Programação Genética com Chaves Aleatórias Viciadas em Notação Pós-Fixada para o Problema de Regressão Simbólica

Pedro Vasconcellos Guimarães[1], **Alessandro Souza da Silva Júnior**[1], **Igor Machado Coelho**[1,2],**Paulo Eustaquio Duarte Pinto**[1]

[1]Instituto de Matemática e Estatística - Universidade do Estado do Rio de Janeiro
Rua São Francisco Xavier, 254, CEP 20550-900, Rio de Janeiro, RJ

[2]Instituto de Computação - Universidade Federal Fluminense
Rua Milton Tavares S/N, São Domingos, Niterói, RJ

vasconcellospedro94@gmail.com, alessandrossjr@yahoo.com.br,

{igor.machado@ime.uerj.br,imcoelho@ic.uff.br}, pauloedp@ime.uerj.br

Resumo. *O problema de regressão simbólica consiste em, dado um conjunto predefinido de símbolos, encontrar expressões algébricas que representem, com menor erro, um dado conjunto de entrada e saída. O problema é bastante desafiador, especialmente quando o conjunto de símbolos considerado é amplo, sendo comumente abordado na literatura por técnicas de programação genética, no campo das hiper-heurísticas. O método proposto é uma variação do BRKGP e utiliza notação pós-fixada permitindo uma decodificação ativa, com o uso da estrutura de pilha e não árvore como na programação genética clássica. Finalmente, este artigo utiliza datasets presentes na literatura para avaliar a qualidade das soluções e também apresenta dados de tempo de execução.*

Abstract. *The problem of symbolic regression consists of, given a predefined set of symbols, finding algebraic expressions that represent a given set of input and output with the least error . The problem is quite challenging, especially when the set of symbols is considered broad, being commonly addressed in the literature by genetic programming techniques in the field of hyperheuristics. The proposed method is a variation of BRKGP and utilizes the post-fix notation which allows active decoding and the use of the stack structure and not a classical genetic programming tree. Finally, this article uses datasets from the literature to assess the quality of solutions and also present the runtime data.*

1. Introdução

A regressão simbólica (RS) é um problema em que se parte de um conjunto de pontos, onde cada ponto é relacionado a um valor, ambos medidos empiricamente. Sendo um ponto $p = (x_0, x_1, x_2, ..., x_n) \in \mathcal{X}^n$ (tipicamente $\mathbb{R}^n$), e valor relacionado $y \in \mathcal{Y}$ (tipicamente $\mathbb{R}$) e uma função implícita $f : \mathcal{X}^n \mapsto \mathcal{Y}$, tal que $f(p) = y$, tenta-se definir a função $\tilde{f}$ tal que $\tilde{f}(p) = \tilde{y} = y + \mathcal{E}$, onde se busca que o erro $\mathcal{E}$ seja o mais próximo de zero possível. Além disso, é esperado que a função $\tilde{f}$ esteja em sua forma mais *simplificada*[1]. A técnica pode ser utilizada, por exemplo, em experimentos científicos, previsões e estatísticas. Seu espaço de soluções é infinito, o que causa uma inviabilidade de resolução do problema de forma analítica. Por conta disso, métodos aproximados são a abordagem mais intuitiva, como por exemplo a programação genética (PG) [Koza 1998].

Com inspiração na teoria da evolução por seleção natural [Darwin 1859], a PG tem como base conceitos de indivíduo, população, mutação, cruzamento, geração e adaptação. Cada indivíduo pode ser visto como uma solução do problema, e pode ser avaliado de acordo com uma função de adaptação.Uma população é um conjunto de indivíduos. A partir da população inicial, são feitos cruzamentos em indivíduos dois a dois resultando em dois descendentes com características mescladas dos pais. Criam-se novas gerações, nas quais se mantém os mais aptos e se descartam os piores. Além disso, podem ser inseridas mutações nos indivíduos, buscando aumentar a variação e evitar uma convergência precoce da população. Técnicas de computação natural com ajuste de parâmetros *caixa-preta*, como Redes Neurais [Haykin 2007], tornam árduo, ou até impossível, o processo de verificação da função abstrata obtida. Por essa razão, são consideradas na literatura técnicas *caixa-branca* como a Programação Genética.

Neste artigo é utilizada a Programação Genética com Chaves Aleatórias Viciadas (do inglês, *Biased Random Keys Genetic Programming* – BRKGP) [Gonçalves and Resende 2018], uma hiper-heurística [Burke et al. 2010] que estende o Algoritmo Genético com Chaves Aleatórias Viciadas (do inglês, *Biased Random Keys Genetic Algorithm* – BRKGA) [Gonçalves and Resende 2011] com ideias de PG. São elas: o uso de codificação e decodificação do indivíduo através de chaves aleatórias; ao invés de haver mutação nas características indivíduos, é preferível que sejam inseridos novos indivíduos totalmente aleatórios (mutantes); no cruzamento, um dos pais deve pertencer ao grupo de melhores indivíduos (elites) e deve haver mais chance de o descendente herdar características dele.

Neste trabalho, é apresentado um levantamento bibliográfico atualizado abrangendo diversas estratégias de otimização capazes de encontrar funções das mais variadas formas, com diversos níveis de precisão, e em variados tempos computacionais. Tais trabalhos foram organizados de forma concisa e, também, foram exploradas as diversas funções propostas pelos autores na literatura que, tipicamente, criam seus próprios *datasets* em cada trabalho. Em especial, foi dada ênfase ao trabalho de [Dabhi and Chaudhary 2015], cuja proposta de estrutura de dados subjacente à busca traz características de notação pós-fixada similares, mas também são evidenciados as diferenças e melhorias propostas neste trabalho. Como contribuições do presente trabalho, podem-se desta-

[1]O conceito de uma *função simples* não é formalmente definida neste artigo. Assim como na literatura [Dabhi and Chaudhary 2015], foi considerado como desejável que a função seja *humanamente amigável* para uso na prática. Tal definição é, de certa forma, *implícita*, pela natureza das funções consideradas no escopo do presente trabalho.

car: o levantamento de diversos problemas-teste da literatura, recentemente publicados; a proposta, com *design* e implementação C/C++ em código-aberto, de um BRKGP com representação da solução em notação pós-fixada e ajuste de soluções inválidas; um mecanismo inteligente de decodificação ativa capaz de evitar a geração de soluções inviáveis; a proposta de um método flexível com diversas possibilidades de calibração de parâmetros (nove no total) para a execução do algoritmo de acordo com o cenário desejado na prática.

O presente artigo conta com uma estrutura de cinco seções, incluindo esta introdução. Na Seção 2 é feita uma revisão recente da literatura. A Seção 3 apresenta o método proposto e detalhes técnicos da sua implementação. Em especial, explica-se o BRKGP, bem como a técnica de decodificação ativa utilizada. A Seção 4 contém os resultados obtidos, tanto em termos de solução quanto de tempo. Eles são organizados em tabelas e comparados com os mais recentes artigos que também tratam de regressão simbólica. Além disso, são dadas informações de como os resultados foram obtidos em temos de potência da máquina e calibrações de parâmetros utilizadas no algoritmo. Finalmente, a Seção 5 apresenta as conclusões, bem como perspectivas futuras para aprimoramento da metodologia e dos resultados.

2. Revisão da Literatura

Problemas de otimização são desafiadores na prática, ou por seu alto nível de combinações, ou pela dificuldade de encontrar técnicas que garantam a otimalidade (mínimos/máximos) em tempo computacional baixo. Por essa razão, heurísticas e meta-heurísticas, que são heurísticas genéricas com capacidade de escapar de ótimos locais, são tipicamente empregadas [Gendreau et al. 2010]. Como estratégias ainda mais genéricas, hiper-heurísticas como a programação genética são amplamente utilizadas para problemas de natureza simbólica [Koza 1998].

O problema de regressão simbólica também tem sido recentemente abordado por meio de técnicas de programação genética e meta-heurísticas. Em [Ferreira 2003], é apresentada uma técnica de *Gene Expression Programming* que permite a manipulação de constantes aleatórias. De forma geral, a estrutura tem semelhanças com o BRKGP proposto por [Gonçalves and Resende 2018], utilizando uma estrutura de *cabeça-cauda*. Tal estrutura permite a representação de funções arbitrárias, em um formato que pode ser codificado como uma sequência de chaves aleatórias. São apresentados resultados experimentais para um conjunto de funções polinomiais, as quais também foram consideradas nesse trabalho (vide resultados na Seção 4). Em 2008, [Vladislavleva et al. 2008] apresentam uma abordagem evolutiva, similar à programação genética clássica, capaz de lidar com funções não-polinomais. O foco deste trabalho consiste em avaliar graus de complexidade das funções obtidas, organizando-as como conjuntos de Pareto. Como comparação, foram consideradas algumas das funções não-polinomiais neste trabalho.

Em 2015 o trabalho de [Morgon and Pereira 2015] apresenta, de forma bastante didática, o processo de otimização para regressão simbólica, também focado em funções polinomiais. Uma variante da programação genética é desenvolvida utilizando técnicas de amostragem com filtragem na entrada de dados, e os resultados indicam que a maioria dos polinômios considerados foram corretamente encontrados pela técnica de otimização. Este trabalho demonstra a dificuldade de se ajustar corretamente alguns polinômios de pequeno porte, aqueles com grau menor de seis, devido a variações em constantes peque-

nas. O trabalho de [Dabhi and Chaudhary 2015] apresenta uma programação genética com representação de indivíduos por notação pós-fixada, que tem semelhança à estratégia adotada no presente artigo. Tal estratégia consiste em explorar a natureza sequencial de uma pilha na representação de indivíduos, evitando-se assim a representação clássica por árvores. Uma vantagem é a simplicidade da representação e facilidade de efetuar operações básicas (mutações e cruzamentos). Porém, ainda é possível gerar indivíduos inviáveis com representação inconsistente, o que é tratado por Dabhi e Chaudhary através de estratégias corretivas para árvores. Essa solução contrasta com a proposta deste artigo, que resolve o problema através de uma técnica eficiente de decodificação ativa (vide Seção 3-Algoritmo 2), que evita toda possível solução inviável em sua essência, sem necessitar de operações extras.

Finalmente, [Jin et al. 2019] apresentam[2] uma estratégia de *Bayesian Symbolic Regression* que contrasta com programação genética baseada em árvores. São propostos diversos mecanismos inovadores baseados em Cadeias de Markov e processamento Monte Carlo integrados à técnica, bem como problemas-teste não-lineares considerados no presente trabalho para comparação.

3. Método Proposto

Inicialmente foram disponibilizadas apenas as operações de soma, subtração, multiplicação e divisão podendo-se adicionar seno, cosseno e raiz quadrada. Contudo, o método não se limita a elas, pode-se incluir outras operações unárias e binárias caso as anteriores não envolvam todo o espaço de busca necessário.

Na PG clássica é utilizada uma árvore sintática que representa a solução. Outras variantes utilizam notação karva e pré-fixada como mostra a Figura 1. Neste artigo, foi utilizada a notação pós-fixada para adaptar a árvore sintática em três vetores, sendo eles a pilha, ordem das operações e ordem dos numerais a serem explicados no decorrer deste capítulo.

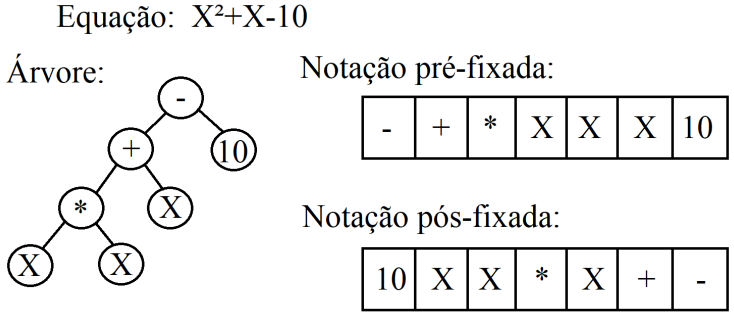

Figura 1. Exemplo de representações da solução

[2]O artigo de [Jin et al. 2019] é um *pre-print* ainda não publicado via processo de *peer-review*.

3.1. Entrada

O arquivo de Entrada deve conter a seguinte organização:

Tabela 1. Exemplo de entrada

Número de variáveis: 2	Testes: 10	Constantes: 3
Variável 1: -78	Variável 2: 36	Resultado: 6120
Variável 1: 34	Variável 2: -42	Resultado: 1114
Variável 1: 64	Variável 2: -40	Resultado: 4056
Variável 1: 24	Variável 2: -54	Resultado: 522
Variável 1: 34	Variável 2: -30	Resultado: 1126
Variável 1: -42	Variável 2: -96	Resultado: 1668
Variável 1: -56	Variável 2: 16	Resultado: 3152
Variável 1: 38	Variável 2: 34	Resultado: 1478
Variável 1: 86	Variável 2: 12	Resultado: 7408
Variável 1: -78	Variável 2: -16	Resultado: 6068
Constante única: 3.14-3.14		
Constante única: 1-1		
Constante intervalo: 1-10		

A entrada é lida em uma matriz (número de testes x numero de variáveis) contendo o valor das variáveis a serem analisadas e um vetor (numero de testes) com o resultado esperado da expressão para cada linha da matriz de variáveis. Ainda é lida a lista das contantes a serem utilizadas, podendo ser intervalos ou constantes fixas. Para isto, a quantidade de variáveis, testes e constantes ou de intervalos são informadas na primeira linha do arquivo de entrada.

A representação escolhida de uma solução foi um conjunto de quatro vetores (vide Figuras 2-5, com detalhes de funcionamento nas próximas seções).

3.2. Mapeamento da Pilha

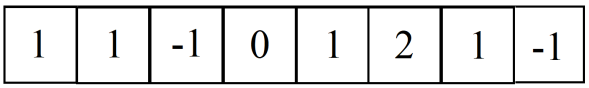

Figura 2. Exemplo de Mapeamento da Pilha

O "Mapeamento da Pilha" dita a ordem em que a expressão matemática será realizada, como na Figura 2. Seu domínio de valores está no intervalo inteiro de -1 a 2, no qual -1 representa uma operação binária (entre duas variáveis ou constantes); o 0 representa uma operação unária (envolve apenas uma constante ou variável); 1 representa uma variável ou constante; 2 representa espaço vazio que não tem interferência na expressão. Este domínio foi pensado dessa forma, pois em uma operação binária se desempilham dois elementos, realiza-se a operação e empilha-se o resultado. Logo, o tamanho do conteúdo da pilha decresce uma unidade. Para a unária se desempilha um elemento e empilha-se o resultado, logo o tamanho se mantém. Já ao incluir uma variável ou constante, o tamanho da pilha aumenta de 1. Assim, o valor presente no índice indica o crescimento da pilha em relação ao estado anterior, com exceção do 2 que não é considerado. A presença do 2 faz com que o tamanho lógico da pilha seja variável, aumentando o espaço de busca do algoritmo, sendo preciso apenas informar um tamanho máximo.

3.3. Ordem dos Numerais

0	-1	N/A	N/A	0	N/A	2	N/A

Figura 3. Exemplo de Ordem dos Numerais

O vetor de Ordem dos Numerais contém valores significativos nos mesmos índices nos quais o "Mapeamento da Pilha" contém valor 1, como na Figura 3. Seu domínio consiste em valores positivos sendo os índices para as variáveis, com zero significando a primeira e a quantidade de variáveis subtraída de 1 representando a última. Também há valores negativos, mas estes se referem a índices na estrutura das constantes, sendo o valor -1 o último índice e o valor mais negativo o primeiro índice. Pode-se observar um exemplo desse mapeamento na Figura 4.

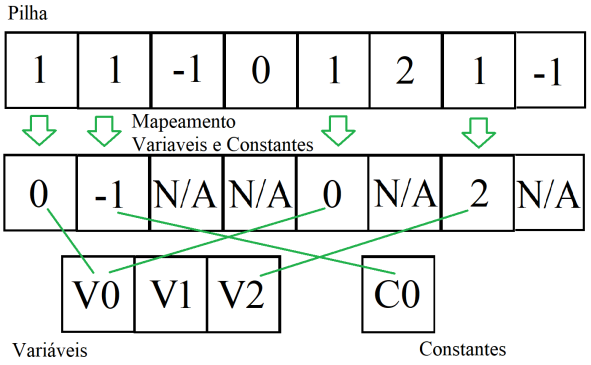

Figura 4. Exemplo de Mapeamento de Variáveis e constantes

3.4. Ordem das Operações

N/A	N/A	1	0	N/A	N/A	N/A	0

Figura 5. Exemplo de Ordem das Operações

Similar ao vetor "Ordem dos Numerais", o "Ordem das Operações" contém valores significativos nos mesmos índices do "Mapeamento da Pilha" que contém os valores -1 (operação binária) ou 0 (operação unária), como na Figura 5. Seu objetivo é mapear as operações pela posição. Sendo assim, os valores de seu domínio variam de 0 até o tamanho do vetor de operações consideradas, dependendo do tipo de operação mapeada na pilha. Essa abordagem faz com que as soluções tratadas na meta-heurística sejam constituídas apenas de mapeamentos, separando assim das particularidades do problema. Pode-se observar um exemplo desse mapeamento na Figura 6.

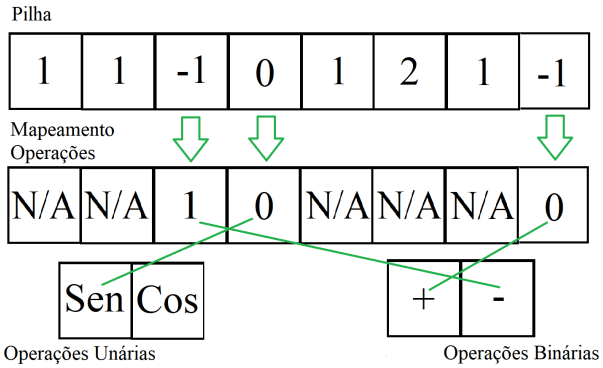

Figura 6. Exemplo de Mapeamento de operações unárias e binárias

3.5. Sementes

Por último tem-se o vetor de sementes. Quando o Ordem de Numerais indica que o valor a ser empilhado é uma constante, verifica-se qual a sua classificação, se é um intervalo ou um valor único. Caso seja do primeiro tipo, é necessário utilizar uma função randômica a partir de uma semente para determinar qual valor será usado. Para que essa informação não se perca, é preciso armazenar nessa estrutura a semente para resgatar a mesma solução em um passo posterior e, com isso, não gerar indeterminismo.

3.6. Indivíduo

O indivíduo é um conjunto de chaves aleatórias no intervalo [0,1) como mostra a figura 7. Ele será composto da junção por justaposição do "Mapeamento da Pilha", "Ordem dos Numerais", "Ordem dos Operadores" e "Sementes", nessa ordem. Cada chave será decodificada dependendo da parte onde ela se encontra, fazendo com que o valor nesse intervalo tenha o significado no vetor que representa.

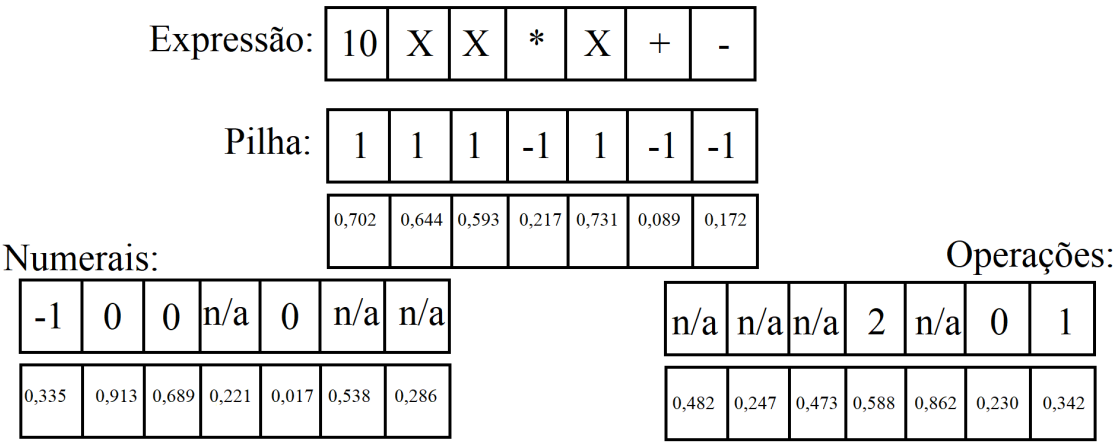

Figura 7. Exemplo de individuo

3.7. BRKGP

O BRKGP entrega um indivíduo aproximadamente otimizado como solução. Primeiramente, é gerada a população inicial de forma aleatória. Com isso, é preciso fazer a

decodificação das chaves, calculando o custo de cada indivíduo de acordo com a função *fitness*, tendo sido utilizada a Raiz do Erro Médio Quadrático (do inglês *Root Mean Square Error* ou *RMSE*). Sabendo o valor de cada indivíduo, eles são separados em dois grupos. Os melhores são os elites e os outros os não elite. Com a população inicial devidamente classificada, será gerada uma nova. Essa terá os elites, indivíduos gerados de forma totalmente aleatória, também chamados de mutantes, e os gerados pelo cruzamento dos elites com qualquer outro. Esse último ocorre escolhendo-se entre cada chave dos pais qual será passada para o descendente. A chave do pai elite tem mais chance de ser selecionada.

Com esse novo conjunto, os indivíduos são decodificados, seus custos calculados e os elites definidos novamente. São comparados os melhores dentre as duas populações e o melhor é guardado. Esse processo é repetido até que haja uma quantidade pré-definida de não melhoras entre as populações. Ao término, é reiniciada uma nova população inicial de forma aleatória e tudo é repetido novamente até que tenha uma quantidade pré-definida de não melhoras entre esses reinícios. Isso é feito para aumentar a diversidade do método que pode ser bastante influenciado pela população inicial. O melhor indivíduo encontrado após esse processo é a solução retornada. A idéia está desenvolvida no Algoritmo 1.

Algoritmo 1: BRKGP

Entrada: Parâmetros do BRKGP P; Conjunto de Entradas e Saídas para avaliação $Q = (Inp, Out)$; Critérios de parada (restartMax, noImprovementMax)
Saída: Melhor indivíduo decodificado I com custo f

```
1  restart ← 0;
2  custoMelhorIndivíduo ← ∞ ;
3  enquanto restart < restartMax faça
4      noImproviment ← 0;
5      Pop ← GeraPopulaçãoAleatória();
6      Pop ← DecodificaPopulação(Q, Pop);
7      Pop ← OrdenaIndividuosPeloCusto(Pop);
8      custoMelhorIndivíduo ←custoMelhorIndG_0(Pop);
9      enquanto noImprovement < noImprovementMax faça
10         Pop' ← GeraMutanteGeração_{k+1}(P, Pop);
11         Pop' ← Crossover(P, Pop');
12         Pop' ← DecodificaPopulação(Q, Pop');
13         Pop' ← OrdenaIndividuosPeloCusto(Pop');
14         se custoMelhorIndG_{k+1}(Pop') < custoMelhorIndG_k(Pop) então
15             noImprovement← 0;
16         senão
17             noImprovement ← noImprovement+1;
18         fim
19         se custoMelhorIndG_{k+1}(Pop') < custoMelhorIndividuo então
20             custoMelhorIndivíduo ← custoMelhorIndG_{k+1}(Pop');
21             restart← 0;
22         fim
23         Pop ← Pop';
24     fim
25     restart ← restart+1;
26 fim
```

3.8. Decodificação Ativa

A utilização de cruzamento chave a chave pode gerar soluções inválidas. Por exemplo, ao fazer cruzamento de x^2+x-10 que, em notação pós fixada é representada por 10xx*x+-, com $4x^2$ ou xx+xx+*, observe que, caso o descendente seja gerado herdando o terceiro elemento do segundo pai e o restante do primeiro, obteríamos 10x+*x+-, onde o * teria somente um operando tornando a expressão matemática sem sentido. Para contornar essas situações, ao invés de restringir os cruzamentos, é permitido realizá-los e, após sua realização, é feito um ajuste caso a solução seja inviável.

Para isso, é preciso checar para cada item da pilha se o seu tamanho é pelo menos 1, porque, se a pilha for zerada, significa que haverá operação binária sendo feita com somente um operando. Também é preciso verificar se o tamanho final da pilha é 1, porque, ao final da expressão, é esperado que se tenha um e somente um valor como resultado. É essa a ideia desenvolvida no Algoritmo 2.

Vale ressaltar que, devido à utilização do indivíduo com chaves aleatórias, todas as posições de todos os vetores contém valor, portanto, ao fazer esse ajuste na pilha, há a informação das operações e operandos que não estavam sendo utilizadas, e que agora passarão a ser.

3.9. Parâmetros

Para permitir maior calibragem ao algoritmo são aceitos parâmetros de entrada. Eles determinam os elementos seguintes (Tabela 2).

Tabela 2. Parâmetros globais do BRKGP

Influencia em	Nome	Descrição
Regressão simbólica	$StackLen$	Tamanho máximo da expressão matemática
Regressão simbólica	$MaxConst$	Quantidade máxima de constantes permitidas
BRKGP	$PopulationLen$	Quantidade de indivíduos
BRKGP	$EliteSize$	Porcentagem de elites na população
BRKGP	$MutantSize$	Porcentagem de mutantes na população
BRKGP	$EliteBias$	Chance de propagação de gene elite
Chamadas ao BRKGP	$RestartMax$	Reinício do BRKGP com população aleatória
BRKGP	$NoImprovementMax$	Quantidade de gerações máxima sem melhora
Validação	$Validation$	Porcentagem da entrada reservada para treino
Aleatório	$Seed$	Semente do aleatório

Algoritmo 2: Decodificação Ativa

Entrada: Um indivíduo $S \in [0,1]^n$ codificado por chaves aleatórias;tamPilha;nVars;nConst;maxConst;
Saída: Indivíduo decodificado $S \in [0,1]^n$ válido;

$somador \leftarrow 0$; $auxSomador \leftarrow 0$; $decodValue \leftarrow 0$;
$contConst \leftarrow 0$; $idConst \leftarrow 0$;
para *cada elemento* $i \in \{0..tamPilha - 1\}$ **faça**
  $decodValue \leftarrow \lfloor S[i] \cdot 4 \rfloor - 1$;
  $idConst \leftarrow \lfloor (S[stackLen + i]) \cdot (nVars + nConst) \rfloor - nConst$;
  se $decodValue \neq 2$ **então**
    $auxSomador \leftarrow auxSomador + decodValue$;
  fim
  se $auxSomador < 1$ **então**
    se $S[i] < 25\%$ **então**
      $S[i] \leftarrow S[i] + 50\%$;
    senão
      se $S[i] >= 75\%$ **então**
        $S[i] \leftarrow S[i] - 25\%$;
      senão
        $S[i] \leftarrow S[i] + 25\%$;
      fim
      $auxSomador \leftarrow somador + 1$;
    fim
  fim
  se $auxSomador > (stackLen - i)$ **então**
    se $somador = 1$ **então**
      $S[i] \leftarrow S[i] + 25\%$; $auxSomador \leftarrow 1$;
    senão
      se $somador > 1$ **então**
        se $(S[i] >= 25\%)$ ***e*** $(S[i] < 50\%)$ **então**
          $S[i] \leftarrow S[i] - 25\%$;
        senão
          se $S[i] >= 75\%$ **então**
            $S[i] \leftarrow S[i] - 75\%$;
          senão
            $S[i] \leftarrow S[i] - 50\%$;
          fim
          $auxSomador = somador - 1$;
        fim
      fim
    fim
  fim
  se $(decodValue = 1)$ ***e*** $idConst < 0$ **então**
    $contConst \leftarrow contConst + 1$;
    se $contConst > maxConst$ **então**
      $S[stackLen + i] \leftarrow ((1/(nVars + nConst)) \cdot nConst) + ((1/(nVars + nConst)) \cdot (randomNumber)$;
    fim
  fim
  $somador \leftarrow auxSomador$;
fim

4. Experimentos Computacionais

Para experimentos computacionais, foi utilizado um computador com Processador Intel(R) Core(TM) i5-9400F e 16GB de memória RAM. Foi utilizado o sistema operacional Windows 10 com tecnologia WSL para Ubuntu 18.04. Não foi encontrado na literatura um conjunto unificado de testes. Cada autor propõe seu próprio. Uma das contribuições deste trabalho é a agregação de diversas bases em um formato unificado. Cada conjunto de testes tem especificidades na tabela de símbolos, tendo sido necessária uma calibração independente do algoritmo para cada tipo de teste. Apresenta-se 4 tipos distintos, nomeados como SET1-SET4 (vide Tabela 3). Foi feita uma calibração empírica (vide Tabelas 4-5), porém não muito aprofundada, pois alguns parâmetros podem ser excessivamente altos em relação ao tempo efetivamente consumido, e não foi feito uso de nenhuma metodologia de calibração exaustiva como o IRACE.

Foi utilizada a metodologia de avaliação cruzada com conjuntos de teste e de validação, com intervalos definidos como 70% e 30%. O cálculo de erro considerado foi o *Root Mean Square Error*:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{N}(\tilde{y}_k - y_k)^2}{N}}$$

4.1. Descrição dos problemas-teste

Tabela 3. Descrição dos problemas-teste da literatura

Conjunto	Autores	Quantidade	Tipo de Função	Intervalo de Dados
SET1	[Morgon and Pereira 2015]	6	Polinômios	[-1,1]
SET2	[Dabhi and Chaudhary 2015]	1	Polinômios	[1,10]
SET3	[Jin et al. 2019]	5	Polinômios*	[-1,1]
SET4	[Vladislavleva et al. 2008]	3	Variado	[-1,1]

* Existe apenas um problema em comum no SET3 e SET4, que inclui funções de seno e cosseno, além de polinômios (originalmente de [Vladislavleva et al. 2008])

Todos os SETs foram avaliados usando 20 entradas diferentes, com exceção do SET2 que foi reproduzido fielmente com 10 entradas. Nos SET1 e SET2 todas as funções foram avaliadas utilizando pontos equidistantes dentro de seus respectivos intervalos. Já nos SET3 e SET4 foram utilizadas valores aleatórios de entrada gerados com distribuição uniforme.

Tabela 4. Parâmetros globais do BRKGP

Nome	Descrição	Valor Adotado
$populationLen$	Tamanho da população	100
$eliteSize$	Porcentagem de elites na população	25%
$mutantSize$	Porcentagem de mutantes na população	10%
$eliteBias$	Chance de propagação de gene elite	85%

Tabela 5. Parâmetros específicos do algoritmo para cada conjunto de teste

	Parâmetros ligados ao conjunto				
Conjunto	Restarts	NoImprovement	Constantes	MaxConstantes	StackLen
SET1	20	5000	$\{1..10\}$	6	41
SET2	20	20000	$\{1..10\}$	6	41
SET3	20	20000	$\{1..10\}, 0, 1$	6	41
SET4	20	20000	$\{1..10\}$	6	41

4.2. Análise dos Resultados

O primeiro set contem problemas com menor grau de dificuldade, sendo composto exclusivamente de polinômios. A Tabela 6 apresenta colunas "Best RMSE" e "RMSE (médio)", com os melhores valores e valor médio do RMSE para dados de treinamento e dados de validação, bem como os tempos médios de execução, a melhor solução e seu respectivo tempo. Pode-se observar que o método proposto obteve as funções corretamente (erro zero) para quatro dos seis problemas-teste. Note que as funções encontradas são equivalentes, porém devido ao requisito do tamanho de pilha ser fixo, o método consome "espaço extra" com operações desnecessárias. Por exemplo, a primeira função encontrada $(x - x) + (x^2 + x) * x + x - (x - x)$ é equivalente a $x^3 + x^2 + x$. Então, apresentam-se na tabela somente as soluções em formato simplificado. É observado que o método não teve limite de tempo estipulado, consumindo entre 94 e 996 segundos em média, parando somente quando os parâmetros esperados são atingidos.

O SET2, SET3 e SET4 são apresentados de forma análoga nas Tabelas 7-9. O segundo conjunto apenas inclui uma função, que não obteve erro zero. Vale ressaltar a grande dificuldade do método em ajustar a pequena constante de valor 1, frente à magnitude das demais operações (sendo ajustada de forma "aproximada" como $\frac{10x^3}{4+x^2}$). Já no SET3, uma situação similar ocorre com a função $6sen(x_0)cos(x_1)$, na qual o método tem dificuldade em encontrar um divisor correto (também no valor 1, sendo aproximado como $0,028\ x_0\ x_1\ cos\ x_0$). O tempo computacional necessário e o erro aumentam fortemente à medida que funções mais complexas são buscadas, como no SET4, com tempo gasto de 5873 segundos.

Tabela 6. SET1

Problemas-teste	Treinamento(70%)		Validação(30%)	
	Best RMSE	**RMSE (médio)**	**Best RMSE**	**RMSE (médio)**
$x^3 + x^2 + x$	**0,000**	**0,000**	**0,000**	**0,000**
$x^4 + x^3 + x^2 + x$	**0,000**	0,021	**0,000**	0,023
$x^5 + x^4 + x^3 + x^2 + x$	**0,000**	0,065	**0,000**	0,063
$x^6 + x^5 + x^4 + x^3 + x^2 + x$	0,034	0,094	0,017	0,074
$x^4 + x^3 + x^2 + x + 1$	**0,000**	0,084	**0,000**	0,085
$2x^5 - 3x^4 + 4x^3 - 5x^2 + 6x - 7$	0,262	0,530	0,046	0,476

Problemas-teste	Tempo de execução(s)		Melhor Solução
	Médio	**Melhor**	
$x^3 + x^2 + x$	94,989	84,828	$x^3 + x^2 + x$
$x^4 + x^3 + x^2 + x$	575,539	571,469	$x^4 + x^3 + x^2 + x$
$x^5 + x^4 + x^3 + x^2 + x$	950,973	1047,219	$x^5 + x^4 + x^3 + x^2 + x$
$x^6 + x^5 + x^4 + x^3 + x^2 + x$	797,186	1321,578	$\frac{7x^5 + 14x^4 + 8x^3 + 10x^2 + 9x}{9}$
$x^4 + x^3 + x^2 + x + 1$	996,666	656,578	$x^4 + x^3 + x^2 + x + 1$
$2x^5 - 3x^4 + 4x^3 - 5x^2 + 6x - 7$	892,872	969,219	$x^4 - 5x^3 + 6x^2 - 6x + 11$

Tabela 7. SET2

Problemas-teste	Treinamento(70%)		Validação(30%)	
	Best RMSE	**RMSE (médio)**	**Best RMSE**	**RMSE (médio)**
$5x^4 + 4x^3 + 3x^2 + 2x + 1$	4,671	14,433	3,041	13,469

Problemas-teste	Tempo de execução(s)		Melhor Solução
	Médio	**Melhor**	
$5x^4 + 4x^3 + 3x^2 + 2x + 1$	1648,131	1148,937	$5x^4 + 4x^3 + 2x^2 + \frac{10x^3}{4 + x^2}$

Tabela 8. SET3

Problemas-teste	Treinamento(70%)		Validação(30%)	
	Best RMSE	**RMSE (médio)**	**Best RMSE**	**RMSE (médio)**
$2,5x_0^4 - 1,3x_0^3 + 0,5x_1^2 - 1,7x_1$	0,327	0,390	0,258	0,379
$8x_0^2 + 8x_1^3 - 15$	0,812	0,946	0,747	1,002
$0,2x_0^3 + 0,5x_1^3 - 1,2x_1 - 0,5x_0$	0,180	0,201	0,153	0,204
$6\ sen\ x_0\ cos\ x_1$	0,008	0,127	0,010	0,126
$1,35 + 5,5\ sen((x_0 - 1)(x_1 - 1))$	0,546	0,691	0,404	0,618

Problemas-teste	Tempo de execução(s)		Melhor Solução
	Médio	**Melhor**	
$2,5x_0^4 - 1,3x_0^3 + 0,5\ x_1^2 - 1,7x_1$	1583,350	1659,531	$x_0^3 - 1,8x_1$
$8x_0^2 + 8x_1^3 - 15$	3333,078	2026,953	$4x_0^2 - x_0 + 7x_1^3 + x_1^2 - 15$
$0,2x_0^3 + 0,5x_1^3 - 1,2x_1 - 0,5x_0$	1875,400	2163,328	$0,3x_1^4\ x_0^2 - 0,03x_1^2 * x_0 - 0,401x_0 - x_1$
$6\ sen\ x_0\ cos\ x_1$	3389,803	3599,375	$\frac{6sen\ x_0\ cos\ x_1}{0,028\ x_0\ x_1\ cos\ x_0}$
$1,35+5,5\ sen((x_0-1)\ (x_1-1))$	2247,701	3054,078	$(8 - x_1 - cos(x_1 - 0,145))\ (sen(0,995\ (x_0x_1 - x_0 + 0,091\ x_1 - sen\ x_1 - 2.1)))$

Tabela 9. SET4

Problemas-teste	Treinamento(70%)		Validação(30%)	
	Best RMSE	RMSE (médio)	Best RMSE	RMSE (médio)
$30\left(\frac{(x_0-1)(x_2-1)}{x_1^2\,(x_0-10)}\right)$	1,547	2,291	1,249	2,272
$6\ sen\ x_0\ cos\ x_1$	0,008	0,127	0,010	0,126
$(x_0-3)(x_1-3)+2\ sen((x_0-4)(x_1-4))$	0,677	0,855	0,649	0,839

Problemas-teste	Tempo de execução(s)		Melhor Solução
	Médio	Melhor	
$30\left(\frac{(x_0-1)(x_2-1)}{x_1^2\,(x_0-10)}\right)$	3164,812	5095,922	$\frac{\frac{x_0\ x_2+11x_2-x_2^2}{4}+2x_0-x_1+x_2-3}{x_1^2}$
$6\ sen\ x_0\ cos\ x_1$	3389,803	3599,375	$\frac{6\ sen\ x_0\ cos\ x_1}{0,028\ x_0\ x_1\ cos\ x_0}$
$(x_0-3)(x_1-3)+2\ sen((x_0-4)(x_1-4))$	3589,844	5873,734	$\frac{9}{\frac{7x_0^2\ x_1+7\ x_0\ x_1^2}{6-\left(\frac{x_1^2-x_1\ cos(sen\ x_1)}{4}\right)}+cos\left(\frac{x_0}{3}\right)}$

5. Conclusão e Trabalhos Futuros

No presente trabalho, foi discutido o problema de regressão simbólica, buscando o ajuste de uma função com formato amigável a humanos, contrastando com técnicas de *otimização caixa-preta*. Considera-se um conjunto amplo de problemas-teste coletados na literatura, envolvendo principalmente funções polinomiais, e algumas funções não-polinomiais. A técnica proposta é um BRKGP utilizando decodificação ativa com uma representação de indivíduos por notação pós-fixada. Os resultados encontrados indicaram que o método consegue encontrar diversas funções com exatidão, enquanto para outras, o erro permanece alto. Dado o imenso número de possibilidades e formatos de representação das funções, ainda há espaço para melhorias da técnica, tendo em vista a redução dos erros RMSE informados. Aspectos positivos deste trabalho incluem: transparência nos tempos computacionais e hardware utilizado nos testes, que não estão disponíveis para a maioria dos trabalhos na literatura; amplo conjunto de testes abordado, pois foram considerados diversos autores no período de 2003-2019; simplicidade da técnica, por não depender de operações em árvores e não necessitar de operações corretivas para soluções inviáveis.

Como trabalhos futuros, cogita-se estender o corpo de testes, bem como avaliar de forma mais detalhada o grau de complexidade das funções obtidas por análise de dominância Pareto. Para melhoria dos resultados, pensa-se em incorporar ideias de exploração de vizinhanças e buscas locais, no escopo de outras hiper-heurísticas inovadoras como o *Variable Neighborhood Programming* [Elleuch et al. 2016], que é um tópico bastante desafiador, dada a natureza da representação computacional considerada.

Referências

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages

449–468. Springer.

Dabhi, V. K. and Chaudhary, S. (2015). Solution modeling using postfix genetic programming. *Cybernetics and Systems*, 46(8):605–640.

Darwin, C. (1859). *On the origin of species by means of natural selection*. Murray, London (Edição 1968).

Elleuch, S., Mladenovic, N., and Jarboui, B. (2016). *Variable Neighborhood Programming: A New Automatic Programming Method in Artificial Intelligence*. GERAD HEC Montréal.

Ferreira, C. (2003). Function finding and the creation of numerical constants in gene expression programming. In *Advances in soft computing*, pages 257–265. Springer.

Gendreau, M., Potvin, J.-Y., et al. (2010). *Handbook of metaheuristics*, volume 2. Springer.

Gonçalves, J. F. and Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Gonçalves, J. and Resende, M. (2018). *Biased random-key genetic programming*, volume 1, pages 23–37. Springer.

Haykin, S. (2007). *Neural networks: a comprehensive foundation*. Prentice-Hall, Inc.

Jin, Y., Fu, W., Kang, J., Guo, J., and Guo, J. (2019). Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*.

Koza, J. R. (1998). *Genetic Programming*. MITPress, London.

Morgon, R. and Pereira, S. d. L. (2015). RegressÃo simbÓlica com seleÇÃo por amostragem. Technical report.

Vladislavleva, E. J., Smits, G. F., and Den Hertog, D. (2008). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349.