Programação Genética para o problema de regressão simbólica

1 Introdução

A programação genética aplicada ao problema de regressão simbólica é uma técnica de inteligência artificial baseada em evolução natural, o qual visa encontrar uma função matemática capaz de prever valores de uma variável de saída com base em um conjunto de variáveis de entrada [2]. Nessa técnica, a evolução artificial é utilizada para otimizar a busca por uma solução adequada, de forma análoga à evolução biológica, em que uma população de indivíduos é submetida a operadores genéticos, como a seleção, o cruzamento e a mutação, para gerar novos indivíduos [7].

No contexto da regressão simbólica, cada indivíduo representa uma função matemática que é avaliada com base na sua capacidade de se ajustar aos dados de treinamento [12]. O processo de evolução ocorre por meio dos operadores genéticos, que combinam características de dois indivíduos para produzir novos indivíduos [14]. O objetivo é encontrar a função que melhor se ajusta aos dados de treinamento, evitando o *overfitting*, ou seja, o ajuste excessivo [1].

A programação genética aplicada à regressão simbólica é amplamente utilizada em diversas áreas, como finanças, engenharia e medicina, para prever valores de saída com base em um conjunto de variáveis de entrada [9]. Esta técnica tem demonstrado eficácia em problemas complexos de regressão simbólica, com resultados superiores ou comparáveis aos obtidos com outras técnicas de inteligência artificial, como redes neurais artificiais e árvores de decisão [8].

O algoritmo desenvolvido neste trabalho foi testado em três conjuntos de dados, sendo dois sintéticos e um real. Na seção 2, deste trabalho, é apresentado a etapa de desenvolvimento com descrição sobre a implementação do PG, incluindo informações sobre a representação dos indivíduos, *fitness* e operadores utilizados. Já na seção 3, são mostrado os resultados obtidos assim com o método de experimentação utilizado, por fim, na seção 4 é feita uma análise sobre os resultados e os impactos dos parâmetros no resultado.

2 Desenvolvimento

A primeira etapa no desenvolvimento deste algoritmo de programação genética foi a definição do conjunto de funções e terminais que serão utilizados na geração dos indivíduos. Uma vez definido o conjunto de funções e terminais, o próximo passo foi a geração aleatória de uma população inicial de indivíduos. Esses indivíduos são então avaliados com base em sua aptidão para resolver o problema. Na terceira etapa, foi realizada a seleção dos indivíduos mais aptos para reprodução. Essa seleção foi implementada considerando os métodos por roleta, torneio e e-lexicase. Após a seleção, é então realizada a reprodução dos indivíduos selecionados por meio de operadores genéticos como crossover e/ou mutação. A quarta etapa é a avaliação dos indivíduos filhos, que são gerados a partir dos indivíduos selecionados na etapa anterior. Essa avaliação é feita utilizando a mesma função de aptidão definida na etapa inicial. O processo de seleção, reprodução e avaliação é repetido até que um critério de parada seja alcançado, neste caso, um número máximo de gerações. Abaixo temos alguns detalhes a respeito de cada etapa.

2.1 Modelagem do indivíduo

Neste trabalho, optou-se por representar o indivíduo por uma árvore composta por nós terminais e operadores. Cada nó terminal da árvore pode ser uma variável ou um número aleatório dentre 0.1, 0.2, 0.5, 1, 2,

3, 4, 5, e os operadores podem ser as operações matemáticas básicas ($+$, $-$, $/$, $*$). Para as as bases *synth1* e *synth2* é feito a leitura dos variáveis x e y, da base, enquanto que para a base *concrete*, são recebidos x, y, z, a, b, c, p, q da base.

Com este individuo formado, é possível construir uma função matemática a partir do caminhamento em ordem dos nós e gerar um conjunto de saídas a partir de um conjunto de entradas. Os nós terminais, ou folhas, das árvores podem ser baseados em constantes ou variáveis do problema, somente, não permitindo um operadores, com adição ou subtração, em nós folha (terminais). Para avaliar a qualidade de cada indivíduo, foi utilizado o critério de avaliação NRMSE (raiz quadrada do erro quadrático médio normalizada). O uso de operadores elitistas também foi considerado no algoritmo, possibilitando a adição do indivíduo gerado à população apenas se ele for melhor que todos os pais.

2.2 Fitness

Como mencionado anteriormente, neste trabalho os indivíduos de uma população são avaliados utilizando a métrica NRMSE (*Normalized root mean squared error*) [5, 16, 6]. Essa métrica é amplamente utilizada em problemas de regressão e permite avaliar a qualidade da solução encontrada pela programação genética, o qual, pode ser calculada por:

$$NRMSE(ind) = \sqrt{\frac{\sum_i^n (y_i - eval(Ind, x_i))^2}{\sum_i^n (y_i - \bar{y})^2}} \tag{1}$$

onde N é o número de instâncias fornecidas, Ind é o indivíduo sendo avaliado, $eval(ind, x_i)$ avalia o indivíduo ind na i-ésima instância de X, y_i é a saída esperada para a entrada x_i e $\bar{Y}$ é a média do vetor de saídas Y.

Com base em uma expressão matemática, gerada pelo caminhamento em ordem da árvore (indivíduo), calculamos a *fitness* do indivíduo. Se, para algum X, o valor da expressão for inválido, por exemplo uma divisão por zero, o resultado é definido como zero.

O uso de métricas de avaliação é fundamental, pois permite comparar diferentes soluções e selecionar as melhores para compor a próxima geração de indivíduos. Além disso, a escolha da métrica apropriada pode influenciar significativamente a eficácia do algoritmo.

Ainda no calculo de fitness, foi considerado a opção de penalizar o crescimento excessivo dos individuo, também conhecido como *Bloat* [1, 7]. Para isto, utilizamos a técnica de penalidade, que consiste em adicionar uma penalidade na função de aptidão (fitness) dos indivíduos que possuem uma alta complexidade, verificada a partir do tamanho da arvore, onde quanto maior a complexidade (tamanho), maior será a penalidade, de fator ajustável, adicionada na função de aptidão do indivíduo.

2.3 População

A população inicial do PG é gerada utilizando o método *Ramped Half-and-Half* (RHH) [11] que combina os métodos *full* e *grow* para aumentar a diversidade inicial. Metade dos indivíduos da população é gerada utilizando o método *full* e a outra metade com o método *grow*. No método *grow*, um nó de uma árvore é escolhido aleatoriamente de elementos em ambos os conjuntos de terminais e funções, considerando uma profundidade máxima. Já no método *full*, o nó de uma árvore é escolhido considerando elementos apenas do conjunto de funções até que se atinja a profundidade máxima. Vale destacar que, mesmo que seja definida uma profundidade máxima, o método *grow* pode gerar árvores com profundidades menores devido à escolha aleatória do conteúdo de um nó, podendo gerar árvores irregulares. Por outro lado, o método *full* cria árvores balanceadas [7, 4].

Se a profundidade da árvore for n, então serão criados o mesmo número de indivíduos com profundidade 2 até n para cada um dos métodos $full$ e $grow$. Se não for possível gerar toda a população inicial de forma balanceada em relação aos dois métodos, parte dos indivíduos será gerada aleatoriamente. Por exemplo, se o tamanho da população for 50 e a profundidade máxima da árvore for 4, serão criados 16 indivíduos com profundidades 2, 3 e 4, sendo 8 deles inicializados utilizando *grow* e 8 *full*. Dessa forma, serão gerados 48 indivíduos. Os 2 indivíduos restantes serão criados escolhendo aleatoriamente o método.

O uso do método RHH pode introduzir *introns* na população inicial, oferecendo um efeito protetor contra o efeito destrutivo do cruzamento [7].

2.4 Operadores

O algoritmo utiliza, para seleção dos indivíduos, os métodos da roleta, torneio e o e-lexicase, juntamente com os operadores característicos do Programação Genética (PG).

2.4.1 Seleção por roleta

O método de seleção por roleta é amplamente utilizado em algoritmos genéticos (AGs) como um método de seleção de indivíduos baseado na aptidão. Neste método, cada indivíduo tem uma probabilidade de ser selecionado proporcional ao seu valor de aptidão. Uma roleta é utilizada para representar a distribuição de probabilidade dos indivíduos e é girada para selecionar os indivíduos que irão se reproduzir. Este método é conhecido por sua simplicidade e eficácia na seleção de indivíduos de alta qualidade para a reprodução em AGs [3, 15].

Para realizar a seleção por roleta, foi necessário calcular a aptidão de cada indivíduo da população e, em seguida, calcular a soma total das aptidões. Em seguida, é calculada a probabilidade de cada indivíduo ser escolhido, dividindo a aptidão do indivíduo pela soma total das aptidões. Essa probabilidade é então utilizada para determinar a posição do indivíduo na roleta.

2.4.2 Seleção por torneio

O método de seleção por torneio é um dos mais comuns em algoritmos genéticos, e consiste em selecionar um subconjunto de indivíduos da população de forma aleatória, e comparar seus valores de aptidão para a seleção dos melhores indivíduos [3, 15]. Esse processo é repetido até que o número de indivíduos desejado seja selecionado.

2.4.3 Seleção por e-lexicase

O método e-lexicase [13] seleciona, um conjunto aleatório de n casos de teste da população. Em seguida, os casos de teste são ordenados lexicograficamente e a lista de indivíduos selecionados é inicializada como vazia. Para cada indivíduo na população. Em seguida, cada indivíduo é avaliado e o erro calculado. Se o erro do indivíduo para o caso de teste for maior que o valor de épsilon, atribuído inicialmente por parâmetro, o indivíduo é descartado.

Se o indivíduo passar em todos os casos de teste, ele é adicionado à lista de indivíduos selecionados. Caso nenhum indivíduo tenha passado em todos os casos de teste, um novo conjunto aleatório de casos de teste é sorteado e o processo de seleção é repetido.

Por fim, o melhor indivíduo dentre os selecionados é escolhido baseado no menor erro de ajuste. Se nenhum indivíduo passou em todos os casos de teste, a seleção é repetida até que um indivíduo seja selecionado, oq ue pode levar a uma demanda maior de tempo.

2.4.4 Cruzamento

Na operação de cruzamento do algoritmo, dois pais são selecionados e um único filho é gerado a partir deles. Esse processo envolve a escolha aleatória de um nó de uma das árvores do primeiro pai(ou mãe) e outro nó de uma das árvores do segundo pai para trocar as subárvores. Em seguida, o filho resultante é gerado trocando um único nó da primeira árvore do pai selecionado por meio de uma técnica de seleção, como a seleção por roleta, torneio ou e-lexicase. Esse método de cruzamento é amplamente utilizado em Programação Genética (PG) para gerar novos indivíduos com características dos pais selecionados [7, 10].

2.4.5 Mutação

Na operação de mutação, a seleção é realizada apenas uma vez e o filho é gerado por meio do cruzamento entre o pai selecionado e um indivíduo gerado aleatoriamente [3]. Para selecionar o nó que será trocado tanto no cruzamento quanto na mutação, o algoritmo define uma profundidade máxima de 3 [15], limitando o tamanho dos indivíduos e evitando que cresçam descontroladamente. Esse tipo de restrição na profundidade da árvore é utilizado em algoritmos genéticos baseados em programação genética [10].

2.4.6 Elitismo

Foram utilizadas abordagens elitistas no algoritmo, onde as operações de cruzamento e mutação são executadas da mesma forma que nas versões não-elitistas, mas o filho resultante é adicionado à próxima geração somente se for melhor que os pais. Caso contrário, o melhor pai em termos de *fitness* é selecionado para continuar na próxima geração. Essa estratégia é mencionada por Goldberg em sua obra "Genetic Algorithms in Search, Optimization, and Machine Learning" [3].

Além disso, se as probabilidades de mutação e cruzamento não somam 1, a reprodução ocorre com uma probabilidade de $1-(pc+pm)$. A operação de reprodução consiste em selecionar um indivíduo e adicioná-lo à nova geração.

2.5 Implementação

Para implementar o GP, utilizou-se a linguagem de programação *Python*. Criaram-se objetos que representam os elementos fundamentais do algoritmo de Programação Genética (GP), onde cada objeto foi implementado por meio de uma classe, a qual contém atributos e métodos específicos específicos.

Para desenvolver o projeto utilizou-se a IDE PyCharm 2023.1. Na pasta do projeto pode-se encontrar os arquivos **main.py** e **gp.py**, os quais utilizam as bibliotecas **random, math, numpy, copy, pandas, time e datetime** para à implementação.

Abaixo iremos detalhar a classe *tree* (arvore), o qual representa o individuo. Tendo como atributos:

- **primitivos**: representa o conjunto de possíveis valores para um nó.
- **profundidade**: profundidade da arvore que representa o Indivíduo.
- **tamanho**: número máximo de nós no Indivíduo.
- **raiz**: raiz da árvore.

E os métodos:

- **full**: geração aleatória por método *full*.
- **grow**: geração aleatória por método *grow*.
- **build**: realiza a construção da expressão matemática percorrendo os caminhos da arvore (Indivíduo).
- **node**: escolhe um nó de uma árvore aleatoriamente.

3 Experimentos

Os experimentos computacionais foram realizados na Google Cloud Platform em servidores categoria e2-medium (2 vCPUs, 4 GB memória) com uma vCPU Intel Broadwell, executando apenas Linux Debian 11 Bulleye, sem interface gráfica instalada. Para cada experimento foram executadas 30 réplicas, obtendo os valores de melhor, pior e desvio padrão, separados por base de dados.

3.1 Synth1

Foi selecionada a primeira base de dados sintética, synth1, para realizar o estudo dos parâmetros do algoritmo de Programação Genética (GP) implementado. Essa base de dados contém 60 instâncias no conjunto de treinamento e 600 no de teste. Os parâmetros investigados foram o tamanho da população, o número de gerações, a probabilidade de mutação (pm), a probabilidade de cruzamento (pc), valor de épsilon, o tamanho do torneio, ausência de elitismo e controle de *bloat* de fator $0,05$.

Uma visão geral dos resultados pode ser visto na Tabela 1.

seleção	**pop.**	**gerações**	**crossover**	**mut.**	**torneio**	**epsilon**	**elit.**	**bloat red.**	**médio**	**DP**	**melhor**
e-lexicase	50	50	0.9	0.05		2	sim	não	1,46E+00	3,50E+00	1,03E-01
roleta	50	50	0.6	0.05			sim	não	6,12E-01	2,45E-01	1,03E-01
roleta	50	50	0.9	0.3			sim	não	8,36E-01	1,52E+00	1,03E-01
roleta	100	50	0,9	0.05			sim	não	5,82E-01	6,50E-01	8,18E-02
roleta	500	50	0,9	0.05			sim	não			
roleta	50	50	0.6	0.05			não	não	6,60E-01	1,42E-01	4,47E-01
roleta	50	50	0.6	0.05			sim	sim			
torneio	50	50	0.9	0.05	2		sim	não	7,65E-01	8,88E-01	2,27E-01
torneio	100	50	0.9	0.05	2		sim	não	4,77E-01	2,24E-01	1,03E-01
torneio	50	50	0.9	0.05	5		sim	não	6,51E-01	3,66E-01	1,81E-01

Table 1: Tabela com resultados synth1.

As Figuras 1, 2 e 3, mostram às convergências dos indivíduos, ao longo das gerações, para o experimento de melhor resultado, de cada ciclo avaliativo, dentre as replicas.

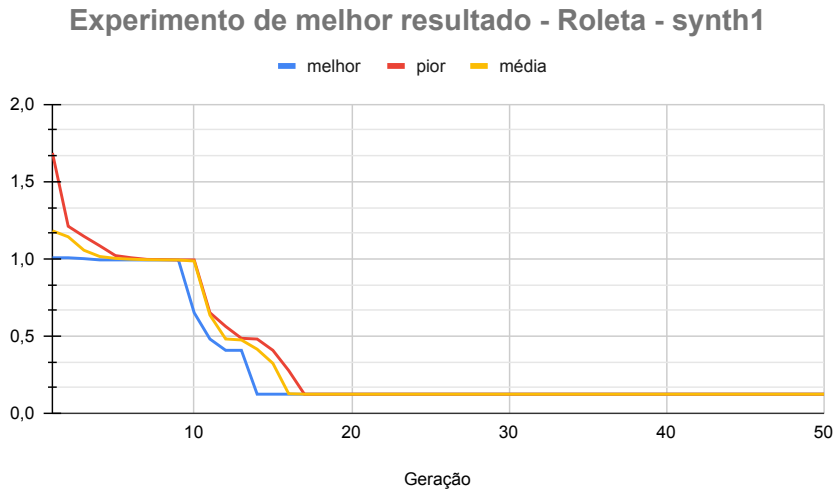

Figure 1: Convergência do método roleta de seleção na base synth1.

Pela Figura 1 notamos uma convergência antes de geração 20, onde o melhor individuo encontrado foi de fitness $1,03E-01$ ou mais exatamente de valor $0,102885512403084$.

Quando avaliamos os mesmos parâmetros para o método de seleção por torneio, percebemos que o melhor individuo $(2,27E-01)$, foi pior se comparado com o melhor da seleção anterior, contudo, quando consideramos a média de todas execuções, seus valores foram superiores.

Para o método de seleção por lexicase, temos uma convergência diferente dos anteriores, com melhorias presentes em gerações iniciais, intermediarias e também finais, indicando uma convergência mais lenta, o

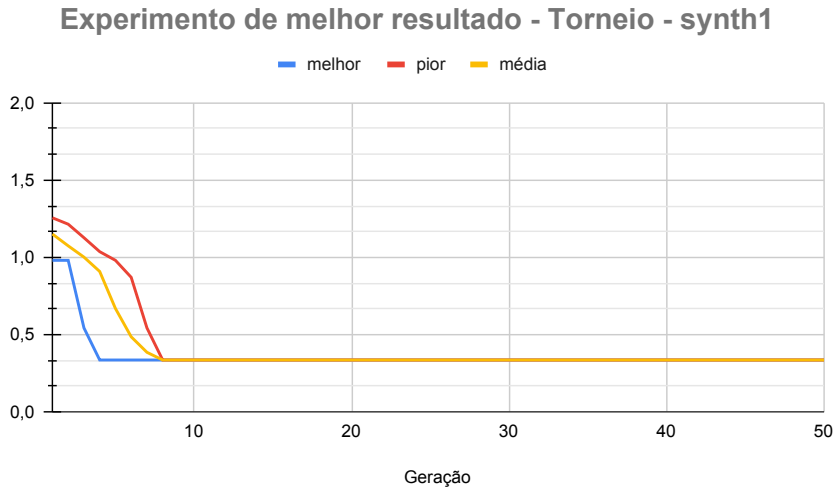

Figure 2: Convergência do método torneio de seleção na base synth1.

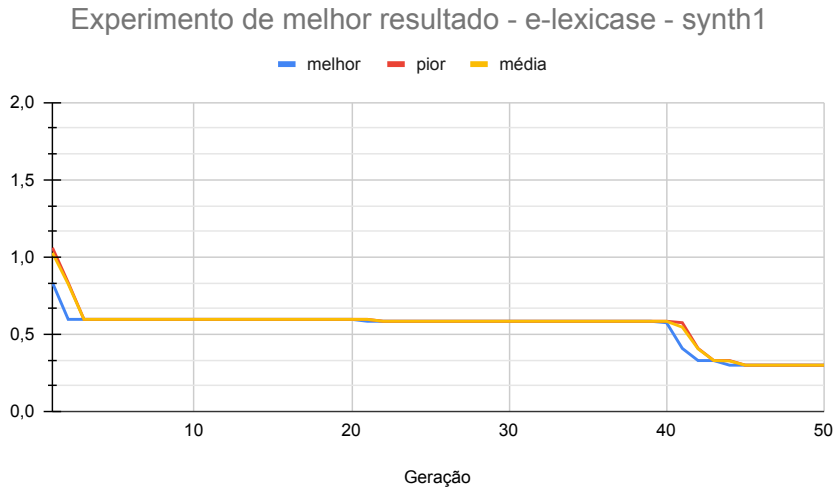

Figure 3: Convergência do método e-lexicase de seleção na base synth1.

que pode indicar uma oportunidade de melhorias ainda maiores através do aumento no número de gerações consideradas (de 50 para 100 por exemplo).

Buscando verificar a influencia das taxas de cruzamento e mutação nos resultados, optamos por variar tais parâmetros, para o método de seleção por roleta, conforme mostrado nas Figuras 4 e 5. Para o método lexicase e torneio não foi realizado a experimentação, devido à maior demanda de tempo para execução destas categorias.

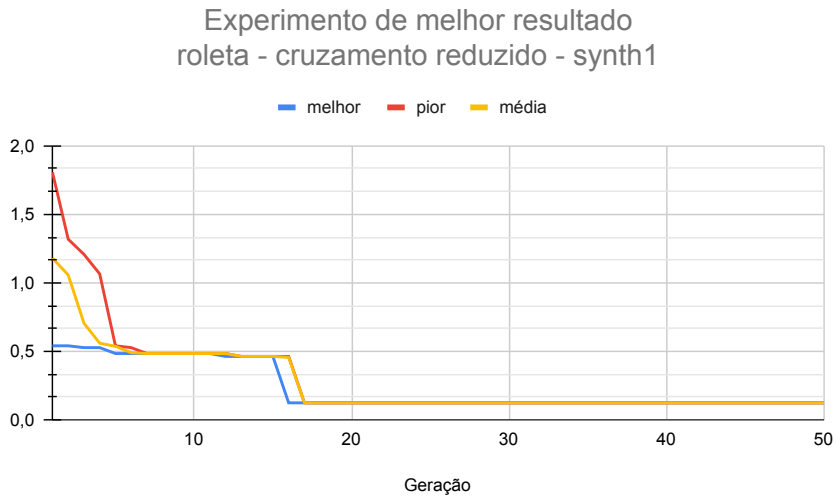

Figure 4: Convergência do método roleta com taxa de cruzamento reduzido.

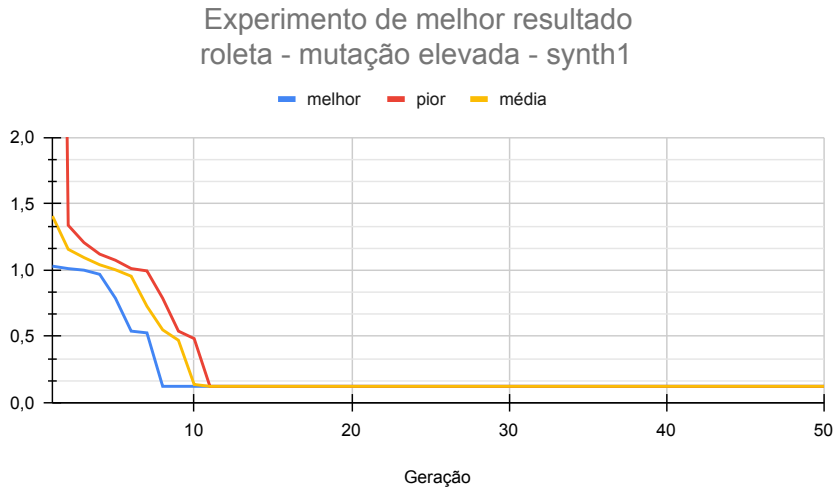

Figure 5: Convergência do método roleta com taxa de mutação elevada.

Em relação ao método de seleção por torneio, optamos por aumentar o tamanho dos torneio de 2 para 5, obtendo então o resultado como na Figura 6.

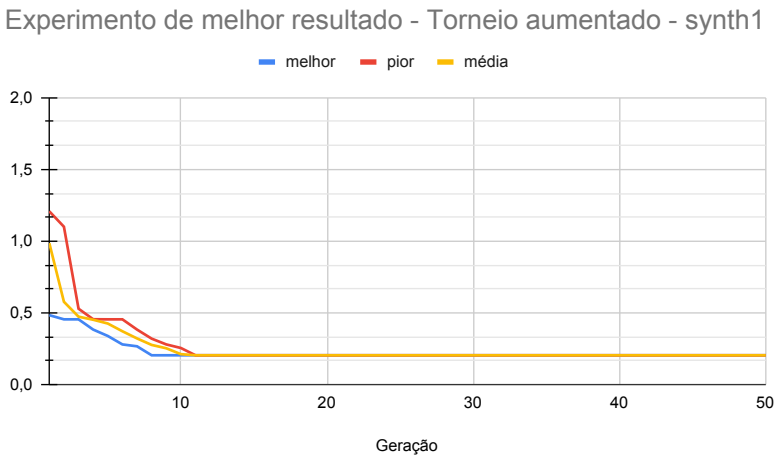

Figure 6: Convergência do método torneio com tamanho elevado.

Além disso, também foi realizado experimentos considerando o impacto quando removemos o elitismo, mostrado na Figura 7, e também quando incluímos a penalidade de *bloat*, mostrado na Figura **??**, ambos para o método de seleção por roleta, devido sua maior rapidez no resultado.

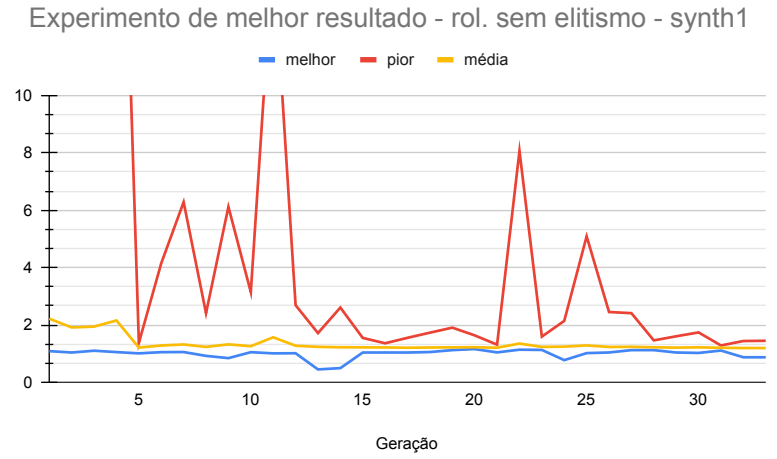

Figure 7: Convergência do método roleta sem elitismo.

Em especial no gráfico da Figura 7, foi necessário alterar a escala, uma vez que houve grande mudanças nos valores dos piores indivíduos. É possível notar que à ausência de elitismo, em certos momentos, promove a perda de boas soluções, gerando eventos de subida na convergência.

Para todos os experimentos realizados, o tamanho máximo do indivíduo foi fixado em 4. Devido a isso, na operação de cruzamento e mutação, a escolha do ponto de troca foi limitada até a profundidade 3. Assim, a altura máxima que um indivíduo pode ter durante a execução do algoritmo é 7, como recomendado pela documentação do TP-I.

Por fim, o conjunto das melhores funções obtidas para esta base, são mostradas na Tabela 2, seguindo a ordem relativa da Tabela 1 anteriormente mostrada.

seleção	melhor função
e-lexicase	(((x)*(x))*(((x)-((y)/(y)))*(x)))
roleta	(((x)*(x))*(((x)*(x))-(x)))
roleta	(((((x)*(x))-(y))+(y))*(((x)*(x))-(x)))
roleta	(((x)*(x))*((0.2)-((x)-((x)*(x)))))
roleta	Cancelado
roleta	((((x)*(((x)/(x))-((2)+(3))))+(((2)-((0.5)*(2)))-(x)))*((1)-((((y)-(y))*((x)+(x)))+(x))))
roleta	em execução
torneio	((((x)*(5))/((x)+(5)))*((x)*(5)))
torneio	((((((x)*(x))*(1))*((x)*(x)))-((y)-(x)))-((x)*(((x)*(x))*(1))))
torneio	(((((x)*(x))*((x)*(x)))+(((((0.5)-((0.2)*(y)))-(x))-(x))-(x)))+((((0.5)-((0.2)*(y)))-(x))-(x)))

Table 2: Funções obtidas para a base de dados synth1.

Após estes primeiros experimentos na base de dados Synth1, decidimos os parâmetros a serem utilizados para as demais bases de dados, considerando resultado e tempo de execução, uma vez que as maquinas utilizadas para execução dos experimentos não foram de alto desempenho (4gb ram).

3.2 Synth2

Ao realizar os testes na base de dados synth2, verificamos uma demanda maior de tempo, onde os resultados são mostrados na Tabela 3.

seleção	pop.	gerações	crossover	mut.	torneio	epsilon	elit.	bloat red.	médio	DP	melhor
e-lexicase	50	50	0.9	0.05		2	sim	não	8,00E-01	1,35E-01	7,10E-01
roleta	50	50	0.9	0.05			sim	não	4,56E+00	4,84E+00	7,31E-01
torneio	50	50	0.9	0.05	5		sim	não	9,71E+00	1,92E+01	5,48E-01

Table 3: Tabela com resultados para synth2.

As curvas de convergência do experimento de melhor resultado, deste conjunto, são mostrados nas Figuras 8, 9 e 10.

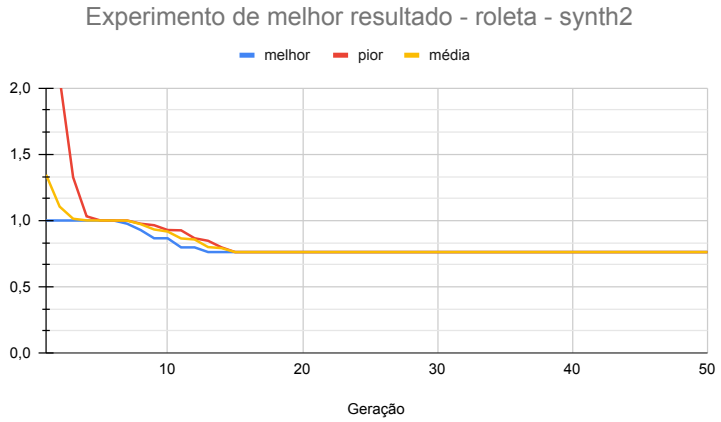

Figure 8: Convergência do método roleta em synth2.

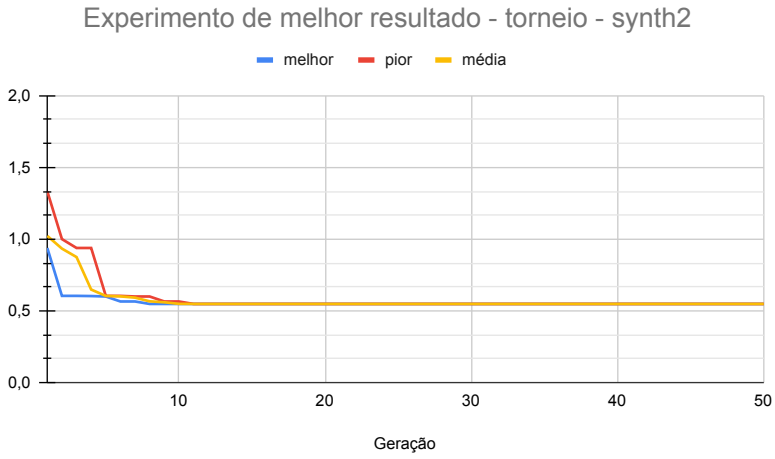

Figure 9: Convergência do método tornei em synth2.

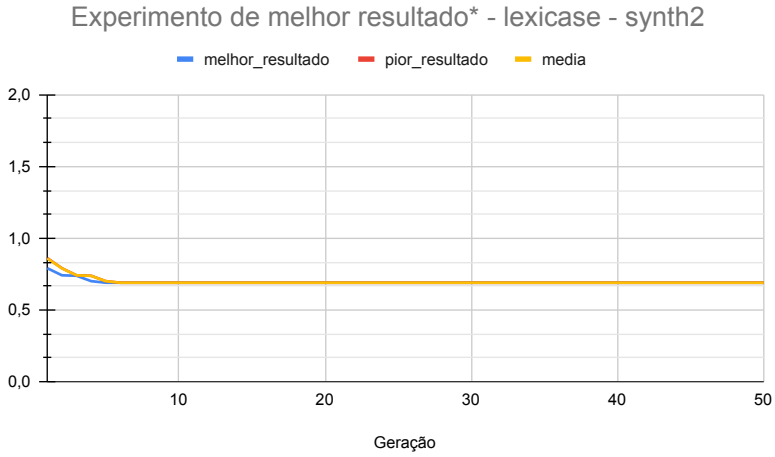

Figure 10: Convergência do método e-lexicase em synth2.

Nesta base verificamos que todos os valores de fitness apresentaram um valor maior em relação aos valores observados na synth1, possivelmente porque as funções utilizadas não consegue formar uma função matemática que represente o conjunto de dados de forma tão eficiente quanto na synth1. As formulações obtidas para synth2 são mostradas na Tabela 4.

seleção	**melhor função**
e-lexicase	(((((y)*(1))-(((x)/(3))-((y)*(0.1))))-((((x)/(3))-((y)*(0.1)))-((y)*(0.1))))*((((x)/(3))-((y)*(0.1)))-((y)*(0.1))))
roleta	(((((y)-(5))+((0.2)+(x)))/((4)+(5)))*(((y)-(5))+(((y)-(5))+(((y)-(5))+((0.2)+(x)))))
torneio	((((((x)*(x))/((y)+(5)))*(((y)/(y))+((y)-(3))))-((((((x)*(x))/((y)+(5)))/(5))+((y)-(3))))-((((((x)*(x))/((y)+(5)))/(5))+((((((x)*(x))/((y)+(5)))/(5))+((y)-(3)))))

Table 4: Funções obtidas para a base de dados synth2.

3.3 Concrete

Por fim, ao realizar os testes na base de dados concrete, obtemos os resultados demonstrados na Tabela 5.

seleção	pop.	gerações	crossover	mut.	torneio	epsilon	elit.	bloat red.	médio	DP	melhor
e-lexicase	50	50	0.9	0.05		2	sim	não	8,00E-01	1,35E-00	7,10E-01
roleta	50	50	0.9	0.05			sim	não	1,29E+00	3,27E-01	9,42E-01
torneio	50	50	0.9	0.05	5		sim	não	1,38E+00	3,84E-01	8,26E-01

Table 5: Resultados obtidos para a base de dados concrete.

As convergências para esta ultima base de dados são mostradas nas Figuras 11, 12 e **??**.

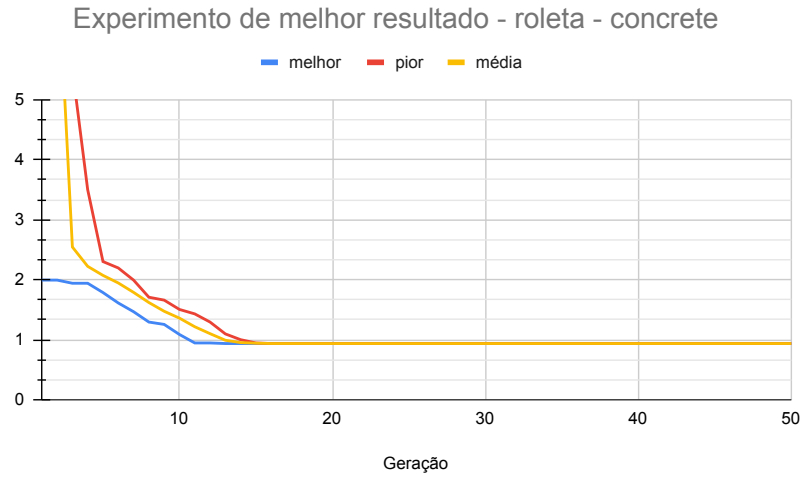

Figure 11: Convergência do método roleta em concrete.

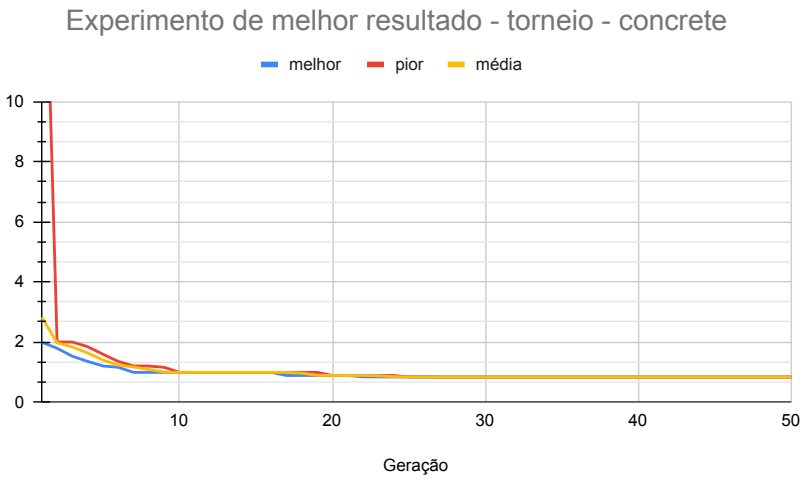

Figure 12: Convergência do método tornei em concrete.

Nesta base, assim como na synth2, percebemos uma média geral, de todas replicas, melhor para o método da roleta, no entanto, o torneio foi o método que trouxe o melhor individuo geral. Em relação as funções, nesta base as melhores encontradas são mostradas na Tabela 6.

seleção	melhor função
e-lexicase	
roleta	(((((((c)+(z))*((b)/(2)))/(((x)-(p))*((q)+(c))))+((c)-(z)))/((b)+((0.1)*(c))))+(((((((c)+(z))*((b)/(2)))/(((x)-(p))*((q)+(c))))+((c)-(z)))/((b)+((0.1)*(c)))))+((b)+(((((((c)+(z))*((b)/(2)))/(((x)-(p))*((q)+(c))))+((c)-(z)))/((b)+((0.1)*(c)))))))
torneio	(((x)*(0.1))+(b))

Table 6: Funções obtidas para a base de dados concrete.

4 Conclusão

Neste estudo, foi desenvolvido um algoritmo de Programação Genética (GP) para solucionar problemas de regressão simbólica. O algoritmo é parametrizado com o tamanho máximo do indivíduo, tamanho da população, número de gerações, probabilidade de mutação, probabilidade de cruzamento, tamanho do torneio, taxa de *epsilon* e *bloat*, e o uso de operadores elitistas.

Para determinar a melhor configuração de parâmetros, foi realizado um estudo individual de cada parâmetro. A seleção adequada desses parâmetros é fundamental, uma vez que eles podem influenciar significativamente no resultado final. Testes foram realizados para compreender o efeito de cada parâmetro nos valores finais de fitness, pressão seletiva e no número de indivíduos repetidos ao longo da evolução. Com base nos parâmetros selecionados, os resultados foram gerados para as três bases de dados disponíveis (synth1, synth2 e concrete). Os melhores resultados foram demonstrados a cada etapa, e podemos perceber que os melhores parâmetros podem variar dependendo do tempo disponivel para obter uma solução, ou seja, a seleção por roleta traz em média resultados bons, enquanto que o método de torneio, apesar de uma media geral inferior, possui resultados pontuais melhores que os obtidos pela roleta. Quanto ao lexicase, notamos que um epsilon muito baixo pode levar o algoritmo a ficar muito tempo em execução, podendo inviabilizar, caso se tenha uma perspectiva de produção.

References

[1] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming - An introduction*. Morgan Kaufmann, 1998.

[2] C. Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

[3] David E Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[4] Luiz R A Gonçalves, Hugo B B Pereira, and Robson M Coelho. Otimização de algoritmos genéticos utilizando paralelismo de nível de população. In *Anais do 13º Simpósio de Informática*, 2011.

[5] Md. Abir Hossain, Md. Kamrul Hasan, and Md. Mahmud. Machine learning based prediction of temperature and relative humidity for efficient mushroom cultivation. In *2020 IEEE Region 10 Symposium (TENSYMP)*, pages 712–715. IEEE, 2020.

[6] Xiaohu Huang, Linlin Zhang, Wenbo Xu, and Yuanzhang Sun. Machine learning-based regression modeling for forecasting of solar irradiation: A review. *Solar Energy*, 185:462–478, 2019.

[7] John R Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT press, 1992.

[8] T. McConaghy and F. Foldager. Symbolic regression versus neural networks: An empirical comparison. *Genetic Programming and Evolvable Machines*, 12(3):267–295, 2011.

[9] P.C. Oliveira. Aplicações da programação genética em problemas de regressão simbólica, 2019.

[10] Riccardo Poli, William B Langdon, and NF McPhee. A hybrid approach to the automatic synthesis of analog electrical circuits using genetic programming. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 2458–2465. IEEE, 2008.

[11] Riccardo Poli, William B Langdon, and Nicholas F McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises UK Ltd, 2008.

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

[13] Lee Spector, Thomas Helmuth, and James Matheson. Improving genetic programming performance using lexicase selection. *Genetic programming and evolvable machines*, 13(2):165–191, 2012.

[14] L. Vanneschi and M. Tomassini. A survey of symbolic regression and its applications to system identification and control. *IEEE Transactions on Evolutionary Computation*, 15(3):326–344, 2011.

[15] Darrell Whitley and Keith E Mathias. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel computing*, 20(12):1631–1646, 1994.

[16] Mingyuan Zhang, Han Zheng, and Xiangmo Zhao. Research on mathematical modeling and simulation of heterogeneous vehicle fuel consumption based on machine learning algorithm. *Mathematical Problems in Engineering*, 2019, 2019.