecidos os erros de teste, calculado sobre 11 pontos igualmente espaçados no intervalo estendido $E = [x_m, x_m + 11 \times h]$, onde h é o espaçamento entre os pontos. É importante destacar que os dados em E não são utilizados durante a evolução e, assim, o erro de teste funciona como um indicador da capacidade de generalização do modelo. Para os casos em que há diferentes condições iniciais, um único modelo que atenda a todas elas deve ser inferido; para isso, os modelos candidatos são avaliados considerando todos os dados de todos os valores iniciais.

Uma tabela apresentando o melhor valor de erro bem como sua mediana, média, desvio padrão (sd) e pior valor, tanto em relação aos dados de treinamento quanto de teste é apresentada para cada problema. Esses valores correspondem às soluções encontradas em todas as execuções independentes. Além disso, análises estatísticas não-paramétricas utilizando o Teste de Soma de Postos de Wilcoxon são realizados para verificar a significância das diferenças entre os resultados quando diferentes quantidades de ruído são introduzidas aos dados. A biblioteca SciPy [1] foi adotada para a realização destes destes.

Finalmente, a biblioteca JScheme, para computar as expressões em linguagem *scheme*, e uma biblioteca científica devida a Michael Thomas[2] foram incluídas na implementação.

[1] Disponível em: http://www.scipy.org.
[2] Disponível em: http://www.ee.ucl.ac.uk/~mflanaga/java

5.1 Modelo de reação química

Este problema tem como objetivo inferir um modelo que represente o comportamento das reações entre o hidrato de formaldeído ($HCHO$) e carbamida (ou ureia, $(NH_2)_2CO$) em solução aquosa produzindo metilol ureia ($H_2N \cdot CO \cdot NH \cdot CH_2OH$) e que, reagindo novamente com a carbamida forma metileno de ureia ($(NH_2CONH)_2CH_2$). Essa reação pode ser representada como

$$HCHO + (NH_2)_2CO \rightarrow H_2N \cdot CO \cdot NH \cdot CH_2OH$$
$$H_2N \cdot CO \cdot NH \cdot CH_2OH + (NH_2)_2CO \rightarrow (NH_2CONH)_2CH_2.$$

As concentrações de $HCHO$ (y_1), $H_2N \cdot CO \cdot NH \cdot CH_2OH$ (y_2) e $(NH_2CONH)_2CH_2$ (y_3), sob reações consecutivas, podem ser modeladas de maneira que satisfazem o sistema de EDOs (Cao et al., 2000)

$$\begin{cases} y_1' = -1{,}4y_1 \\ y_2' = 1{,}4y_1 - 4{,}2y_2 \\ y_3' = 4{,}2y_2 \end{cases}$$

Os dados de entrada são formados por 101 registros com $x \in [0, 1]$ e os valores de $\mathbf{y}$ gerados a partir da condição inicial $\mathbf{y}(0) = (0{,}1; 0; 0)$.

A Tabela 1 mostra a análise dos erros dos modelos encontrados para o problema corrente. Os modelos encontrados para representar a reação química apresentam valores médios de erro da ordem de 10^{-5}, para os dados de treinamento, e da ordem de 10^{-6}, em relação aos de teste. Destaca-se assim a capacidade de generalização dos modelos encontrados. Além disso, as análises estatísticas indicam que os modelos encontrados são adequados independente da quantidade de ruído introduzido nos dados, ou seja, dadas as intensidades de ruídos testadas, os modelos encontrados apresentam erros de treinamento e teste similares. Esse resultado ressalta a robustez da técnica de busca adotada aqui.

Tabela 1. Erros observados nos sistemas inferidos para o modelo de reação química.

p_{ruido}	**melhor**	**mediana**	**média**	**sd**	**pior**
			Erros de treinamento		
0,1	$2{,}59 \times 10^{-6}$	$5{,}03 \times 10^{-6}$	$1{,}44 \times 10^{-5}$	$1{,}76 \times 10^{-5}$	$6{,}72 \times 10^{-5}$
0,3	$9{,}63 \times 10^{-7}$	$4{,}66 \times 10^{-6}$	$1{,}29 \times 10^{-5}$	$2{,}87 \times 10^{-5}$	$1{,}61 \times 10^{-4}$
0,5	$3{,}77 \times 10^{-6}$	$5{,}70 \times 10^{-6}$	$1{,}09 \times 10^{-5}$	$9{,}53 \times 10^{-6}$	$3{,}73 \times 10^{-5}$
			Erros de teste		
0,1	$1{,}36 \times 10^{-6}$	$3{,}00 \times 10^{-6}$	$5{,}43 \times 10^{-6}$	$6{,}71 \times 10^{-6}$	$3{,}44 \times 10^{-5}$
0,3	$1{,}25 \times 10^{-6}$	$2{,}54 \times 10^{-6}$	$3{,}95 \times 10^{-6}$	$7{,}36 \times 10^{-6}$	$4{,}31 \times 10^{-5}$
0,5	$1{,}37 \times 10^{-6}$	$2{,}68 \times 10^{-6}$	$3{,}61 \times 10^{-6}$	$2{,}52 \times 10^{-6}$	$1{,}12 \times 10^{-5}$

5.2 Modelo da equação de fertilidade para dois alelos

Segundo Hofbauer & Sigmund (1998), ao denotar as frequências de ocorrência dos genótipos A_1A_1, A_1A_2 e A_2A_2, respectivamente, por y_1, y_2 e y_3 a equação de fertilidade para dois alelos, A_1 e A_2, pode ser modelada pelo sistema de EDOs

$$\begin{cases} y_1' = 2y_1^2 + 2{,}5y_1y_2 + 0{,}375y_2^2 - y_1P(y_1, y_2, y_3) \\ y_2' = 0{,}75y_2^2 + 2{,}5y_1y_2 + 2{,}5y_2y_3 + 3y_1y_3 - y_2P(y_1, y_2, y_3) \\ y_3' = 1{,}5y_3^2 + 2{,}5y_2y_3 + 0{,}375y_2^2 - y_3P(y_1, y_2, y_3) \end{cases}$$
$$P(y_1, y_2, y_3) = 2y_1^2 + 5y_1y_2 + 1{,}5y_2^2 + 3y_1y_3 + 5y_2y_3 + 1{,}5y_3^2.$$

Três sequências com 41 valores de $x \in [0, 10]$ são utilizadas na geração dos dados de entrada para a inferência deste modelo. As observações foram artificialmente geradas considerando as condições iniciais $\mathbf{y}(0) = (0{,}5, 0{,}5, 0)$, $(0{,}5, 0, 0{,}5)$ e $(0, 0{,}5, 0{,}5)$.

Os resultados para o problema abordado nessa seção podem ser encontrados na Tabela 2. Assim como no problema apresentado na seção anterior, os modelos encontrados aqui apresentam valores baixos de erro, tanto ao que se refere aos dados de treinamento quanto aos de teste. Esse resultado destaca novamente a capacidade de generalização dos modelos encontrados. Não obstante, segundo análises estatísticas, os modelos encontrados apresentam erros similares independente da quantidade de ruído introduzido nos dados, mostrando que a técnica é robusta em relação à qualidade dos dados.

Tabela 2. Erros observados nos sistemas inferidos para o modelo da equação de fertilidade para dois alelos.

p_{ruido}	**melhor**	**mediana**	**média**	**sd**	**pior**
			Erros de treinamento		
0,1	$1{,}63 \times 10^{-3}$	$4{,}16 \times 10^{-3}$	$4{,}29 \times 10^{-3}$	$1{,}36 \times 10^{-3}$	$8{,}27 \times 10^{-3}$
0,3	$2{,}76 \times 10^{-3}$	$3{,}87 \times 10^{-3}$	$4{,}33 \times 10^{-3}$	$1{,}36 \times 10^{-3}$	$8{,}00 \times 10^{-3}$
0,5	$2{,}03 \times 10^{-3}$	$4{,}46 \times 10^{-3}$	$4{,}67 \times 10^{-3}$	$1{,}33 \times 10^{-3}$	$8{,}04 \times 10^{-3}$
			Erros de teste		
0,1	$8{,}91 \times 10^{-4}$	$2{,}76 \times 10^{-3}$	$3{,}26 \times 10^{-3}$	$1{,}78 \times 10^{-3}$	$8{,}04 \times 10^{-3}$
0,3	$9{,}37 \times 10^{-4}$	$3{,}23 \times 10^{-3}$	$3{,}80 \times 10^{-3}$	$2{,}19 \times 10^{-3}$	$1{,}00 \times 10^{-2}$
0,5	$1{,}09 \times 10^{-3}$	$4{,}15 \times 10^{-3}$	$4{,}40 \times 10^{-3}$	$2{,}64 \times 10^{-3}$	$1{,}19 \times 10^{-2}$

5.3 Modelo de rede de regulação gênica

Tendo como objetivo principal identificar a relação entre os mecanismos regulatórios dos genes (Noman & Iba, 2005), a modelagem das redes de regulação gênicas, que são sistemas biológicos complexos, também pode ser feita por um sistema de equações diferencias ordinárias, representando a dinâmica da iteração entre os genes e os *S-systems*.

Aqui, o sistema de EDOs a ser inferido é o apresentado por Tominaga et al. (2000) para um caso particular composto por cinco genes:

$$\begin{cases} y_1' = 15y_3y_5^{-0,1} - 10y_1^2 \\ y_2' = 10y_1^2 - 10y_2^2 \\ y_3' = 10y_2^{-0,1} - 10y_2^{-0,1}y_3^2 \\ y_4' = 8y_1^2y_5^{-1} - 10y_4^2 \\ y_5' = 10y_4^2 - 10y_5^2 \end{cases}$$

Os dados são compostos por 31 valores em três sequências, sendo $x \in [0,0{,}3]$ e as condições iniciais dadas por $\mathbf{y}(0) = (0{,}1, 0{,}1, 0{,}1, 0{,}1, 0{,}1)$, $(0{,}5, 0{,}5, 0{,}5, 0{,}5, 0{,}5)$ e $(1{,}5, 1{,}5, 1{,}5, 1{,}5, 1{,}5)$.

A gramática adotada para esse problema foi modificada, assim como em Bernardino & Barbosa (2011). As regras de produção R e as produções de reparo R_r são apresentadas, respectivamente, nas Figuras 5 e 6.

<base> ::= (+ <base> <base>) | <expr>
<expr> ::= (<op> <expr> <expr>) | <var>
<op> ::= $+$ | $-$ | $\times$
<var> ::= $y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \mid y_1^{-1} \mid y_2^{-1} \mid y_3^{-1} \mid y_4^{-1} \mid y_5^{-1} \mid$
$y_1^{0,1} \mid y_2^{0,1} \mid y_3^{0,1} \mid y_4^{0,1} \mid y_5^{0,1} \mid y_1^{-0,1} \mid y_2^{-0,1} \mid y_3^{-0,1} \mid y_4^{-0,1} \mid y_5^{-0,1}$

Figura 5. Regras de produção R adotadas na inferência de um modelo para a rede de regulação gênica.

Os resultados para esse problema são apresentados na Tabela 3, que mostra o bom desempenho tanto sobre os dados de treinamento quanto ao que se refere aos dados de teste. Todavia, os erros de teste são uma ordem de grandeza pior do que aqueles obtidos sobre os dados de treinamento, o que pode indicar um superajuste dos modelos. Diferentemente do que ocorreu nos problemas anteriores, os modelos encontrados apresentam aqui erros de teste estatisticamente diferentes quando $p_{ruido} = 0{,}1$. Isso provavelmente se deve ao fato do pouco ruído contribuir para uma convergência mais rápida para um modelo adequado e, consequentemente, na sequência inicializar um processo de superajuste aos dados durante o treinamento. Verifica-se assim a necessidade de monitorar os erros de teste a cada geração afim de evitar essa perda de generalização dos modelos.

$$
\begin{aligned}
(+\ \text{<base>}\ \text{<base>}) ::= \ & \text{<base>} \mid y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \\
& \mid y_1^{-1} \mid y_2^{-1} \mid y_3^{-1} \mid y_4^{-1} \mid y_5^{-1} \\
& \mid y_1^{0,1} \mid y_2^{0,1} \mid y_3^{0,1} \mid y_4^{0,1} \mid y_5^{0,1} \\
& \mid y_1^{-0,1} \mid y_2^{-0,1} \mid y_3^{-0,1} \mid y_4^{-0,1} \mid y_5^{-0,1} \\
(\text{<op>}\ \text{<expr>}\ \text{<expr>}) ::= \ & \text{<expr>} \mid y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \\
& \mid y_1^{-1} \mid y_2^{-1} \mid y_3^{-1} \mid y_4^{-1} \mid y_5^{-1} \\
& \mid y_1^{0,1} \mid y_2^{0,1} \mid y_3^{0,1} \mid y_4^{0,1} \mid y_5^{0,1} \\
& \mid y_1^{-0,1} \mid y_2^{-0,1} \mid y_3^{-0,1} \mid y_4^{-0,1} \mid y_5^{-0,1} \\
\text{<base>} ::= \ & y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \\
& \mid y_1^{-1} \mid y_2^{-1} \mid y_3^{-1} \mid y_4^{-1} \mid y_5^{-1} \\
& \mid y_1^{0,1} \mid y_2^{0,1} \mid y_3^{0,1} \mid y_4^{0,1} \mid y_5^{0,1} \\
& \mid y_1^{-0,1} \mid y_2^{-0,1} \mid y_3^{-0,1} \mid y_4^{-0,1} \mid y_5^{-0,1} \\
\text{<expr>} ::= \ & y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \\
& \mid y_1^{-1} \mid y_2^{-1} \mid y_3^{-1} \mid y_4^{-1} \mid y_5^{-1} \\
& \mid y_1^{0,1} \mid y_2^{0,1} \mid y_3^{0,1} \mid y_4^{0,1} \mid y_5^{0,1} \\
& \mid y_1^{-0,1} \mid y_2^{-0,1} \mid y_3^{-0,1} \mid y_4^{-0,1} \mid y_5^{-0,1}
\end{aligned}
$$

Figura 6. Regras de reparo R_r adotadas na inferência de um modelo para a rede de regulação gênica.

Tabela 3. Erros observados nos sistemas inferidos para o modelo da rede de regulação gênica.

p_{ruido}	**melhor**	**mediana**	**média**	**sd**	**pior**
			Erros de treinamento		
0,1	$4{,}03 \times 10^{-2}$	$8{,}62 \times 10^{-2}$	$8{,}31 \times 10^{-2}$	$1{,}52 \times 10^{-2}$	$1{,}06 \times 10^{-1}$
0,3	$5{,}51 \times 10^{-2}$	$8{,}69 \times 10^{-2}$	$8{,}49 \times 10^{-2}$	$1{,}29 \times 10^{-2}$	$1{,}05 \times 10^{-1}$
0,5	$4{,}68 \times 10^{-2}$	$8{,}89 \times 10^{-2}$	$8{,}59 \times 10^{-2}$	$1{,}51 \times 10^{-2}$	$1{,}14 \times 10^{-1}$
			Erros de teste		
0,1	$1{,}32 \times 10^{-1}$	$2{,}38 \times 10^{-1}$	$2{,}21 \times 10^{-1}$	$3{,}81 \times 10^{-2}$	$2{,}61 \times 10^{-1}$
0,3	$7{,}32 \times 10^{-2}$	$1{,}78 \times 10^{-1}$	$1{,}78 \times 10^{-1}$	$5{,}88 \times 10^{-2}$	$2{,}61 \times 10^{-1}$
0,5	$6{,}16 \times 10^{-2}$	$2{,}06 \times 10^{-1}$	$1{,}90 \times 10^{-1}$	$5{,}79 \times 10^{-2}$	$2{,}70 \times 10^{-1}$

6. Conclusão

O presente capítulo descreve a programação imunológica gramatical, um algoritmo imuno-inspirado para a evolução de programas guiados por gramáticas formais, combinando a técnica de seleção clonal conhecida como CLONALG com o mapeamento adotado na evolução gramatical (EG). Aplicações em problemas inversos mostram que a técnica consegue encontrar modelos acurados e com capacidade de generalização, mesmo quando os dados apresentam ruídos. A inferência de modelos dinâmicos na forma de sistemas de equações diferenciais ordinárias é uma das várias tarefas na descoberta de conhecimento a partir de dados, e a programação imunológica gramatical mostrou-se promissora.

Referências

Augusto, D.A.; Barbosa, H.J.C.; Barreto, A.M. & Bernardino, H.S., Evolving numerical constants in grammatical evolution with the ephemeral constant method. In: Proc. of the Portuguese Conference on Progress in Artificial Intelligence – EPIA. Berlin, Heidelberg: Springer-Verlag, p. 110–124, 2011.

Bernardino, H.S., *Programação Imunológica Gramatical para Inferência Automática de Modelos e Projeto Ótimo de Estruturas*. Tese de doutorado, Laboratório Nacional de Computação Científica, Petrópolis, RJ, Brasil, 2012. Orientador: Helio J. C. Barbosa.

Bernardino, H.S. & Barbosa, H.J.C., Grammar-based immune programming for symbolic regression. In: Proc. Intl. Conf. on Artificial Immune Systems – ICARIS. Springer Berlin / Heidelberg, v. 5666 de *LNCS*, p. 274–287, 2009.

Bernardino, H.S. & Barbosa, H.J.C., Comparing two ways of inferring a differential equation model via grammar-based immune programming. In: Proc. of the Iberian-Latin-American Congress on Computational Methods in Engineering. Asociación Argentina de Mecánica Computacional, v. 29, p. 9107–9124, 2010a.

Bernardino, H.S. & Barbosa, H.J.C., Grammar-based immune programming. *Natural Computing*, 10(1):209–241, 2010b.

Bernardino, H.S. & Barbosa, H.J.C., Inferring systems of ordinary differential equations via grammar-based immune programming. In: Liò, P.; Nicosia, G. & Stibor, T. (Eds.), Proc. Intl. Conf. on Artificial Immune Systems – ICARIS. Springer Berlin / Heidelberg, v. 6825 de *LNCS*, p. 198–211, 2011.

Bernardino, H.S.; Castro, E.S.; ao N.C. Guerreiro, J. & Barbosa, H.J.C., Inferring strains on a locally deformed pipe via grammar-based immune programming. In: Proc. of the Iberian Latin American Congress on Computational Methods in Engineering – CILAMCE. Washington, DC, USA, 2011.

Burnet, F.M., A modification of Jerne's theory of antibody production using the concept of clonal selection. *Australian Journal of Science*, 20:67–69, 1957.

Cao, H.; Kang, L. & Chen, Y., Evolutionary modelling of systems of ordinary differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 1(4):309 – 337, 2000.

Castro, L.N. & von Zuben, F.J., The clonal selection algorithm with engineering applications. In: Proc. of the Genetic and Evolutionary Computation Conference – GECCO. p. 37, 2000.

Castro, L.N. & von Zuben, F.J., Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002.

Chomsky, N., *Syntactic Structures*. Mouton de Gruyter, 2002.

Ciccazzo, A.; Conca, P.; Nicosia, G. & Stracquadanio, G., An advanced clonal selection algorithm with ad-hoc network-based hypermutation operators for synthesis of topology and sizing of analog electrical circuits. In: Bentley, P.J.; Lee, D. & Jung, S. (Eds.), Proc. of the Intl. Conf. on Artificial Immune Systems – ICARIS. Springer, v. 5132 de *LNCS*, p. 60–70, 2008.

Gan, Z.; Chow, T.W. & Chau, W., Clone selection programming and its application to symbolic regression. *Expert Systems with Applications*, 36(2):3996 – 4005, 2009a.

Gan, Z.; Yang, Z.; Shang, T.; Yu, T. & Jiang, M., Automated synthesis of passive analog filters using graph representation. *Expert Systems with Applications*, 37(3):1887 – 1898, 2010.

Gan, Z.; Zhao, M.B. & Chow, T.W., Induction machine fault detection using clone selection programming. *Expert Systems with Applications*, 36(4):8000 – 8012, 2009b.

Hofbauer, J. & Sigmund, K., *Evolutionary Games and Population Dynamics*. Cambridge University Press, 1998.

Hugosson, J.; Hemberg, E.; Brabazon, A. & O'Neill, M., An investigation of the mutation operator using different representations in grammatical evolution. In: Proc. of the Intl. Symposium Advances in Artificial Intelligence and Applications. v. 2, p. 409–419, 2007.

Hugosson, J.; Hemberg, E.; Brabazon, A. & O'Neill, M., Genotype representations in grammatical evolution. *Applied Soft Computing*, 10(1):36 – 43, 2010.

Johnson, C.G., Artificial immune system programming for symbolic regression. In: Proc. of the European Conference on Genetical Programming – EuroGP. Springer, v. 2610 de *LNCS*, p. 345–353, 2003.

Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

Lau, A. & Musilek, P., Immune programming models of cryptosporidium parvum inactivation by ozone and chlorine dioxide. *Information Sciences*, 179(10):1469–1482, 2009.

McKinney, B.A. & Tian, D., Grammatical immune system evolution for reverse engineering nonlinear dynamic bayesian models. *Cancer Informatics*, 6:433–447, 2008.

Mera, N.; Elliott, L. & Ingham, D., Detection of subsurface cavities in IR-CAT by a real coded genetic algorithm. *Applied Soft Computing*, 2(2):129 – 139, 2002.

Musilek, P.; Lau, A.; Reformat, M. & Wyard-Scott, L., Immune programming. *Information Sciences*, 176(8):972–1002, 2006.

Noman, N. & Iba, H., Inference of gene regulatory networks using s-system and differential evolution. In: Proc. of the Genetic and Evolutionary Computation Conference – GECCO. New York, NY, USA: ACM, p. 439–446, 2005.

O'Neill, M. & Ryan, C., Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

O'Neill, M. & Ryan, C., *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

Ryan, C.; Collins, J. & Neill, M.O., Grammatical evolution: Evolving programs for an arbitrary language. In: Proc. of the European Workshop on Genetic Programming. Springer-Verlag, v. 1391 de *LNCS*, p. 83–95, 1998.

Schmidt, M. & Lipson, H., Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

Tominaga, D.; Koga, N. & Okamoto, M., Efficient numerical optimization algorithm based on genetic algorithm for inverse problem. In: Proc. of the Genetic and Evolutionary Computation Conference – GECCO. p. 251–258, 2000.