Regressão Simbólica

Daniel Souza de Campos - 2018054664

0 - Sumário

1 - Introdução

A seleção natural é um mecanismo que pune os indivíduos menos adaptados ao ambiente e dá a chance para os que se adaptam de se reproduzirem e, possivelmente, gerar indivíduos mais adaptados e, portanto, melhores. Pense no exemplo de coelhos que possam ser da cor marrom ou branco e possam ter pernas

fortes ou mais fracas. A combinação dessas duas características, cada uma com dois possíveis valores, gera 4 tipos de coelhos diferentes.

Imagine agora que esses coelhos moram em uma planície de área verde com um matagal de tamanho médio que é cercado por uma região um pouco mais montanhosa e arborizada. O falcão é um predador natural de coelhos. Na planície, com campos de vegetação de porte médio, é mais difícil para o falcão enxergar coelhos marrons do que os brancos. Dessa forma, existe uma tendência para que coelhos brancos sejam pegos, mortos e, portanto, não passem seus genes adiante. Ao longo do tempo, coelhos marrons se tornam a maior parte população de coelhos na planície.

Entretanto, os coelhos brancos de pernas mais fortes conseguiram fugir para as áreas montanhosas e arborizadas, o que os protegeu melhor dos falcões. Coelhos de cor marrom também foram mas em menor número dado que eles se sentem relativamente seguros nos campos. Dentre os coelhos brancos, por ser uma região montanhosa e de difícil locomoção, houve uma tendência para que apenas os que têm pernas mais fortes se reproduzirem. Isso gerou uma população de coelhos brancos que possuem apenas pernas fortes. Dentre os marrons das montanhas, esse padrão se repetiu.

Ao final, existem coelhos marrons de pernas fortes e mais fracas, enquanto que só existem coelhos brancos de pernas fortes. Esse exemplo ilustra como ser melhor adaptado em geral ao ambiente define, ao longo do tempo, a permanência de apenas algumas características na população local.

Algoritmos evolucionários são uma categoria de algoritmos que tentam empregar regras gerais de seleção natural para evoluir soluções candidatas para um problema. Seguindo essa ideia, os indivíduos da população são as soluções candidatas e, de acordo com o problema, elas são sequencialmente filtradas para que apenas as ajustadas ao problema permaneçam.

Em especial, a programação genética é um subconjunto dentro dos algoritmos evolucionários que tende a representar seus indivíduos por meio de uma estrutura chamada árvore. Essa árvore compõe a solução de forma que seus nós representam operações como funções e valores como constantes e variáveis. Assim, dada uma árvore, é possível avaliar esse indivíduo sobre um conjunto de dados ao aplicar as operações contidas na árvore. No mundo real, a adaptabilidade de um indivíduo ao ambiente leva em consideração atratividade para reprodução, posição social, se existir uma sociedade, força, agilidade, camuflagem, sorte entre outros. Para os algoritmos, a adaptabilidade de um indivíduo é sumarizada um uma medida chamada *fitness*. É majoritariamente de acordo com a *fitness* de uma solução que é decidido se ela deve permanecer ou ser descartada. Para as que sobram, também existem operações de reprodução sexuada ou assexuada como cruzamento ou mutação que ajudam a diversificar as soluções candidatas e aumentar a busca no espaço de soluções, realizando, efetivamente, uma busca global.

O objetivo deste trabalho é implementar um algoritmo de programação genética baseado em gramática para resolver problemas de regressão simbólica. A

regressão simbólica é a tarefa de gerar soluções que se adequem a um conjunto de dados em uma predição supervisionada. Isso significa que já temos as soluções para instâncias do problema, mas queremos encontrar uma função que se ajuste à curva real e desconhecida que gerou esses resultados.

Assim, é esperado que seja desenvolvido um mini-framework de programação genética baseado em gramáticas que seja capaz de receber um conjunto de dados como entrada, evoluir soluções e retornar a melhor encontrada.

2 - Ferramentas para o trabalho

O trabalho foi desenvolvido em um notebook Dell com sistema operacional Linux Mint. Ele possui processador Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz e memória RAM de 16GB. Foi utilizada a linguagem de programação Python 3.10.6 no Visual Studio Code. Para armazenamento, foi utilizado o GitHub e o trabalho assim como todos os seus commits estão disponíveis em um repositório.

3 - Gramática

A primeira parte do trabalho foi a tarefa de desenvolver classes necessárias para implementar uma gramática. Essa implementação deveria permitir que a gramática expusesse acesso a funções como recuperar uma expressão aleatória de uma regra, recuperar uma regra aleatória, recuperar uma regra que só gere terminais e recuperar uma regra que só gere não terminais.

Para tal, foram implementadas as classes presentes no arquivo *src/grammar.py* de nome Grammar, Rule, Expansion, Term e subclasses FuncTerm, NonTerminalTerm, TerminalTerm, BinFuncExp, UnFuncExp, FuncExp, VarExp, StrExp e NumericExp. Além disso, o código foi testado no arquivo *src/tests/test_grammar.py*.

Para os experimentos, a seguinte gramática foi definida:

- expr: term binop term | unop term | expr binop expr | unop expr
- term: var | const
- binop: + | - | /* | *
- unop: abs | log10 | cos | sin
- var: X1 | … | Xn
- const: 0 | … | 10

A operação /* para a regra binop se refere a uma divisão segura, ou seja, se o dividendo for próximo de 0, ele retorna o numerador.

A operação log10 para a regra unop se refere a um cálculo seguro, ou seja, se o valor de entrada for menor ou igual a 0, é retornado 0.

As expressões da regra var se referem às variáveis que podem ser acessadas de acordo com a instância do conjunto de dados. Espera-se que cada instância possua pelo menos uma variável independente. A quantidade de expressões de var será definida uma vez que o conjunto de dados seja recebido.

A configuração dessa instância de gramática pode ser vista no arquivo *src/main.py* linha 123 no método *configure_grammar*.

Foi considerado que as operações disponíveis na gramática são suficientes para propor soluções que podem resolver uma boa gama de problemas.

4 - Programação Genética

4.1 - Gerando um indivíduo

Uma vez com a gramática implementada, é necessário conseguir gerar indivíduos ou soluções a partir dela. No geral, 3 métodos são famosos para gerar árvores de indivíduos a partir de uma gramática. São eles: Grow, Full e Ramped half-and-half. Para esse trabalho, apenas o método Grow foi implementado na classe GrowTreeGenerator em *src/grammar.py* linha 450.

A ideia do método Grow é que, a partir de uma gramática, uma regra base, uma altura máxima e a profundidade atual, o algoritmo vá expandindo regras aleatoriamente de forma que o indivíduo gerado nunca ultrapasse a altura máxima.

O indivíduo é simplesmente uma árvore que possui um nó raiz. O indivíduo foi implementado na classe Individual e os seus nós são todos da classe pai Node, ambos implementados no arquivo *src/ind_generator.py*. Subclasses de Node foram implementados em *src/grammar.py* nas classes OPNode, BinOPNode, UnOPNode, VarNode e ConstNode a partir da linha 313.

As principais funcionalidades de um Individual são: se avaliar em uma instância dos dados, recuperar um nó aleatório e o seu nó pai, encontrar um nó de um certo tipo e seu pai. Além disso, o indivíduo possui uma variável cache que indica a sua fitness ao ser avaliado sobre o dataset inteiro.

4.2 - Selecionando um indivíduo

Uma vez podendo gerar indivíduos, fica fácil gerar uma população. Durante a execução do algoritmo de programação genética, será necessário selecionar indivíduos da população na etapa de seleção. Para tal, foram implementados 3 métodos de seleção de indivíduos. São eles: Seleção por Roleta, Seleção por Torneio e Seleção Lexicase.

4.2.1 - Seleção por Roleta

A seleção por roleta está implementada na classe RoulleteSelection na linha 239 no arquivo *src/genetic_prog.py*. O seu funcionamento é simples: Avalie todos os indivíduos da população sobre o conjunto de dados de entrada inteiro. Se for considerado que uma avaliação é melhor se ela for menor, é realizado uma conversão da avaliação original para refletir essa regra. Depois, realizamos n

amostras com reposição sendo que o peso de cada indivíduo é proporcional à sua avaliação. Retornamos as cópias desses indivíduos.

4.2.2 - Seleção por Torneio

A seleção por torneio está implementada na classe TournamentSelection na linha 266 no arquivo *src/genetic_prog.py*. O seu funcionamento também é simples: Devem ser realizadas n seleções. Para cada seleção, é feita uma amostragem com reposição de k indivíduos da população. Esses k indivíduos são avaliados em todo o conjunto de dados de entrada. Apenas o que possui a melhor fitness, seguindo a regra já mencionada de ser a menor ou a maior, será selecionado. Ao final, retorna-se as cópias dos indivíduos selecionados.

4.2.3 - Seleção Lexicase

A seleção Lexicase está implementada na classe LexicaseSelection na linha 299 do arquivo *src/genetic_prog.py*. A seleção Lexicase funciona da seguinte forma: São necessárias n seleções. Para cada seleção, o conjunto de dados de entrada é embaralhado. Sequencialmente, avalia-se os indivíduos atuais sobre essa única instância de dados. Depois, de acordo com o MAD, um threshold de fitness sobre a melhor fitness encontrada para a instância de dados atual, são escolhidos indivíduos. Esses indivíduos serão separados para a próxima iteração onde serão avaliados na próxima instância de dados. Isso acontece até que sobre apenas um indivíduo ou até que todas as instâncias de dados sejam consideradas. Nesse último caso, um indivíduo aleatório é selecionado dentre os que sobraram.

4.3 - Operações

Após a escolha dos indivíduos, deve-se aplicar os operadores de mutação ou cruzamento. A implementação foi feita de forma que seja possível escolher quantas mutações e cruzamentos devem ocorrer. A quantidade deve levar em consideração a quantidade de indivíduos na população.

4.3.1 - Cruzamento

A classe que realiza cruzamento entre indivíduos se chama CrossoverOP e está implementada a partir da linha 397 no arquivo *src/genetic_prog.py*. O cruzamento ocorre da seguinte forma: São fornecidos dois indivíduos iniciais, não necessariamente diferentes, e uma altura máxima que os indivíduos resultantes devem respeitar. É escolhido um nó aleatório do primeiro indivíduo e é realizada uma busca no segundo indivíduo por um nó equivalente e apto a ser trocado pela subárvore do primeiro nó de forma a respeitar a altura máxima. Essa permissão é

dada pela lógica de que, se a profundidade do nó atual adicionado da altura do outro nó for menor ou igual a altura máxima, então o nó atual pode ser substituído na sua árvore pelo segundo. Essa condição é avaliada tanto para o primeiro nó substituir o segundo e vice-versa. Se encontrado um nó apto no segundo indivíduo, o cruzamento é uma simples substituição de filhos dos nós pais dos nós trocados.

4.3.2 - Mutação

A classe que realiza a mutação de um indivíduo se chama MutationOP e está implementada a partir da linha 371 no arquivo *src/genetic_prog.py*. O funcionamento dessa operação é simples: A partir de uma gramática, um gerador de indivíduos (no caso o Grow), um indivíduo e uma altura máxima, escolhemos um nó aleatório do indivíduo. Geramos uma nova subárvore para substituir o nó escolhido e a sua subárvore no indivíduo. Substituímos, de fato, o nó pela nova subárvore e retornamos o novo indivíduo.

5 - Experimentos e resultados

5.1 - Executando o algoritmo

Para executar o algoritmo, primeiramente, crie um ambiente virtual python e instale as dependências contidas no arquivo *requirements.txt*. Depois, deve-se executar o código no arquivo *src/main.py*. O algoritmo precisa receber muitos parâmetros e, portanto, para facilitar a vida do usuário, foi disponibilizado um script shell de nome *src/run.sh* em que é possível definir os parâmetros necessários à execução do algoritmo. Parâmetros:

- --train_data_path: O caminho para o arquivo contendo os dados de treino
- --test_data_path: O caminho para o arquivo contendo os dados de teste
- --target_col: A coluna alvo a ser predita pelos indivíduos. Por padrão, é o ‘Y’. Detalhe, os arquivos de entrada não precisam ter cabeçalho. Fica convencionado que a última coluna de dados se refere à coluna ‘Y’.
- --num_inds: Número de indivíduos na população
- --num_gens: Número de gerações
- --selection_type: Tipo de seleção. Pode ser um dos seguintes valores: ‘Tournament’, ‘Roullete’ e ‘Lexicase’.
- --max_height: Altura máxima dos indivíduos
- --selection_k: O tamanho da seleção para os modos de seleção que precisam dele
- --better_fitness: Indicativo de como considerar que uma fitness é boa. Pode ser um dos seguintes valores: ‘lower’ ou ‘greater’.

- --elitism: Essa é uma flag indicando o uso de elitismo.
- --n_mut: Número de mutações a serem realizadas a cada geração
- --n_cross: Número de cruzamentos a serem realizados a cada geração
- --p_cross: Probabilidade de cruzamento
- --p_mut: Probabilidade de mutação
- --num_runs: Número de execuções a se fazer
- --base_name: O nome base do conjunto de dados.

Todas essas opções tem variáveis correspondentes no script shell fornecido a não ser pela flag –elitism que deve ser manualmente inserida ou retirada do comando python executado.

A quantidade de mutações e cruzamentos deve respeitar a seguinte conta:

- Se não elitismo: num_inds == n_mut + 2*n_cross
- Se com elitismo: num_inds -1 == n_mut + 2*n_cross

Dessa forma, caso essas restrições não sejam satisfeitas, o algoritmo irá levantar uma exceção apropriada.

Além desses parâmetros, são necessárias duas funções de fitness dependendo do método de seleção. Uma função de fitness diz respeito à avaliação do indivíduo sobre todo o dataset. A outra função de fitness diz respeito à avaliação do indivíduo sobre uma única instância de dados do dataset.

A primeira é utilizada nas seleções por torneio e roleta e também para computar estatísticas sobre os indivíduos, enquanto a segunda é usada somente durante a seleção lexicase.

A função de fitness usada sobre todo o dataset foi implementada de forma *hardcoded* na linha 155 do arquivo *src/main.py* e ela equivale ao RMSE. A função de fitness sobre uma única instância foi implementada de forma hardcoded na linha 220 em *src/main.py* como uma simples função lambda que retorna o erro absoluto da predição sobre o esperado.

Por padrão, os resultados serão gerados dentro da pasta ‘resultados’. Com base no valor informado em –base_name, uma pasta será criada dentro de ‘resultados’. Após as –num_runs execuções, será criada uma pasta no formato: <ano><mes><dia>-<hora><minuto><segundo> com os resultados do experimento.

Cada experimento gera –num_runs arquivos *.csv* contendo estatísticas computadas para cada iteração e um outro arquivo de nome *fitness_stats.csv* que possui os dados de fitness no treino e teste, tempo de execução e a seed aleatória usada para gerar esses resultados a cada iteração.

5.2 - Informações gerais

Os experimentos realizados variaram o número de indivíduos entre 50, 100 e 500. Para seguir a regra imposta sobre o número de mutações e cruzamentos, foram adotados os seguintes parâmetros:

- Para 50 indivíduos: n_cross=19 e n_mut=11 com elitismo e n_mut=12 sem.
- Para 100 indivíduos: n_cross=35 e n_mut=29 com elitismo e n_mut=30 sem.

- Para 500 indivíduos: n_cross=200 e n_mut=99 com elitismo e n_mut=100 sem.

Além disso, as probabilidades de cruzamento (p_cross) e de mutação (p_mut) foram, respectivamente, de 0.9 e 0.05 a não ser que informado outros valores.

Todos os experimentos foram executados 30 vezes e com elitismo a não ser que informado o contrário.

5.2 - Experimentos sobre synth1

Essa seção apresenta os experimentos realizados sobre o dataset synth1 e os seus resultados seguindo o guia para experimentos fornecido.

5.2.1 - Variando população e gerações

O primeiro experimento é o de variar a quantidade de indivíduos e de gerações mantendo as probabilidades de cruzamento e mutação e o método de seleção, no caso, roleta. A Figura 1 mostra a melhor fitness de treino e teste usando 50 indivíduos e variando o número de gerações.

Figura 1

Os resultados médios para a melhor fitness são mostrados na Tabela 1:

Modo	50 gerações	100 gerações	500 gerações
Treino	15.375	14.467	11.097
Teste	22.170	21.578	16.597

Tabela 1: Média aparada da fitness do melhor indivíduo (50 indivíduos)

Com esses resultados, é possível ver que usar mais gerações melhora o resultado. No geral, esse foi o padrão observado.

As figuras 2 e 3 mostram as mesmas análises para as populações de 100 e 500 indivíduos e as tabelas 2 e 3 mostram as médias das melhores fitness.

Figura 2

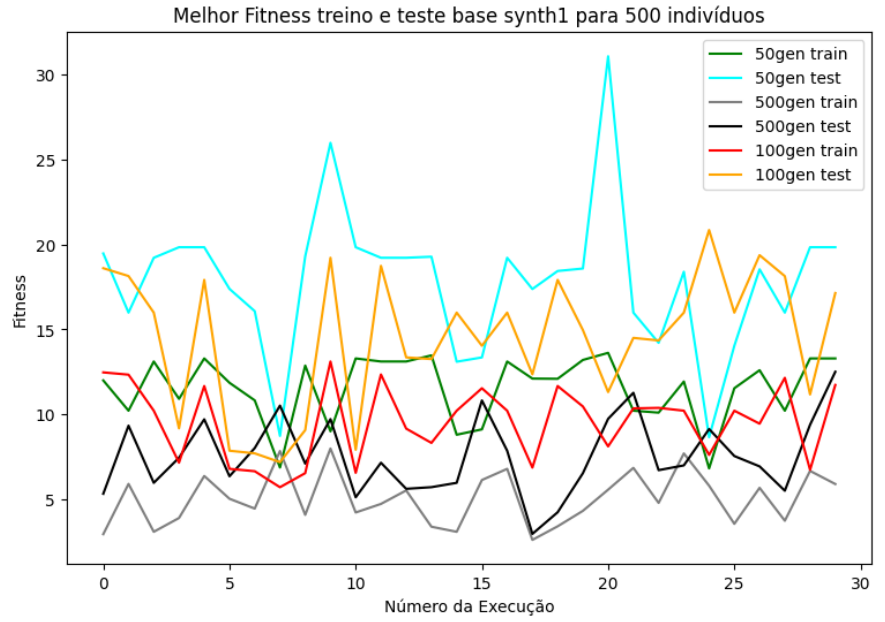

Figura 3

Modo	50 gerações	100 gerações	500 gerações
Treino	15.230	12.285	5.800
Teste	21.778	19.165	8.600

Tabela 2: Média aparada (5%) da fitness do melhor indivíduo (100 indivíduos)

Modo	50 gerações	100 gerações	500 gerações
Treino	11.628	9.579	5.060
Teste	17.719	14.503	7.567

Tabela 3: Média aparada (5%) da fitness do melhor indivíduo (100 indivíduos)

Os resultados também melhoram quanto mais gerações permitimos ao algoritmo. De acordo com a média aparada, os resultados obtidos com 500 gerações são levemente melhores do que para o mesmo número de gerações e 100 indivíduos. A Figura 4 mostra a comparação da fitness do melhor indivíduo para 500 gerações.

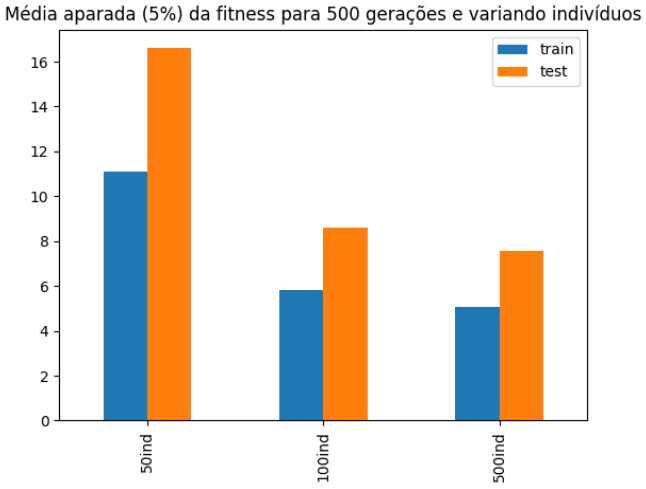

Figura 4

É possível perceber na Figura 4 que o ganho médio associado a utilizar 500 indivíduos quando comparado a 100 indivíduos é baixo. A Figura 5 mostra que, dado o grande aumento do tempo de execução para 500 indivíduos, talvez não valha a pena utilizar de 500 indivíduos para um ganho tão pouco em performance.

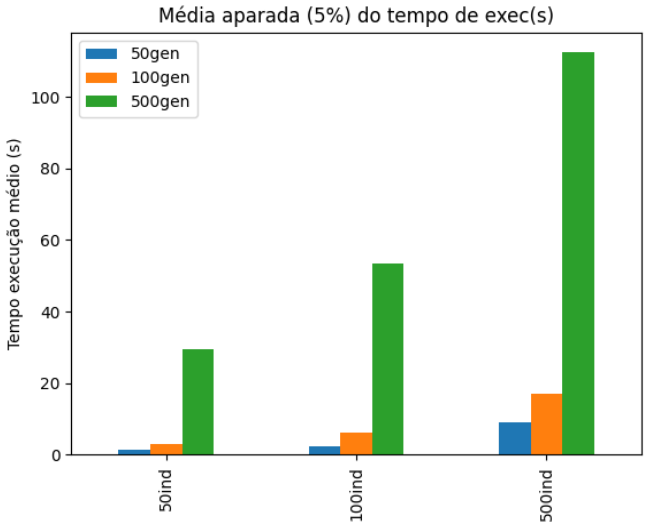

Figura 5

A Figura 6 compara a média da melhor fitness a cada geração ao variar a quantidade de indivíduos ao longo de 500 gerações. É possível perceber que a utilização de 500 indivíduos bate a performance média das outras configurações. Além disso, de acordo com a faixa verde representando os valores máximos e mínimos, é possível ver que as fitness possuem um desvio padrão menor do que as outras configurações.

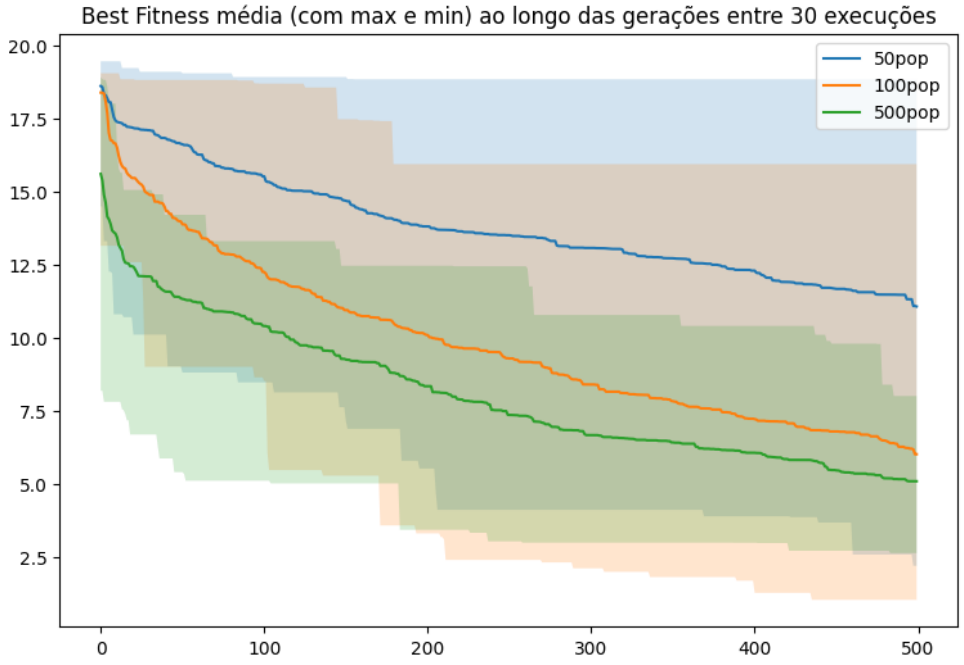

Figura 6

Apesar disso, a utilização de 100 indivíduos também segue o mesmo movimento ao usar 500. Dessa forma, já que utilizar de 500 indivíduos chega a dobrar o tempo de execução (Figura 5), resolvi seguir com os outros experimentos utilizando 100 indivíduos e 500 gerações.

5.2.2 - Variando método de seleção

De acordo com o experimento anterior, utilizar 100 indivíduos e 500 gerações foi considerado melhor. Acontece que o experimento anterior foi testado somente com a seleção por roleta. Nessa subseção, realizaram-se experimentos com seleção de torneio e Lexicase.

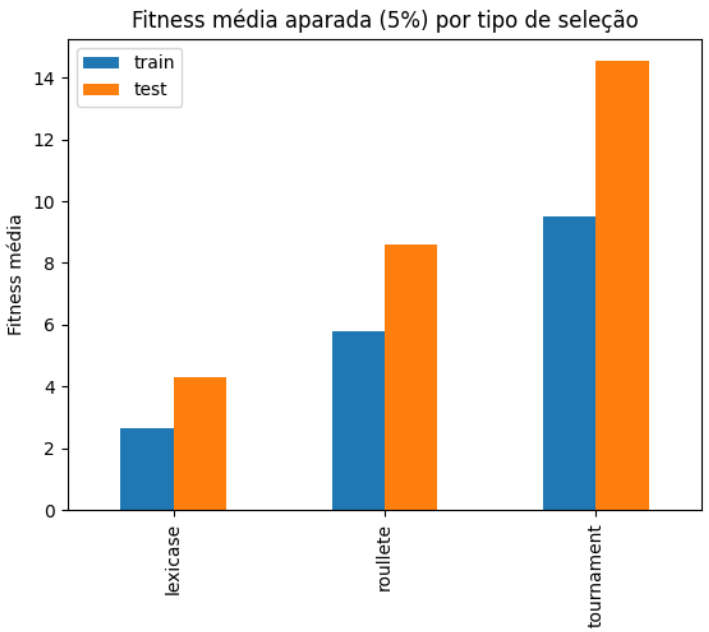

Figura 7

A Figura 7 mostra que usar a seleção Lexicase é claramente melhor do que as outras duas, sendo que a fitness média de teste é menor do que a fitness média de treino da roleta e menos da metade da fitness média de treino do torneio.

Tipo Seleção	Tempo médio
Lexicase	397.401305
Roleta	53.540892
Torneio	62.608704

Tabela 4: Tempo médio de execução por método de seleção

A Tabela 4 mostra que, apesar de produzir resultados melhores, o método Lexicase foi cerca de 8 vezes mais custoso do que o por Roleta e 5 vezes mais custoso do que o por Torneio.

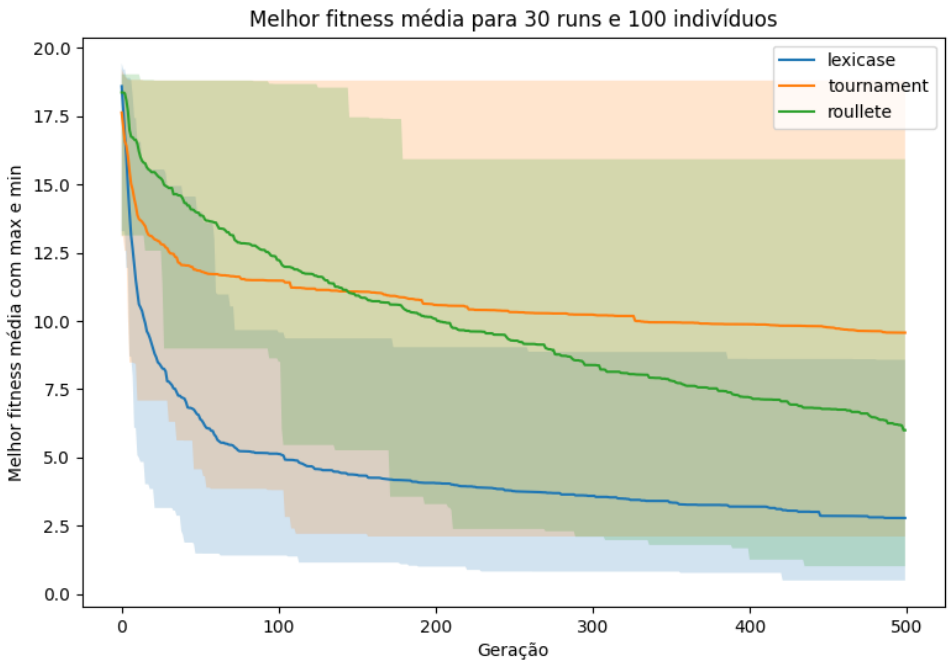

Figura 8: Média da melhor fitness por tipo de seleção

A Figura 8 mostra que a seleção por Lexicase possui uma capacidade muito boa de se aproximar do melhor indivíduo em cerca de 100 gerações, mas ainda assim ser capaz de melhorá-lo e não ficar em um platô. Assim, na média, com cerca de 200 iterações ela já é capaz de aproximar suficientemente o melhor indivíduo.

A seleção por roleta, apesar de ter uma pressão seletiva maior, parece ser diversa o suficiente para que tenha um movimento de melhoria constante em média. No pior caso, com cerca de 180 gerações, não conseguimos melhorar a sua solução.

Já a seleção por torneio, no seu pior caso, não conseguiu melhorar em quase nada a solução inicial, mesmo com torneios de tamanho k=2. Em média, ela possui um movimento semelhante à seleção por lexicase mas com força menor de melhoria da solução.

A Figura 9 mostra a melhor execução realizada por tipo de seleção. O gráfico mostra que, para as melhores execuções, o resultado usando a seleção por roleta chega muito próximo do que uma seleção por Lexicase, apesar de que a diferença entre a fitness média e a melhor é constantemente menor para o caso Lexicase quando comparado com a roleta. Isso indica que, talvez, se escolhermos a seleção por roleta e a permitirmos mais iterações, podemos tentar chegar à performance da seleção Lexicase e em menos tempo de execução.

Já que usar Lexicase leva cerca de 8 vezes mais tempo do que a Roleta, poderíamos usar de até 4000 gerações e ainda levaríamos o mesmo tempo que o Lexicase. Entretanto, essa hipótese não será testada.

Dessa forma, como a seleção por Roleta, executando cerca de 30 vezes, consegue se aproximar em muito da melhor solução do Lexicase e ser 8 vezes mais barata de computar, escolhi manter essa como a melhor seleção.

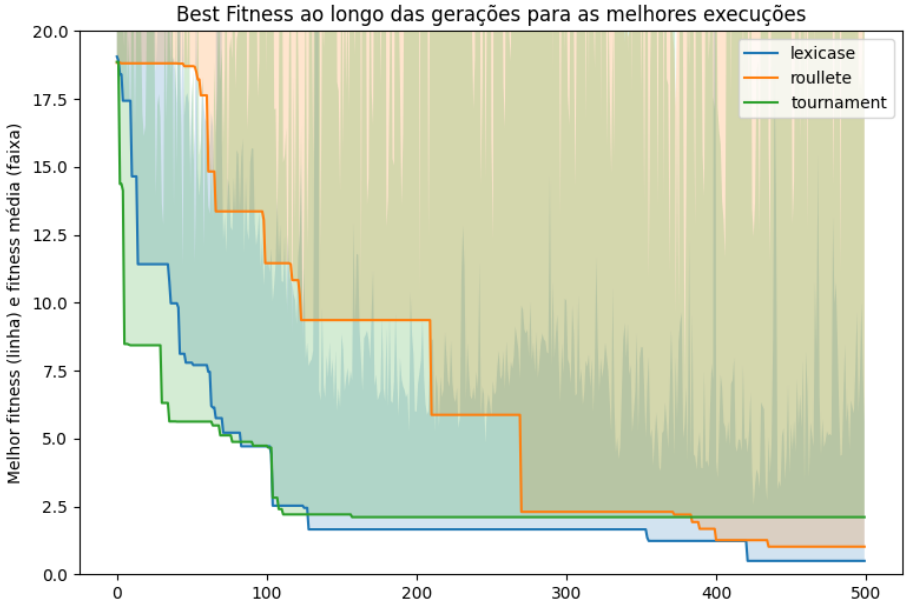

Figura 9: Fitness do melhor indivíduo encontrado por método de seleção

5.2.3 - Variando probabilidades de cruzamento e mutação

Os dois últimos experimentos foram realizados utilizando probabilidade de cruzamento igual a 0.9 e de mutação igual a 0.05. Vou chamar essa configuração de base. Agora vamos manter 100 indivíduos, 500 gerações, seleção por roleta mas variar a probabilidade de cruzamento para 0.6 e de mutação para 0.3. Vou chamar essa configuração de nova.

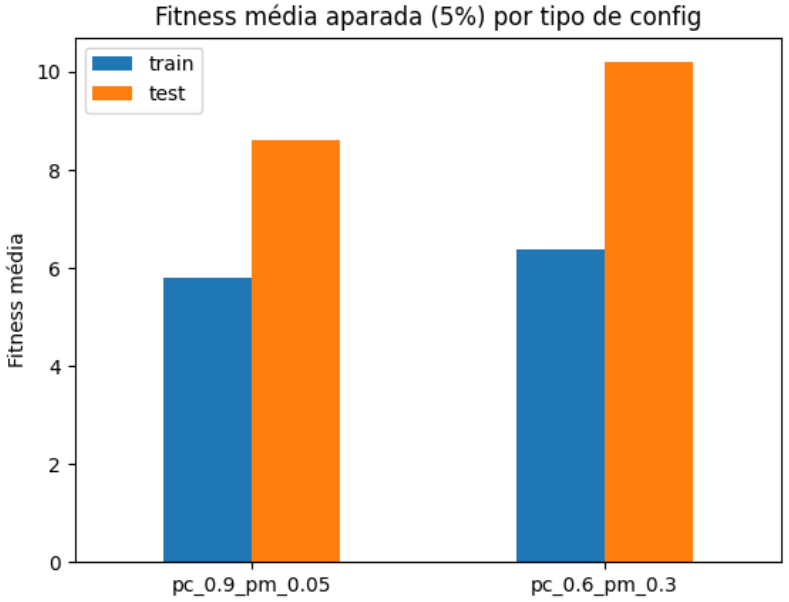

Figura 10: FItness média variando probabilidades de cruzamento e de mutação

A Figura 10 compara a fitness média obtida entre utilizar a configuração base e a nova. A configuração base é levemente melhor do que a nova.

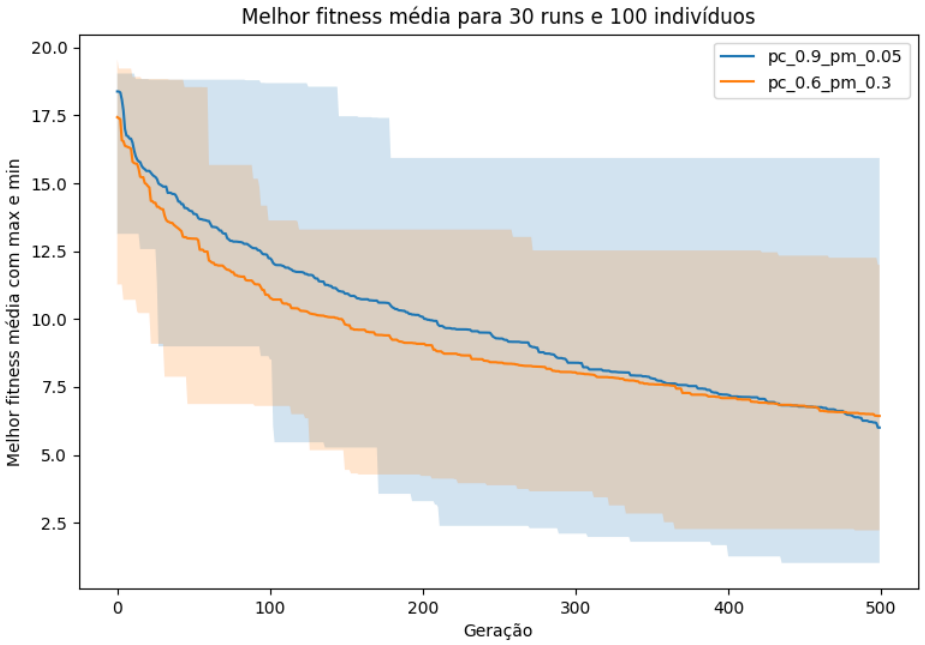

Figura 11: Média da melhor fitness para cada configuração de probabilidades

A Figura 11 mostra que, na média, a nova configuração tem uma força maior para melhorar as soluções. Isso pode indicar que existe um ganho entre aumentar a probabilidade de mutação e diminuir um pouco a de crossover. Entretanto, à medida que as iterações passam, a taxa de melhora vai diminuindo e, já ao redor de 480 iterações, a configuração base possui um indivíduo melhor, em média, do que a configuração nova.

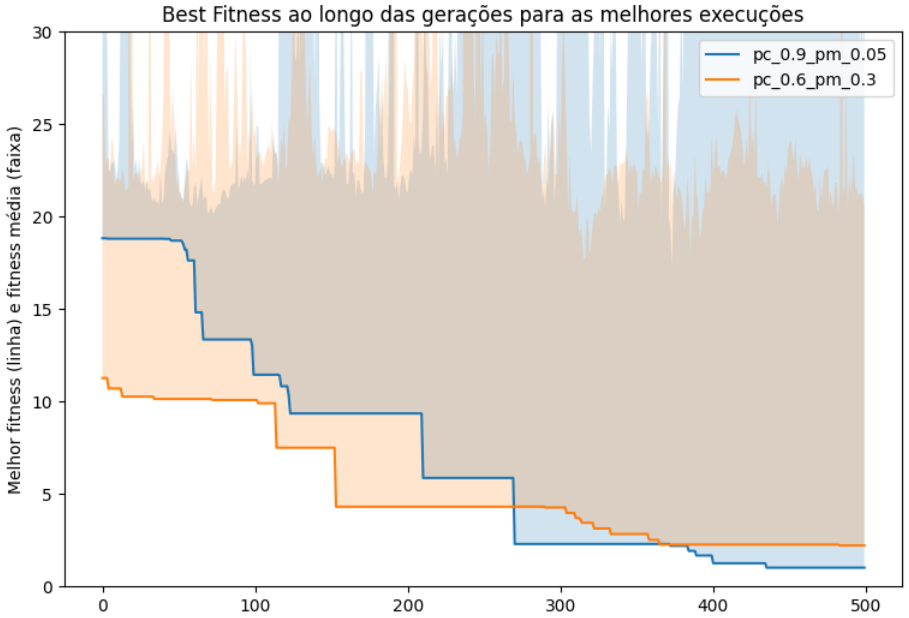

Figura 12: Comparando a melhor execução para ambas as execuções

A Figura 12 confirma o movimento em que a nova configuração tem indivíduos melhores nas primeiras gerações mas, já nas últimas gerações, ele produz indivíduos piores do que a configuração base. Dessa forma, vou continuar usando a configuração base.

5.2.4 - Elitismo ou não

Todos os experimentos anteriores foram utilizando elitismo. Isso é perceptível já que a melhor fitness dentro de uma mesma execução nunca é pior do que a melhor fitness da geração passada. Vou ver como não usar elitismo afeta o resultado mantendo tudo constante.

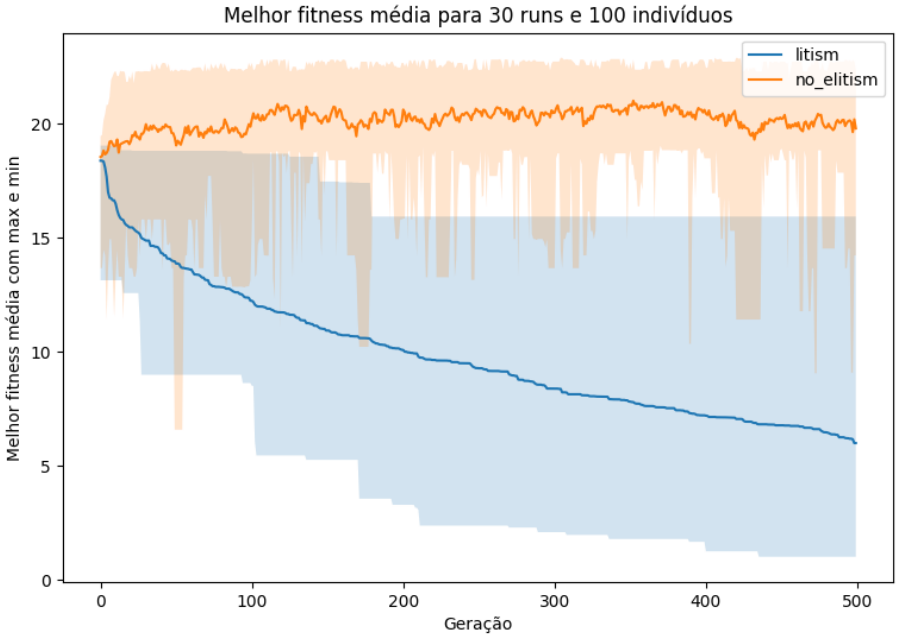

Figura 13: Média das melhores fitness usando ou não elitismo

Existe uma diferença óbvia entre usar ou não elitismo. Dessa forma, esse experimento serviu apenas para confirmar a sua importância.

5.3 - Experimento sobre synth2

Seguindo o guia proposto de experimentação, vou aplicar os melhores parâmetros encontrados na base sintética 1 sobre a base sintética 2 e analisar os resultados.

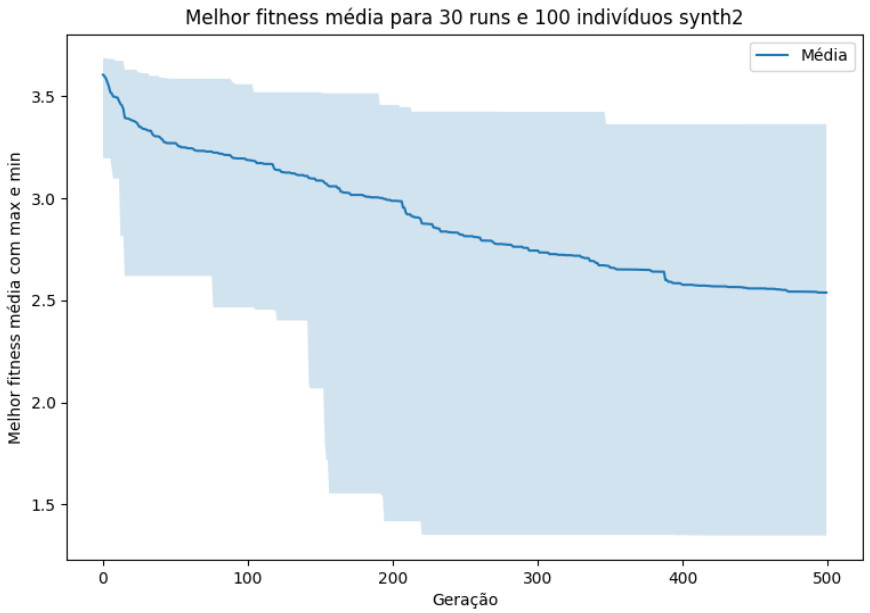

Figura 14: Média da melhor fitness para o synth2

A Figura 14 mostra o comportamento médio do algoritmo ao definir os melhores parâmetros sobre a base synth2. É possível perceber que consegui bons resultados médios. Além disso, o melhor resultado de todos obteve uma fitness muito boa.

Analisando os resultados, percebi que duas seeds aleatórias geraram o mesmo melhor resultado. A Tabela 5 mostra essa situação.

Fitness de treino	Fitness de teste	Tempo execução treino (s)	Random Seed
1.351	1.358	73.085	173
1.351	1.358	107.472	942

Tabela 5: Duas seeds diferentes geraram o mesmo melhor resultado

Assim, resolvi comparar a execução para essas duas seeds. A Figura 15 mostra essa comparação. De acordo com o gráfico, a execução com a seed 173 consegue atingir a melhor solução em cerca de 100 iterações a menos do que a seed 942.

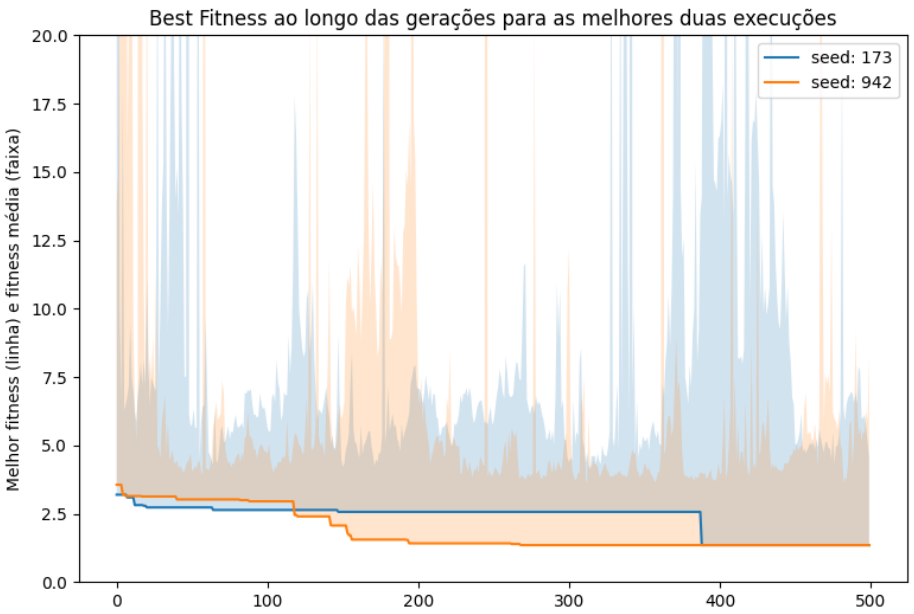

Figura 15: Comparando execuções de duas seeds diferentes

5.4 - Experimentos sobre concrete

Os experimentos realizados sobre a base de dados concrete são exatamente os mesmos que aqueles realizados sobre o synth1. Entretanto, os resultados são diferentes.

5.4.1 - Variando população e gerações

O primeiro experimento é o de variar a quantidade de indivíduos e de gerações mantendo as probabilidades de cruzamento e mutação e o método de seleção, no caso, roleta.

A Figura 16 mostra os resultados das melhores fitness de treino e execução para 50 indivíduos variando o número de gerações. Pode-se perceber que a performance sobre os dados de treino é muito parecida com aquelas dos dados de teste. Esse é um comportamento mais organizado quando comparado com aquele da Figura 1. Isso pode indicar que existe uma maior qualidade na separação dos dados de treino e teste, de forma que ambos representam, de forma equivalente, o problema.

A Tabela 6 mostra a média aparada da fitness do melhor indivíduo para 50 indivíduos variando a quantidade de gerações.

Figura 16

Modo	50 gerações	100 gerações	500 gerações
Treino	17.163	16.442	15.719
Teste	16.749	15.953	15.318

Tabela 6

As Figuras 17 e 18 e as Tabelas 7 e 8 mostram os mesmos resultados para 100 e 500 indivíduos respectivamente. Entretanto, não foi realizada a execução de 500 indivíduos e 500 gerações devido ao seu custo computacional.

Figura 17

Figura 18

Modo	50 gerações	100 gerações	500 gerações
Treino	17.089	16.303	14.602
Teste	16.635	15.817	14.324

Tabela 7: Média da melhor fitness para 100 indivíduos na base concrete

Modo	50 gerações	100 gerações
Treino	15.345	14.553
Teste	14.874	14.269

Tabela 8: Média da melhor fitness para 500 indivíduos na base concrete

A Figura 20 mostra a média aparada das melhores fitness para cada configuração e 100 gerações. Pode-se perceber que a utilização de 500 indivíduos traz um pequeno ganho sobre os outros.

A Figura 21 mostra a média do tempo de execução para todas as configurações. Com ela, é possível ver que existe um enorme aumento no custo de computação quando se usa 500 indivíduos.

A Figura 22 mostra o comportamento médio, mínimo e máximo ao variar o número de indivíduos e manter 100 gerações. Nesse gráfico, é possível ver que a utilização de 500 indivíduos realmente é melhor. Outra observação é a de que a melhor fitness encontrada ao usar de 100 indivíduos é melhor do que a melhor fitness média ao usar de 500 indivíduos. Isso mostra que empregar 100 indivíduos não é tão pior assim.

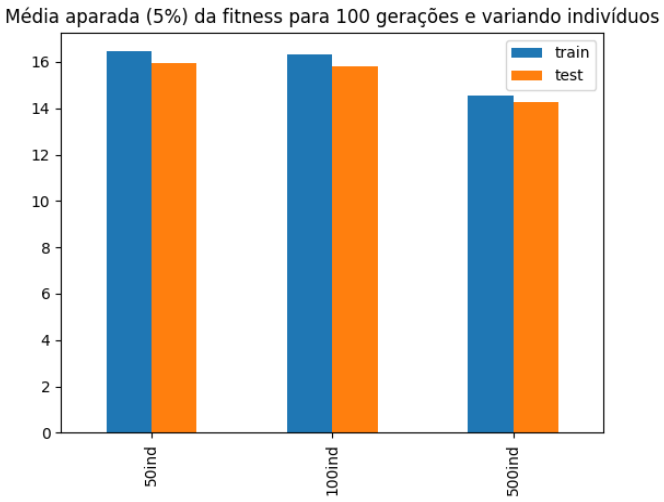

Figura 20

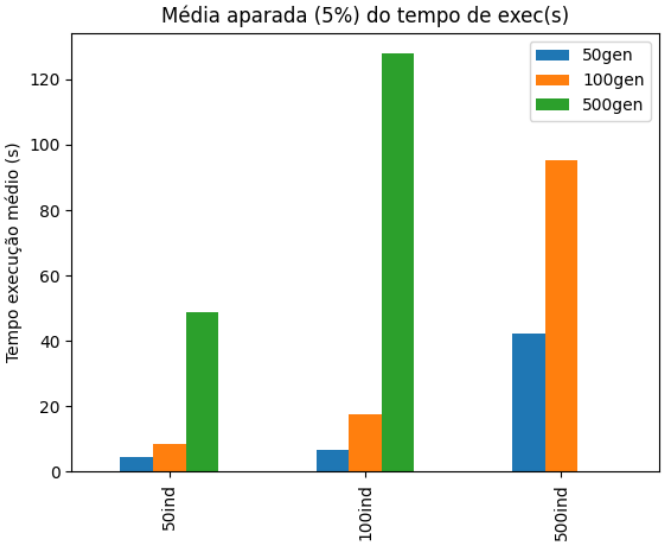

Figura 21: Média dos tempos de execução para a base concrete

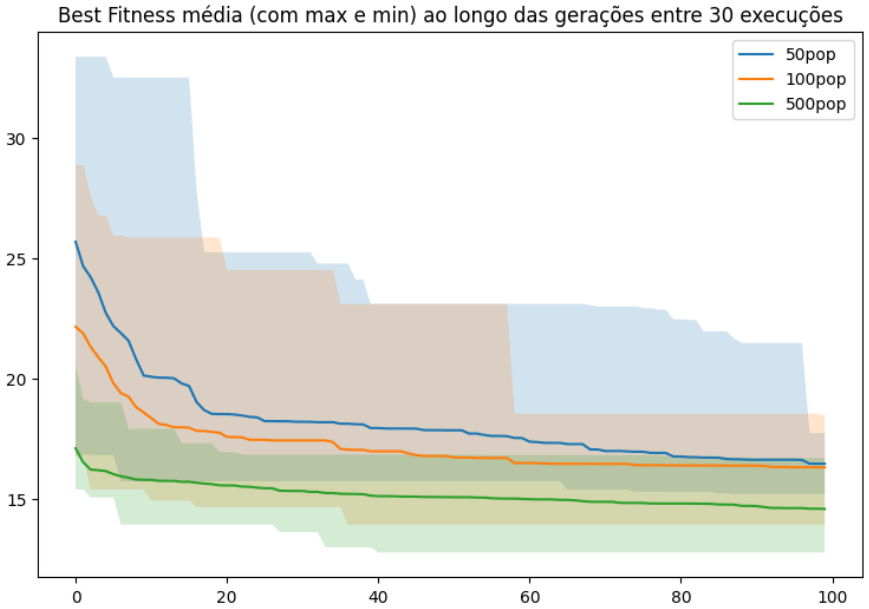

Figura 22: Média da melhor fitness ao longo das gerações para a base concrete

Dessa forma, dado o custo computacional de executar para 500 indivíduos, escolhi permanecer com 100 indivíduos.

5.4.2 - Variando método de seleção

Uma vez definido usar 100 indivíduos, vou variar os métodos de seleção. Além disso, para compensar o não uso de 500 gerações, vou permitir 300 gerações. O método de seleção Lexicase já havia sido mostrado caro apesar de mais efetivo que os outros. Dessa forma, não realizei esse experimento com o método de seleção Lexicase.

A Figura 23 mostra a média aparada da melhor fitness para cada seleção. Por ela, vemos que a seleção por Torneio teve uma performance levemente melhor do que a por Roleta. Além disso, o tempo médio de execução usando do método Torneio, 105.34 segundos, foi consideravelmente menor do que o método Roleta de 127.75 segundos.

A Figura 24 mostra média das melhores fitness entre todas as execuções para cada método de seleção. De acordo com esse gráfico, apesar de a seleção por Torneio ter uma fitness média final pior do que a seleção por Roleta, o seu melhor caso foi consideravelmente melhor do que o do outro método. Além disso, o gráfico mostra que ambos tipos de seleção possuem um movimento semelhante. Com cerca de 150 gerações eles já atingem uma solução muito próxima da melhor encontrada.

A Figura 25 compara melhor as melhores execuções dos dois métodos. Nela, é possível ver que, ao usar o Torneio, o melhor indivíduo é encontrado cerca de 50 gerações antes do que a do Roleta. Além disso, o uso de Torneio foi sempre melhor do que as soluções apresentadas pela Roleta.

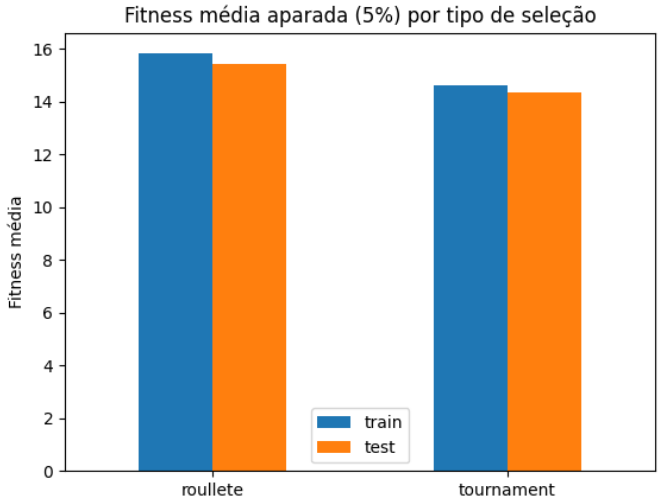

Figura 23: Média das melhores fitness para os métodos de seleção

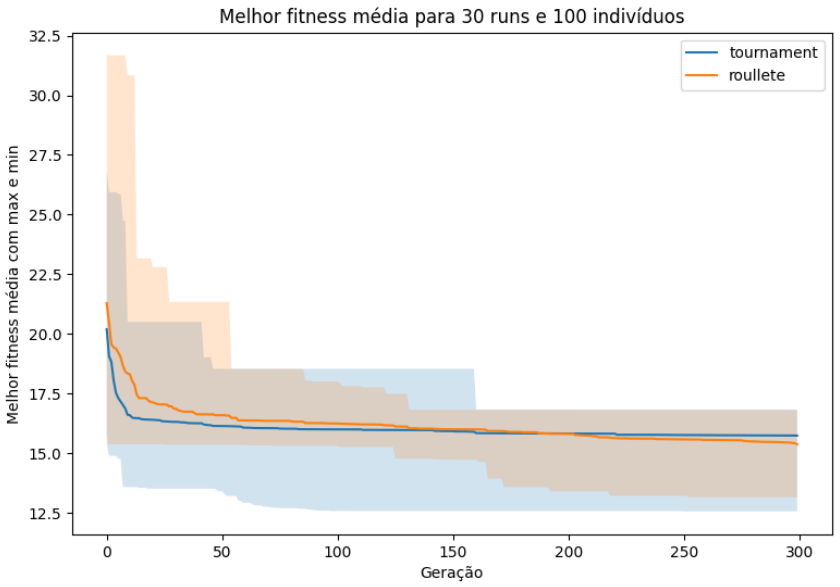

Figura 24

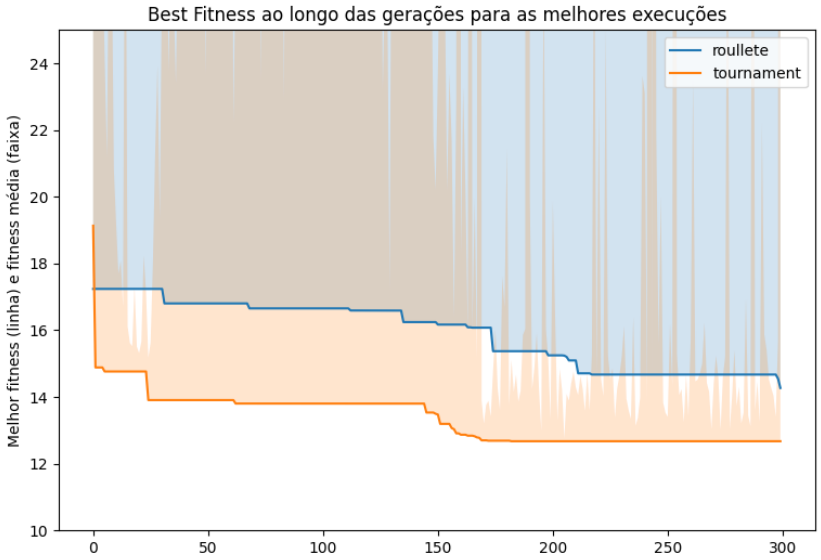

Figura 25

De acordo com todos os benefícios de usar o Torneio sobre a Roleta, decidi passar a usar esse método de seleção.

5.4.3 - Variando probabilidade de cruzamento e mutação

Da mesma forma que na seção 5.2.3, vou chamar a configuração de p_cross = 0.9 e p_mut = 0.05 de base e a configuração p_cross = 0.6 e p_mut = 0.05 de nova. Mantive todo o resto igual, ou seja, 300 gerações, 100 indivíduos, elitismo e seleção por torneio.

A Figura 26 compara a fitness média aparada ao variar as duas configurações. A nova configuração foi levemente melhor do que a base.

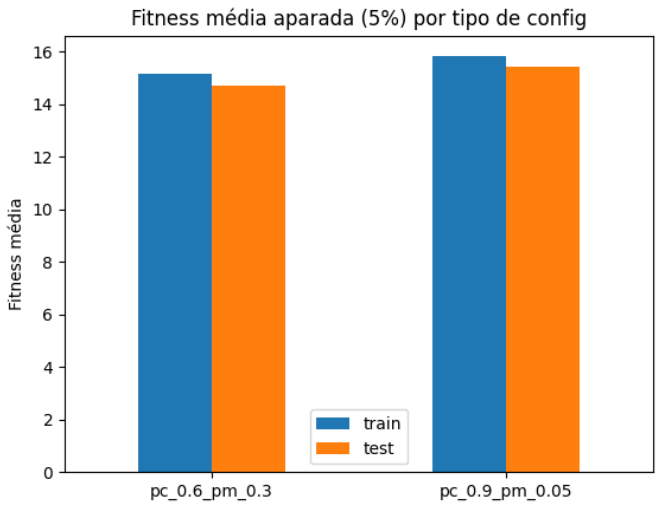

Figura 26

A Figura 27 compara a média das melhores fitness para cada geração e configuração. Pode-se perceber que a média das fitness para a configuração nova é levemente menor do que a configuração base. Também é possível ver que o melhor caso para a nova configuração é bem melhor do que o melhor caso para a configuração base.

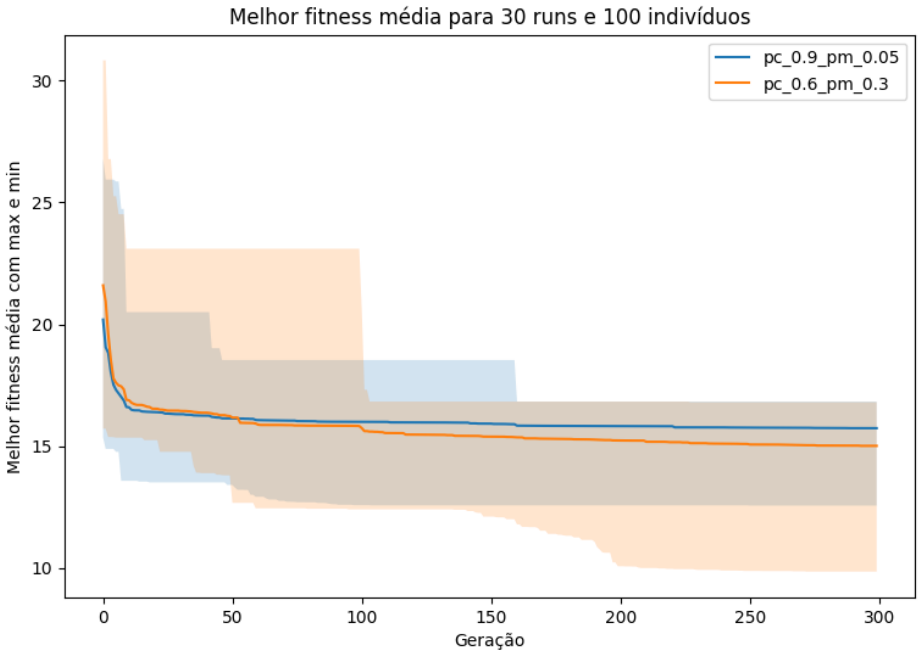

Figura 27

Dadas essas características, decidi adotar a configuração nova como a melhor.

5.4.4 - Elitismo ou não

O último experimento é o de avaliar o uso de elitismo ou não na solução final. A Figura 28 mostra a média aparada da melhor fitness por configuração. É possível ver que, desta vez, o não uso de elitismo não piorou consideravelmente a média das fitness, o que é inesperado. Isso pode mostrar que, sem usar mecanismos de

diversidade, o não uso de elitismo se baseia em uma boa seed aleatória que permita gerar indivíduos melhores ao longo do tempo.

A Figura 29 confirma que o movimento do algoritmo ao não usar elitismo é muito parecido com aquele que usa. Esse resultado é muito diferente daquele obtido na seção 5.2.4.

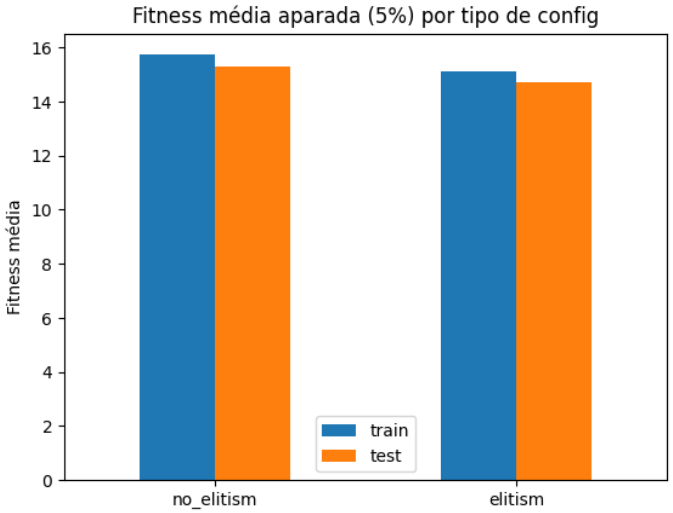

Figura 28

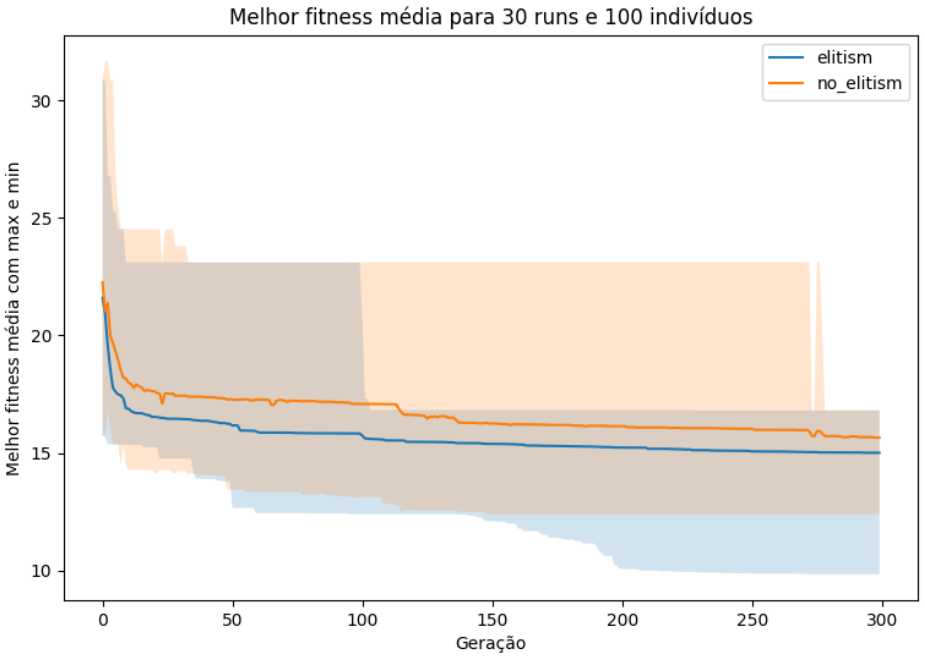

Figura 29

6 - Conclusão

O objetivo do trabalho era o de desenvolver um mini-framework que permitisse executar algoritmos de programação genética e realizar vários experimentos com uma boa parametrização. Acredito que esse resultado tenha sido alcançado dado todos os experimentos executados nas seções anteriores.

Com esse trabalho pude aplicar técnicas utilizadas em algoritmos evolucionários vistos em sala de aula para resolver o problema de regressão simbólica. O guia de experimentação foi crucial para o desenvolvimento do trabalho, já que ele direcionou os meus experimentos para que eles fossem o mais consistente possível.

Entre as principais questões analisadas na experimentação, a variação do número de gerações e da população se mostraram muito importantes e, talvez, as principais. Os experimentos realizados nas seções 5.2.1 e 5.4.1 deixaram clara a diferença de se permitir usar mais indivíduos e gerações além de expor o aumento do custo computacional.

A variação do método de seleção também se mostrou muito importante e tendeu a seguir a mesma lógica ao variar a população e número de gerações: melhores resultados vem de métodos com maior custo computacional. Entretanto, os melhores resultados podem ser aproximados com métodos mais baratos ao aumentar o número de gerações disponíveis e, ainda assim, provavelmente ter um custo computacional menor.

No geral, a variação de probabilidades de cruzamento e mutação não se mostrou muito efetiva em alterar o resultado. Já o uso de elitismo mantém o algoritmo na direção certa de melhora de resultados sem se basear na sorte de conseguir uma boa seed aleatória ao não usar o elitismo.