Trabalho Prático Computação Natural - Regressão Simbólica

Diego Matos [*1]

1. Introdução

A Programação Genética (GP) é uma técnica de otimização baseada em algoritmos genéticos que busca soluções para problemas complexos através da evolução de programas. Ela utiliza conceitos inspirados na biologia e na teoria da evolução, como reprodução, mutação, seleção natural e herança genética. A GP é aplicada em diversos domínios, incluindo aprendizado de máquina, otimização, programação automática, entre outros.

Neste trabalho, é apresentado uma implementação de um algoritmo de programação genética e a análise de seu desempenho em relação a diferentes parâmetros e técnicas, como geração de população, função de fitness, operadores de seleção e probabilidade de cruzamento e mutação. A implementação é baseada em árvores de expressão, que representam soluções candidatas para o problema a ser resolvido. Além disso, realizamos experimentos para analisar o impacto das variações destes parâmetros na qualidade das soluções obtidas e na velocidade de convergência do algoritmo.

O objetivo deste trabalho é compreender os aspectos fundamentais da programação genética e explorar como as diferentes técnicas e parâmetros afetam seu desempenho. Através dos resultados experimentais, esperamos fornecer insights valiosos para o projeto e a aplicação de algoritmos de programação genética em problemas reais.

2. Implementação

O código gerado define a estrutura básica de uma árvore de expressão e as operações necessárias para manipular e avaliar essas árvores. A classe Node é a unidade fundamental da árvore de expressão e pode representar variáveis, constantes ou operadores.

As variáveis e constantes são chamadas de ”terminais” e os outros operadores são chamados de ”não-terminais”. A implementação atual suporta terminais como ”X1”, ”X2” e ”const”, e operadores como ”+”, ”-”, ”*”, ”/”, ”sin”, ”cos”, ”exp” e ”ln”.

A classe Node fornece métodos para criar e manipular árvores de expressão, como ”evaluate”, ”copy”, ”subtree”, ”replace subtree”, ”count nodes”, ”get depth” e outros. Esses métodos permitem a criação e manipulação de árvores de expressão (indivíduos) durante a evolução.

Foram implementados técnicas de geração de população conhecidas em programação genética, são eles, o ”ramped half”, ”grow” e ”full”, que são utilizados para gerar indivíduos iniciais com diferentes propriedades, como profundidade e quantidade de nós. Esses métodos são importantes na fase de inicialização de uma população de árvores de expressão e ajudam a garantir a diversidade na população.

A função de fitness utilizada foi a ”mean absolute error”. A função recebe como entrada dois vetores NumPy com os valores verdadeiros e previstos, respectivamente. Em seguida, a função calcula a diferença absoluta entre cada par de valores (um valor verdadeiro e seu correspondente previsto) e tira a média desses valores absolutos para obter o MAE.

Foram implementados também operadores de seleção por torneio e por roleta. A seleção por roleta é uma abordagem que usa um mecanismo de seleção proporcional, em que a probabilidade de um indivíduo ser selecionado é proporcional à sua adequação relativa em relação aos outros indivíduos da população. Em outras palavras, indivíduos mais aptos têm uma probabilidade maior de serem selecionados para reprodução do que indivíduos menos aptos. A seleção por torneio é uma abordagem de seleção de indivíduos que envolve a realização de torneios em que vários indivíduos são selecionados aleatoriamente da população e comparados com base em sua adequação (fitness). O indivíduo mais adequado é selecionado para reprodução e seu genoma é copiado para a próxima geração.

A função ”mutate” tem como objetivo aplicar uma mutação aleatória em uma árvore de expressão que representa um indivíduo. A mutação aleatória é uma das principais operações usadas em GP para criar novas soluções, e consiste em fazer pequenas alterações na árvore para criar uma nova solução. Essas alterações podem envolver a remoção ou adição de nós, a troca de operadores ou variáveis, entre outras possibilidades. A implementação define um tamanho máximo de sub-árvore para ser aleatoriamente inserida no indivíduo, e essa sub-árvore é construída a partir do método ramped-half and half, que gera árvores com tamanhos variados entre 1 e 7 níveis, com uma distribuição uniforme de tamanhos. Como é requerido que cada indivíduo tenha no máximo 7

níveis de profundidade, é verificado se a profundidade do indivíduo gerado é menor ou igual a 7. Se for, o indivíduo gerado é adicionado na nova população. Este processo pode ser ilustrado nas imagens:

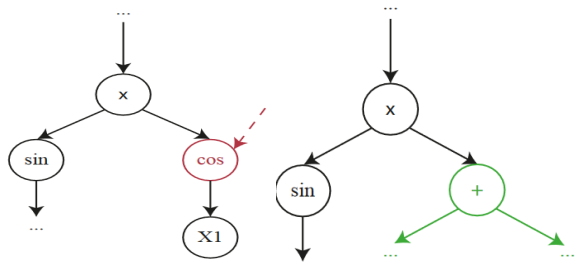

Figure 1. Representação da mutação:seleção aleatória de um nó do indivíduo, seguida da substituição do nó por uma sub-arvore aleatória.

Os cruzamentos seguiram a seguinte estratégia: Dado dois indivíduos pais, iremos selecionar aleatoriamente um nós de mesma profundidade entre os dois indivíduos. Ou seja, caso o só selecionado seja de profundidade 7, então significa que os dois pais devem ter tamanho 7 e apenas um nó folha será trocado; caso seja selecionado um nó de profundidade 6, então uma sub-arvore de no máximo tamanho 2 será selecionada. Essa estratégia garante que o tamanho dos filhos gerados não ultrapasse o tamanho máximo definido.

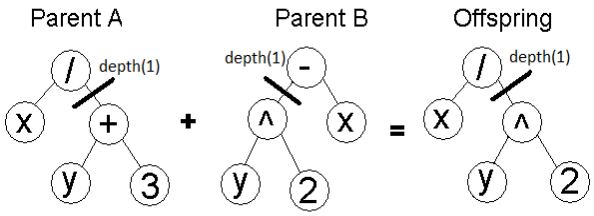

Figure 2. Representação do cruzamento: sub-arvore do pai 1 é escolhido com profundidade 1, portanto o a segunda sub-arvore também deve ter a mesma profundidade.

Após a execução do algoritmo são armazenadas as estatísticas por geração sobre: a quantidade de indivíduos iguais, melhor fitness, melhor indivíduo, pior indivíduo, média da fitness, total de cruzamento e total de mutações.

3. Experimentação

Nesta seção serão mostrados experimentos e suas repectivas análises. Os valores padrão para cada experimento foram:

Table 1. Parâmetros

Parâmetro	Valor
Taxa de mutação	0.6
Taxa de crossover	0.2
Tamanho da população	100
Gerações	100
Elitismo	Verdadeiro
Tamanho do torneio	2

Ao variar a probabilidade de cruzamento é possível controlar a frequência com que os indivíduos são cruzados durante a geração da nova população. Isso pode afetar a diversidade da população e também a rapidez com que a população converge para soluções boas.

Quando a probabilidade de cruzamento é alta, há uma maior probabilidade de que os indivíduos sejam cruzados, o que pode ajudar a aumentar a diversidade da população, mas também pode levar a uma convergência mais lenta para soluções boas. Por outro lado, quando a probabilidade de cruzamento é baixa, há uma menor probabilidade de que os indivíduos sejam cruzados, o que pode reduzir a diversidade da população, mas também pode levar a uma convergência mais rápida para máximos locais.

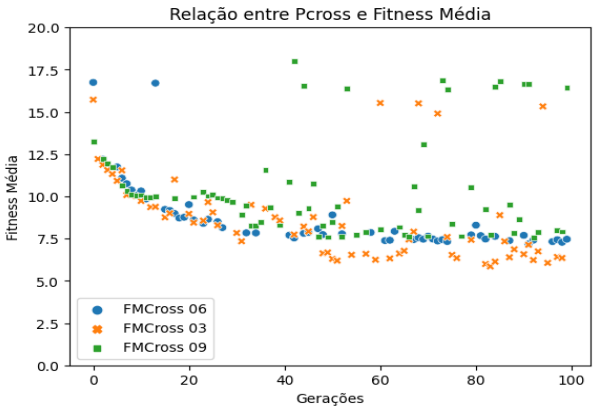

Figure 3. Variação da fitness média por gerações ao alterar a probabilidade de cruzamento dos indivíduos aplicado no conjunto de dados 1.

É possível observar que para valores de probabilidade de cruzamento muito alto a diversidade dos indivíduos aumenta, já que novos indivíduos são gerados a partir de combinações diferentes dos indivíduos existentes. No entanto, isso pode levar a uma maior oscilação na fitness média, uma vez que os novos indivíduos gerados podem ter características muito diferentes dos indivíduos originais.

Por outro lado, com uma probabilidade baixa de cruzamento, a diversidade dos indivíduos é menor, pois há uma menor

chance de que os indivíduos sejam cruzados e gerem novos indivíduos com características diferentes. Isso pode levar a uma convergência mais rápida para mínimos locais na fitness média, uma vez que os indivíduos são mais semelhantes e têm mais probabilidade de compartilhar as mesmas características.

Assim como na probabilidade de cruzamento, a escolha da probabilidade de mutação ideal em um algoritmo genético pode afetar significativamente seu desempenho. Quando a probabilidade de mutação é muito baixa, pode ocorrer uma convergência prematura para mínimos locais, pois a população não será capaz de explorar completamente o espaço de solução em busca de soluções de alta qualidade. Por outro lado, se a probabilidade de mutação for muito alta, pode ocorrer um alto nível de perturbação da população, o que pode prejudicar a convergência da população e tornar difícil a identificação de soluções de alta qualidade.

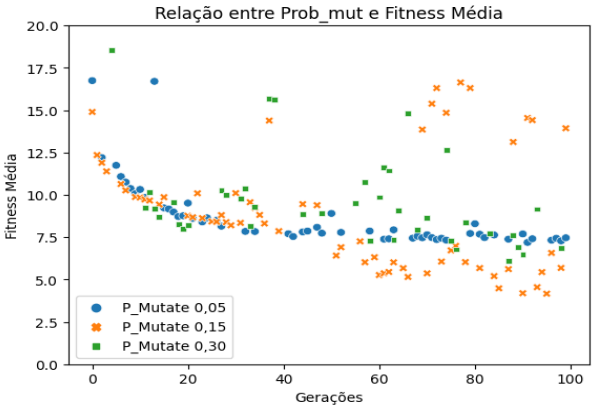

Figure 4. Variação da fitness média por gerações ao alterar a probabilidade de mutação dos indivíduos aplicado no conjunto de dados 1.

Na figura é possível ver que o uso de uma probabilidade de mutação muito baixa pode gerar uma convergência prematura do algoritmo, pois a população apresenta pouca diversidade e a busca fica restrita a soluções locais. O gráfico mostra uma estabilização prematura da fitness média, sem alcançar valores melhores. Por outro lado, com o aumento da probabilidade de mutação, a fitness média começa a oscilar de forma mais ampla, o que pode ser indicativo de que a população está explorando o espaço de solução em busca de soluções mais diversas e de alta qualidade. No entanto, valores muito altos podem prejudicar a convergência e dificultar a identificação de soluções de alta qualidade. Assim, é importante encontrar um equilíbrio adequado entre a exploração do espaço de solução e a convergência (exploration x exploitation). O uso de valores intermediários de probabilidade de mutação pode ajudar a alcançar esse equilíbrio, permitindo que a população explore o espaço de solução de forma mais ampla e ao mesmo tempo convergindo para soluções de alta qualidade.

Uma população maior pode ajudar a encontrar melhores soluções mais rapidamente, pois há uma maior probabilidade de encontrar bons indivíduos aleatoriamente. Além disso, um conjunto maior de soluções candidatas pode aumentar a eficácia das operações de crossover e mutação, aprimorando a qualidade dos indivíduos gerados nas gerações subsequentes. Por outro lado, ao utilizar uma população pequena, podemos saturar o modelo rapidamente, já que os indivíduos tendem a convergir para um mínimo local e acabam se tornando muito semelhantes uns aos outros. Isso pode ser observado na seguinte figura:

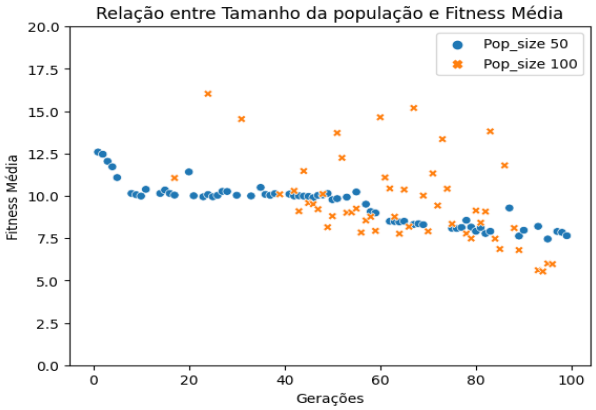

Figure 5. Variação da fitness média por gerações a partir do tamanho da população de indivíduos utilizada aplicados no conjunto de dados 1.

Com o uso de uma população muito pequena, o algoritmo converge rapidamente para uma solução local, enquanto com o uso de uma população maior, uma solução melhor é encontrada aleatoriamente, o que deve contribuir para uma convergência mais eficiente em gerações futuras.

Como esperado, o operador seletivo que define a probabilidade de seleção com base na aptidão (fitness) obtida e utiliza essas estatísticas globais teve o pior desempenho. É comum que a seleção por roleta acabe selecionando previamente uma maioria de indivíduos bons, gerando muita pressão seletiva logo no início da execução. Isso leva o algoritmo a convergir precocemente para um mínimo local.

Por outro lado, a seleção por torneio tem um controle maior da pressão seletiva por meio do parâmetro K, que define a quantidade de indivíduos que serão selecionados aleatoriamente para entrar no torneio. Com isso, é possível afirmar que a diversidade utilizando o método de seleção por torneio tende a ser maior quanto menor o valor de K. Isso contribui para o equilíbrio entre exploração e explotação, e é possível observar uma melhora devido a isso na figura abaixo.

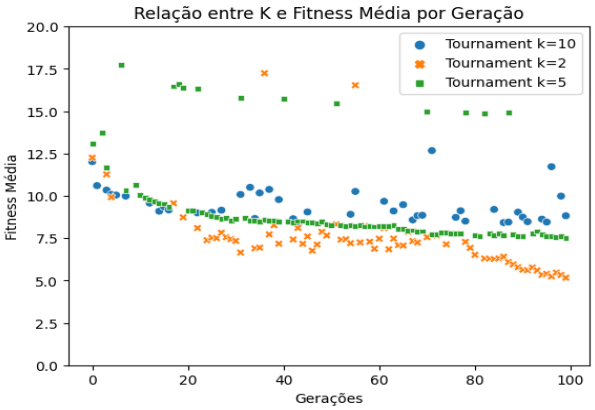

Figure 6. Variação da fitness média por gerações ao alterar o valor de K.

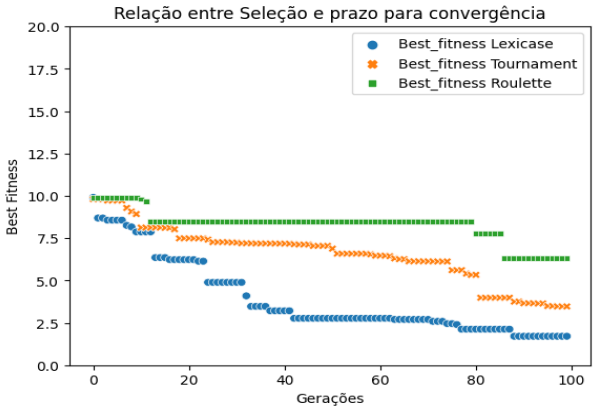

Figure 7. Melhor indivíduo por geração em cada tipo de seleção usada aplicado no conjunto de dados 1 (elitismo = True).

É possível observar que o valor de K define o equilíbrio entre pressão seletiva e diversidade. Com um valor de K muito elevado, teremos uma pressão seletiva muito grande e, consequentemente, mais indivíduos iguais serão selecionados (como no caso em que k=10). No entanto, foi impedido que indivíduos iguais fossem selecionados, para que o gráfico não gerasse apenas um indivíduo. Para valores menores de K, a diversidade aumenta, o que faz com que o k=2 selecione diferentes indivíduos.

A seleção lexicase é uma técnica de seleção de pais em algoritmos genéticos que visa preservar a diversidade na população e evitar a convergência prematura. Essa abordagem considera a performance dos indivíduos em múltiplos casos de teste, em vez de apenas um valor de aptidão agregado. Dessa forma, a seleção lexicase enfatiza a importância de resolver bem todos os casos de teste e promove a evolução de soluções especializadas para diferentes partes do problema.

Com isso é possível selecionar indivíduos que se saem muito bem em alguns casos de teste mesmo que em outros eles sejam piores. Isso ajuda a preservar diversidade e ainda mantém indivíduos ”uteis” distribuídos no espaço de busca. Com o uso do epislon para definir o intervalo de erro é possível manter intervalos que evoluem junto com a população de forma a ficar cada vez mais ”apertada” e gerar indivíduos dentro de faixas de erro cada vez menores.

Dentre os métodos de seleção utilizados o que performou melhor foi a epislon-lexicase, isso se deve a todas as suas características citadas e principalmente ao fato deste mecanismo manter a diversidade sem manter indivíduos ”absurdos de ruim” na população.

4. Conclusão

Em conclusão, o desenvolvimento de algoritmos genéticos para resolver problemas complexos envolve uma série de considerações e ajustes de parâmetros, como a probabilidade de cruzamento, probabilidade de mutação, tamanho da população e o método de seleção utilizado. Os resultados experimentais apresentados nesta análise demonstraram que encontrar um equilíbrio adequado entre exploração e explotação é crucial para garantir a convergência e a obtenção de soluções de alta qualidade.

A seleção epsilon-lexicase mostrou-se a melhor opção dentre os métodos de seleção analisados, devido à sua capacidade de preservar a diversidade na população e evitar a convergência prematura. O método leva em consideração a performance dos indivíduos em múltiplos casos de teste e promove a evolução de soluções especializadas para diferentes partes do problema.

A maior dificuldade encontrada foi implementar os operadores de mutação e cruzamento. Durante a etapa de implementação foram testados inúmeras formas de implementar o cruzamento e a mutação. Foi utilizando cruzamento com elitismo, cruzamento clássico, cruzamento penalizando infinitamente filhos gerados com profundidade maior que 7, cruzamento que gera apenas indivíduos de tamanho 7 ou menos e várias variações da mutação com a mesma ideia. Percebi também que o esses operadores acabavam gerando indivíduos muito diferentes ao trocar um único nó da ávore (muito comum se trocar o operador para

um exp, a fitness tende a ir para números extremamente altos). Muito provável que o uso de árvore acaba gerando uma dificuldade de localidade dos indivíduos, o que dificulta a convergência do algoritmo. Para tentar solucionar isso foram modificados os operadores utilizados diversas vezes, sem o uso de "/" melhora drasticamente a localidade do indivíduo, porém percebi que o sem este operador as funções não podem ser descrevidas, tentei também diminuir a probabilidade deste operador aparecer mas isso acabou gerando árvores maiores para tentarem descrever o dado, o que gerou um overfitting e um processamento lento, devido ao tamanho da maioria dos indivíduos ficarem maior.

References

Foram utilizadas as aulas e slides das aulas como referência.