Abordagem Genética para Regressão Simbólica

Jackson Nunes Silva[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brasil

jacksonns@ufmg.br

Resumo. *O objetivo desse trabalho é implementar um algoritmo baseado em programação genética para resolver o problema da regressão simbólica, bem como em analisar os impactos dos parâmetros na qualidade da solução fornecida. Para isso, são utilizadas as técnicas de evolução gramatical estruturada, seleção por torneio e por roleta, geração de população baseada no método hamped half and half e substituição de população steady-state.*

1. Introdução

O problema da Regressão Simbólica consiste em encontrar uma expressão matemática que melhor se ajusta a um conjunto de amostras contendo o valor de n variáveis de entrada e um valor de saída, através da combinação de funções básicas (adição, subtração, multiplicação e divisão), funções trigonométricas, logaritmos, etc. Em suma, é uma generalização do conceito de regressão linear ou regressão polinomial.

Por ser um problema complexo de ser resolvido, uma das abordagens utilizadas na literatura visando buscar uma solução aproximada é a utilização da programação genética. A execução se baseia em criar uma população inicial de indivíduos, em que cada indivíduo representa uma possível solução para o problema (no caso são funções, geralmente representadas por árvores), em seguida todos os indivíduos tem sua *fitness* avaliada, um subconjunto é selecionado e são aplicados operadores genéticos entre eles (cruzamento e mutação), gerando uma nova população. Esse processo se repete até que um número máximo de gerações tenham sido criadas ou algum outro critério de parada tenha sido satisfeito.

2. Implementação

O algoritmo foi implementado na linguagem de programação Python 3.8 e dividido nos seguintes arquivos:

- **genotype.py**: Contém a definição do genótipo (representado por listas e árvores) baseado em uma gramática estruturada.
- **population.py**: Contém duas classes representando o indivíduo e a população.
- **selection.py**: Contém a implementação dos algoritmos de seleção (Torneio e Roleta).
- **geneticSR.py**: Contém a implementação da programação genética, bem como uma classe para controle de escrita dos resultados.
- **main.py**: Chamada principal do programa.

A execução do programa deve ser feita através do seguinte comando, onde –config é o argumento contendo o caminho até o **arquivo json** com os parâmetros da programação genética e –results é o nome do diretório onde devem ser escritos os arquivos contendo os resultados da execução:

```
python main.py --config config_path --results results_dir
```

Um modelo que pode ser seguido para o arquivo json de configuração é o seguinte, onde o campo *name* de *selection* pode ser “tournament” ou “roulette”.

```
{
    "var_num": 2,
    "population_size": 250,
    "generations": 200,
    "train_file": "../datasets/synth1/synth1-train.csv",
    "test_file": "../datasets/synth1/synth1-test.csv",
    "selection":{
        "name": "tournament",
        "args": {
            "k": 2
        }
    },
    "mutation_prob": 0.05,
    "crossover_prob": 0.90,
    "elitism": true,
    "verbose": false
}
```

A seguir, são comentadas as decisões de implementação a respeito de cada etapa da programação genética, como representação dos indivíduos, método de seleção, substituição da população, etc.

2.1. Indivíduos

A representação dos indivíduos segue uma regra de construção gramatical estruturada, introduzida em [Lourenço et al. 2016], no qual o genótipo do indivíduo possui um conjunto de genes referentes a cada não terminal da gramática. Cada gene é representado por uma lista, cujo tamanho é o número máximo de expansões do respectivo não terminal e os valores são inteiros que identificam a expansão. Desse modo, o genótipo do indivíduo é representado por uma lista de listas, que em seguida é mapeado para uma árvore. Para a solução do problema, foi adotada a seguinte regra de produção:

⟨*expr*⟩ ::= (⟨*expr_lvl1*⟩⟨*op*⟩⟨*expr_lvl1*⟩)

⟨*expr_lvl1*⟩ ::= (⟨*expr_lvl2*⟩⟨*op*⟩⟨*expr_lvl2*⟩) | var | const

⟨*expr_lvl2*⟩ ::= (⟨*expr_lvl3*⟩⟨*op*⟩⟨*expr_lvl3*⟩) | var | const

⟨*expr_lvl3*⟩ ::= (⟨*var*⟩⟨*op*⟩⟨*var*⟩) | ⟨*preop*⟩(⟨*var*⟩) | var | const

⟨*op*⟩ ::= + | - | * | /

⟨*preop*⟩ ::= sin | cos

⟨*var*⟩ ::= x | y

⟨*const*⟩ ::= [-1, 1]

Como é possível perceber, foram adotados três níveis de recursão e apenas no terceiro é possível expandir uma das duas funções trigonométricas disponíveis (seno e cosseno). Foi decidido estabelecer essa restrição para que as funções não fossem poluídas com operadores desnecessários, o que melhorou os resultados obtidos para os conjuntos de dados considerados. Como o nível máximo de recursão é 3, a altura máxima da árvore gerada também é restringida para no máximo 6.

2.2. População

Para a inicialização da população, é utilizada uma combinação dos métodos de *grow*, para uma metade da quantidade de indivíduos e *full* para a outra metade, em uma adaptação da técnica *hamped half and half*. O método *grow* gera um genótipo totalmente aleatório, de modo que a árvore se expande recursivamente sem restrições. Já o método *full* adotado escolhe uma profundidade de recursão aleatoriamente para um indivíduo e expande apenas nós do conjunto de funções até que a profundidade tenha sido atingida.

2.3. Fitness

A fitness dos indivíduos é calculada pelo erro quadrático médio normalizado (NRMSE), o que facilita a comparação experimental entre diferentes conjuntos de dados. O NRMSE é calculado pela seguinte fórmula:

$$f(Ind) = \sqrt{\frac{\sum_{i=i}^{N}(Eval(Ind, x_i) - y_i)^2}{\sum_{i=i}^{N}(y_i - mean(Y))^2}} \tag{1}$$

Nesse caso, $Eval(Ind, x_i)$ é o valor obtido pelo indivíduo na instância x_i, y_i é o valor esperado, N é o número de instâncias e $mean(Y)$ é a média dos valores esperados das instâncias.

2.4. Seleção

Os dois tipos de seleção implementados foram por torneio e roleta. A seleção por torneio recebe um parâmetro k, que é o número de indivíduos aleatoriamente selecionados da população e escolhe aquele cuja fitness é a menor entre eles. Já a seleção por roleta seleciona os indivíduos com probabilidade proporcional à fitness, mas como o problema é de minimização, é feita inversão da função de fitness para o cálculo das probabilidades, através da fórmula $f(Ind) = max(f) - f(Ind)$, onde $max(f)$ é o valor da maior fitness. Já a seleção por *lexicase* proposta não foi implementada em tempo hábil.

2.5. Operadores Genéticos

Os operadores genéticos foram implementados conforme a definição de cruzamento e mutação em Grámaticas Estruturadas [Lourenço et al. 2016]. Para o cruzamento, é gerada uma máscara, que é formada por uma lista de bits do tamanho do número de genes, no qual cada bit determina se seu respectivo gene será trocado entre os dois indivíduos selecionados. A figura 1, retirada do artigo de referência, mostra um exemplo dessa operação. Já a mutação é feita trocando um único inteiro da lista correspondente a um gene escolhido aleatoriamente.

Após a aplicação dos operadores genéticos, dois filhos são gerados para o cruzamento, e apenas um é gerado para a mutação. No caso do cruzamento, são colocados na

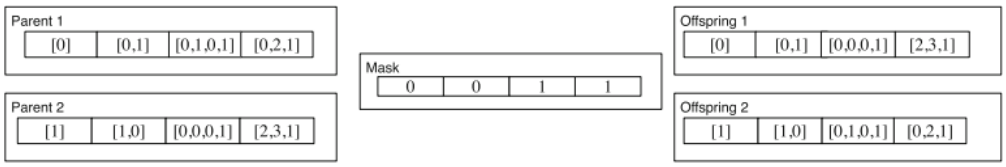

Figura 1. Exemplo de cruzamento

nova geração os dois melhores indivíduos entre os 2 pais e os 2 filhos. Já para a mutação, o filho gerado sempre é colocado na nova geração. Essa decisão foi tomada arbitrariamente após verificar uma convergência acelerada da população nos testes experimentais.

3. Experimentação

A condução dos testes experimentais se deu através da análise inicial da influência dos parâmetros na qualidade da solução utilizando a base de dados *synth1*. Entre os parâmetros avaliados, foram considerados o tamanho da população, a probabilidade dos operadores genéticos, o uso de elitismo e o método de seleção.

A fim de obter uma conclusão consistente, cada teste foi executado 30 vezes e foi coletada a média das seguintes métricas para cada geração: fitness do melhor indivíduo, fitness média, número de indivíduos únicos e número de filhos melhores que os pais. O objetivo foi gradativamente otimizar os parâmetros e verificar alguma possível melhoria no algoritmo.

3.1. Dataset Synth1

Inicialmente, foram testados 4 tamanhos de população distintos (50, 100, 250 e 500), a fim de analisar a média da melhor fitness em cada geração. Para isso, foi fixada uma taxa de cruzamento de 0.9, taxa de mutação de 0.05, uso de elitismo e seleção por torneiro de tamanho 2 ao longo de 300 gerações.

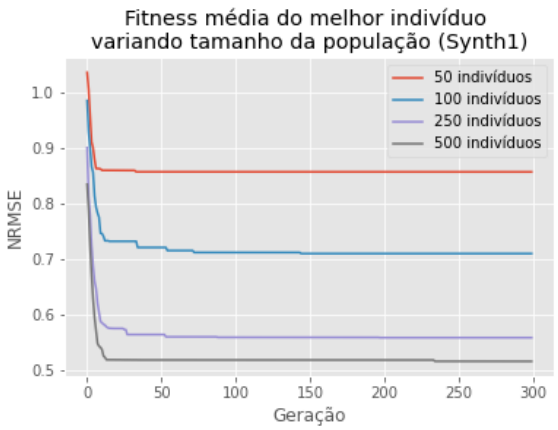

Figura 2. Evolução por nº de indivíduos

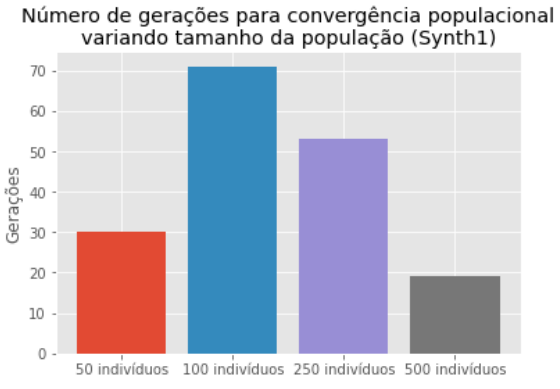

Figura 3. Nº de gerações para convergência populacional

A Figura 2 mostra o resultado dos testes. É possível perceber que a população de tamanho 500 obteve o melhor resultado, no entanto, converge extremamente rápido (em

menos de 20 gerações, como é possível observar na Figura 3). Portanto, é interessante considerar a população de tamanho 250, que possui um resultado similar, converge com menor velocidade e tem um custo de tempo menor. É possível perceber também que a convergência ainda está acelerada, mas isso será discutido posteriormente.

Em seguida, foi fixada o tamanho da população em 250 e número de gerações igual a 200 e foram feitos testes com as probabilidades dos operadores genéticos. Foram consideradas três combinações (pc = 0.90 e pm = 0.05; pc = 0.75 e pm = 0.20; pc = 0.60 e pm = 0.30). A Figura 4 mostra os resultados, de onde é possível perceber que a melhor combinação foi a intermediária (pc = 0.75 e pm = 0.20).

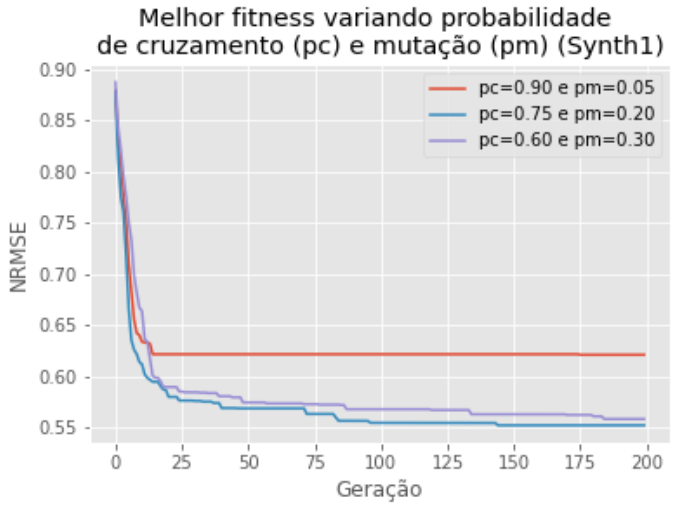

Figura 4. Variação dos operadores genéticos

O próximo passo foi testar a eficácia do uso de elitismo, então foram mantidos os mesmos parâmetros escolhidos anteriormente e comparado a execução com e sem elitismo. Na Figura 5 é possível perceber que o elitismo melhora ligeiramente as soluções.

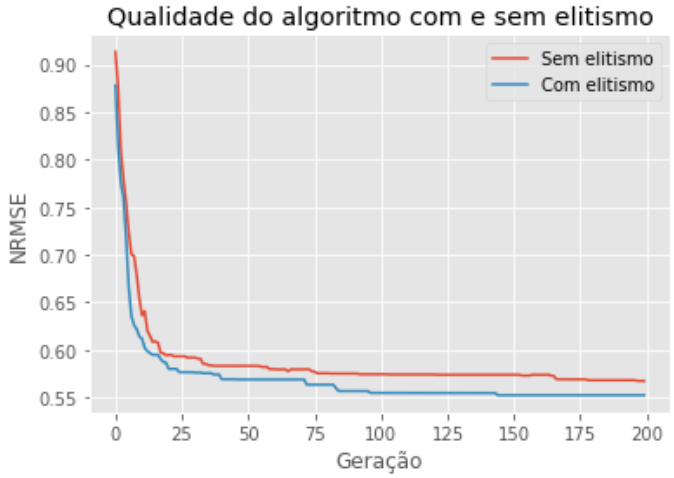

Figura 5. Impacto do elitismo

Como mencionado anteriormente, um problema visto no decorrer dos experimentos era a alta velocidade de convergência da população, cuja diversidade (número de in-

divíduos únicos) diminuía bruscamente nas primeiras gerações. Para amenizar esse problema, foi adicionada uma condição de substituição na qual quando filhos iguais eram gerados, apenas um deles era introduzido na nova geração. Após fazer isso, houve uma melhora significativa na diversidade populacional. Assim, os testes foram continuados nessa nova versão do programa, agora alterando a técnica de seleção (diferença entre torneio e roleta).

A Figura 6 mostra um resultado inesperado: para o algoritmo implementado, a seleção por roleta tem resultados bem melhores que por torneio. Portanto, a otimização final dos parâmetros ficou fixada em: população de tamanho 250, probabilidade de cruzamente de 0.75, probabilidade de mutação de 0.20, seleção por roleta e uso de elitismo por pelo o menos 200 gerações.

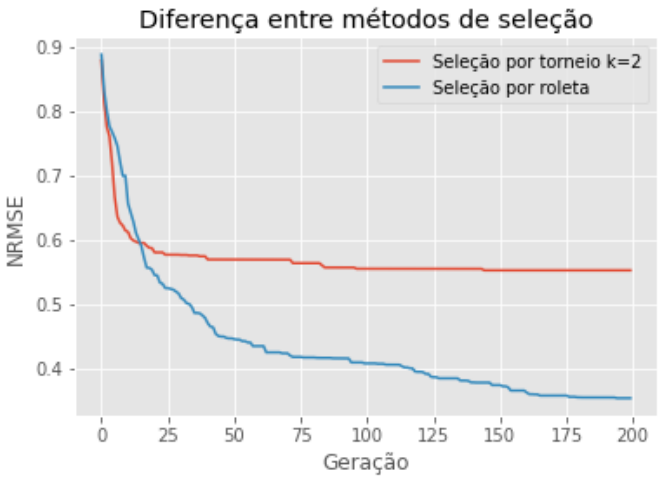

Figura 6. Métodos de Seleção

Com esses valores estipulados, o melhor indivíduo foi encontrado com uma fitness equivalente a 0.11 no conjunto de treino e NRMSE de 0.08 no conjunto de teste. Sua expressão depende apenas de x e é representada pela seguinte fórmula:

$$f(x, y) = x^2 \cdot (x^2 + 0.12669 - x) \tag{2}$$

3.2. Dataset Synth2

Testes não foram realizados em tempo hábil.

3.3. Concrete

Testes não foram realizados em tempo hábil.

4. Conclusões

Esse trabalho apresentou bem o poder da programação genética em aproximar soluções de problemas inerentemente difíceis. Aplicado no contexto da Regressão Simbólica, foi possível observar e experimentar diversas combinações de parâmetros, bem como analisar seu impacto na qualidade das soluções obtidas.

Em tempo hábil, apenas foi possível otimizar os parâmetros no primeiro conjunto de testes (synth1), através do qual foi possível observar inicialmente um problema na implementação do algoritmo, visto que a população convergia muito rapidamente para um valor de fitness. Após o problema ser corrigido, foi possível notar que nem sempre um grande número de indivíduos fornece uma boa solução, já que pode favorecer uma convergência prematura além de aumentar a complexidade de tempo. Foi possível notar também que o elitismo favoreceu boas soluções e, surpreendentemente, a seleção por roleta funcionou bem melhor no algoritmo implementado.

Em suma, os resultados de fitness para a primeira base de dados foram relativamente satisfatórios, com a melhor fitness tendo média de 0.35 na geração 200. No entanto, ainda faltam análises a serem feitas, como os testes de parâmetros nas outras bases de dados (synth2 e concrete), análise da variação do parâmetro k na seleção por torneio, análise do impacto da alteração da gramática (como adição de um novo nível de recursão, possibilidade de funções trigonométricas, etc). De toda forma, foi uma excelente oportunidade de aprender na prática o funcionamento de algoritmos baseados em programação genética.

Referências

Lourenço, N., Pereira, F. B., and Costa, E. (2016). Unveiling the properties of structured grammatical evolution. *Genetic Programming and Evolvable Machines*, 17:251 – 289.