Trabalho Prático 1 - Programação Genética

Renato Sérgio Lopes Júnior - 2020667570

1 Introdução

Neste trabalho foi desenvolvido um programa que encontra uma função $d(e_i, e_j)$ que mede a distância entre pares de exemplos e_i e e_j de uma base de dados qualquer. Para encontrar essa função d foi utilizado o arcabouço da Programação Genética (GP) e a técnica de regressão simbólica. Uma vez encontrada essa função $d(e_i, e_j)$, ela é utilizada em um algoritmo de agrupamento, o que possibilita classificar as amostras da base de dados.

Nas próximas seções serão descritos detalhes da implementação do programa e o resultado dos experimentos realizados com diferentes bases de dados e valores dos parâmetros da GP.

2 Implementação

A implementação do programa foi feita com a linguagem Python. Várias decisões tiveram que ser tomadas durante a codificação da GP. Essas decisões serão descritas nas subseções a seguir.

2.1 Representação do Indivíduo

A população consiste de indivíduos, onde cada indivíduo representa uma possível solução para o problema, ou seja, uma possível função de distância. Para representar essas funções foi utilizada uma estrutura de árvore binária, onde os nós internos representam as funções e os nós folhas representam os terminais. Mais detalhes sobre o conjunto de funções e de terminais serão apresentados na próxima seção.

Um exemplo da representação das funções como árvore é apresentado na Listing 1, onde é apresentada a árvore que representa a função $(0.5 * var5) + var10$. Cada nó interno tem dois filhos, que podem ser outras funções ou terminais.

A classe que representa essas árvores pode ser encontrada no arquivo `function_tree.py`.

Listing 1: Representação da função $(0.5 * var5) + var10$ como árvore

```
SUM
        MUL
                0.5
                var5
        var10
```

2.2 Conjunto de Funções e Terminais

Para o conjunto de funções, foram usadas as funções aritméticas básicas: soma, subtração, multiplicação e divisão. A operação de divisão foi alterada para retornar 0 quando o divisor for igual a 0, de forma a manter a propriedade de Fechamento.

O conjunto de Terminais foi dividido em dois subconjuntos: Constantes e Variáveis. O conjunto de constantes é definido como os números reais no intervalo de $[-1, 1]$ e foi adicionado para aumentar o poder de expressão das funções candidatas (propriedade da Suficiência). O conjunto das variáveis se refere as valores que representam os pares de exemplos que são passados para a função. Por exemplo, na base de dados `breast_cancer_coimbra`, os exemplos são representados por 9 variáveis, totalizando 18 variáveis disponíveis dentro da função.

Quando um nó de função é criado, uma das quatro funções apresentadas anteriormente é selecionada de maneira aleatória. Da mesma forma, quando um nó de constante é criado, um número real no intervalo especificado é selecionado aleatoriamente e quando um nó de variável é criado, uma variável é selecionada aleatoriamente.[1]

Os conjuntos de funções e terminais e as funções que os selecionam podem ser encontrados no arquivo `symbols.py`.

2.3 Inicialização da População

A população é inicializada usando o método *Ramped half-and-half*, que combina os métodos *full* e *grow* para aumentar a diversidade da população. Com isso, durante a inicialização a população é dividida em grupos de forma a produzir árvores com todas as alturas possíveis (considerando o parâmetro da altura máxima), sendo que metade dessas árvores é inicializada com o método *full*, que produz árvores balanceadas, e a outra metade é inicializada com o método *grow*, que produz árvore com forma irregulares.

A implementação desse método de inicialização pode ser encontrada nos seguintes métodos presentes no arquivo `genetics.py`:

- `initialize_population`
- `init_tree_full`
- `init_tree_grow`

2.4 Fitness e Seleção

Para calcular a fitness de um indivíduo, o algoritmo de *clustering* é executado usando a função representada por aquele indivíduo como métrica de distância. Após a execução desse algoritmo, o valor da fitness do indivíduo é dado pelo *V-measure score*, que é calculado usando os resultados da classificação do algoritmo de *clustering* e os rótulos *ground truth*. As funções que executam o algoritmo de *clustering* e calculam o *V-measure score* estão no arquivo `clustering.py`.

A seleção é feita usando torneio. Assim, um subconjunto de k indivíduos (onde k é um parâmetro do programa) é escolhido aleatoriamente na população atual e o indivíduo com a maior fitness entre os k indivíduos desse subconjunto é selecionado. Essa seleção é implementada na função `k_tournament`, no arquivo `genetics.py`.

[1] As variáveis são identificadas por meio da posição do vetor obtido concatenando as representações dos dois exemplos passados para a função. Por exemplo, na base `breast_cancer_coimbra`, onde cada exemplo possui 9 variáveis, o nó com a variável 0 retorna o valor da primeira variável do primeiro exemplo e um nó com a variável 9 retorna o valor da primeira variável do segundo exemplo.

2.5 Operadores Genéticos

Foram utilizados os operadores genéticos de cruzamento e mutação para a criação da nova população. Na operação de cruzamento, são selecionados dois nós na mesma altura da árvore em cada um dos pais e é realizada a troca dessas subárvores entre os pais. Esse método é implementado na função `crossover` no arquivo `genetics.py`.

Para a operação de mutação foram usadas três formas diferentes de mutar um indivíduo:

- Mutação de ponto: implementada na função `point_mutation`, nessa mutação o valor de um dado nó é alterado de forma aleatória. Assim, um nó do tipo função que é uma soma pode se tornar uma multiplicação. Da mesma forma, um nó de variável por mudar a variável a qual ele faz referência;
- Mutação de Expansão: implementada em `expansion_mutation`, um nó terminal (constante ou variável) é selecionado aleatoriamente e é alterado por uma nova subárvore, que pode ser inicializada usando o método *grow* ou *full*;
- Mutação de Redução: implementada em `reduction_mutation`, ela faz o contrário da mutação anterior, selecionando aleatoriamente um nó do tipo função e o transformando em um nó terminal (constante ou variável), reduzindo a árvore.

Cada operador possui uma probabilidade associada: p_c para cruzamento e p_m para mutação. Essas probabilidades são parâmetros para o programa e são usadas para escolher qual operação será utilizada durante a criação da nova população em cada geração (loop for na função `execute_trials` no arquivo `main.py`). Se a mutação for escolhida, é realizado um novo sorteio para decidir qual será o tipo de mutação aplicado, sendo que cada tipo tem a mesma probabilidade de ser escolhido.

2.6 Base de Dados

Foram utilizadas as bases de dados `breast_cancer_coimbra` e `glass` para testar o programa implementado. Os dados são lidos dos arquivos `csv`, sendo que, para cada base, há um arquivo com os dados de treino e outro com os dados de teste. Os dados são normalizados usando `sklearn.preprocessing.MinMaxScaler` para que fiquem no intervalo $[0, 1]$. As funções que lidam com os dados estão no arquivo `data.py`.

2.7 Parâmetros e Execução

Como foi mencionado nas últimas seções, há vários parâmetros que determinam como o programa irá funcionar. Eles podem ser facilmente alterados por meio dos argumentos passados quando o programa é executado. Uma lista com esses parâmetros é apresentada a seguir:

- trials: número de vezes que o algoritmo será executado;
- population_size: número de indivíduos da população;
- generations: número de gerações;

- function_tree_size: tamanho máximo do indivíduo (altura da árvore);
- crossover_prob: probabilidade de realizar cruzamento;
- mutation_prob: probabilidade de realizar mutação;
- tournament_k: parâmetro k da seleção por torneio;
- elitism: usar ou não elitismo.

O script principal `main.py` contém a função principal que irá, para cada uma das n trials, executar a GP com os passos de inicialização da população, cálculo da fitness, seleção de indivíduos e aplicação dos operadores genéticos. Após a execução das n trials, os resultados são salvos no diretório que foi passado no argumento `output_path`: um arquivo `log.txt` com os parâmetros utilizados e as melhores fitness encontradas no treino e no teste; gráficos de fitness, indivíduos iguais na população e comparação entre filhos e pais para os operadores de cruzamento e mutação.

Para executar o programa use `python main.py`, acrescentando os argumentos necessários. Para obter a lista completa dos parâmetros aceitos pelo programa, use `python main.py -h`. O script `run.sh` possui um exemplo de execução do programa com os parâmetros. Para executá-lo, use o comando `./run.sh` em um terminal.

Os seguintes pacotes são necessários para execução do programa:

- numpy
- pyclustering
- matplotlib
- scikit-learn
- pandas

3 Experimentos

Foram realizados vários experimentos com o intuito de se obter os melhores valores para os parâmetros da GP discutidos na seção anterior. Nas próximas subseções, serão apresentados os resultados desses experimentos. Os valores apresentados são a média e o desvio padrão dos resultados obtidos em 10 execuções independentes.

3.1 Escolha do tamanho da população e número de gerações

Para definir o tamanho da população e o número de gerações, foram feitos experimentos na base de dados `breast_cancer_coimbra`. O número de gerações foi fixado em um valor alto, 200, e foram testados diferentes valores para o tamanho da população: 50, 150, 250 e 350. Na Tabela 1 são apresentados os melhores valores de fitness obtidos no treino e no teste. Na Figura 1 são apresentados os gráficos Fitness x Geração para os diferentes valores de

população testados. Nestes gráficos estão plotados os valores máximo, mínimo e médio de fitness obtida em cada uma das 200 gerações.

Analisando os resultados dispostos na Tabela 1, podemos verificar que, como esperado, o maior valor de fitness obtido nos dados de treino foi obtido com o maior tamanho da população testado, 350. Isso pode ser explicado pela maior diversidade da população induzida pelo maior número de indivíduos. Quando olhamos para o teste, podemos notar que, para todos os tamanhos de população, houve uma grande queda na melhor fitness encontrada no teste quando comparada à encontrada no treino, o que mostra a dificuldade em se generalizar a função encontrada para dados fora do conjunto de treinamento. Assim, com base nessa análise, o valor do tamanho da população foi definido em 100, que é um meio termo entre 50 e 150. Esse valor foi escolhido, pois, para valores maiores, o ganho na melhor fitness encontrada não justifica o maior custo computacional causado pelo maior número de indivíduos na população.

Quanto às gerações, por meio da análise dos gráficos dispostos na Figura 1, podemos verificar que após a 100ª geração, há apenas um pequeno aumento nos valores de fitness máximo e médio obtidos. Logo, de maneira análoga ao tamanho da população, o número de gerações foi definido como 100, uma vez que o pequeno aumento obtido no valor de fitness não justifica o grande aumento do custo computacional causado por um número maior de gerações.

Tab. 1: Melhores fitness na base de dados `breast_cancer_coimbra` variando o tamanho da população

	Best Fitness Train	Best Fitness Test
Population 50, Generations 200	0.2463 ± 0.0597	0.0164 ± 0.0230
Population 150, Generations 200	0.2689 ± 0.0534	0.1358 ± 0.1258
Population 250, Generations 200	0.2808 ± 0.0480	0.1086 ± 0.1232
Population 350, Generations 200	0.3492 ± 0.0112	0.1078 ± 0.0817

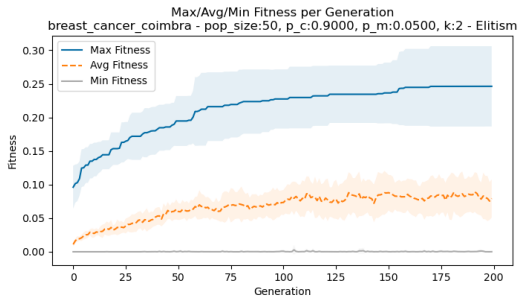
3.2 Parâmetros para cruzamento e mutação

Foram realizados experimentos com duas configurações para os parâmetros da probabilidade de ocorrer cruzamento (p_c) e mutação (p_m). Os resultados estão dispostos na Tabela 2. Os gráficos com os resultados dos experimentos estão dispostos na Figura 2.

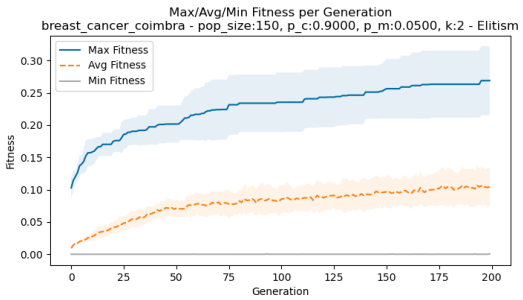
Tab. 2: Melhores fitness encontradas na base de dados `breast_cancer_coimbra` variando as probabilidades de cruzamento e mutação

	Best Fitness Train	Best Fitness Test
$p_c = 0.9, p_m = 0.05$	0.2525 ± 0.0453	0.0986 ± 0.0740
$p_c = 0.6, p_m = 0.3$	0.2535 ± 0.0504	0.0697 ± 0.1030

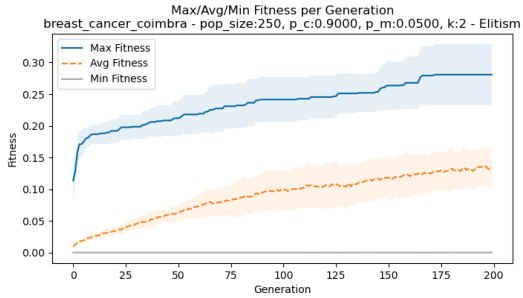
Analisando a Tabela 2, podemos verificar que ambas configurações tiveram resultados parecidos no treino. Entretanto, a primeira configuração, com $p_c = 0.9, p_m = 0.05$, obteve os melhores resultados no teste. Analisando os gráficos 2a e 2b, podemos confirmar que a



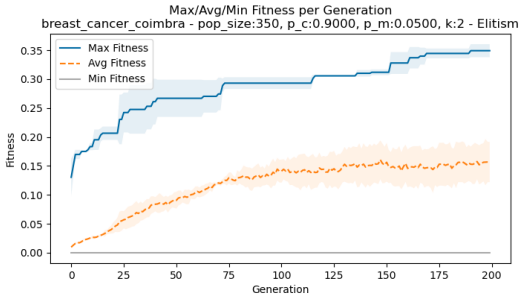
(a) Population 50, Generations 200



(b) Population 150, Generations 200



(c) Population 250, Generations 200



(d) Population 350, Generations 200

Fig. 1: Gráficos Fitness x Geração, onde são apresentados o maior, o menor e o valor médio de fitness obtida por geração para diferentes tamanhos da população.

mudança desses parâmetros não resultou em uma grande mudança no comportamento da fitness.

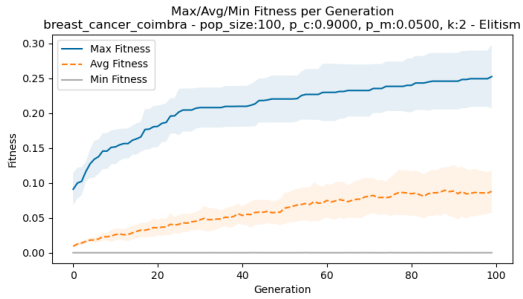
Os gráficos 2c e 2d permitem a análise do efeito do operador do cruzamento na população. Neles estão dispostos o número de filhos que têm fitness melhores que a média dos pais em azul, e o número de filhos que tiveram fitness piores que essa média em vermelho. Podemos notar que a maior parte dos filhos são piores que os pais, mas que, ainda assim, vários filhos são melhores.

Da mesma forma, os gráficos 2e e 2f permitem a análise do efeito do operador de mutação. Podemos notar que o operador da mutação, no geral, tem um efeito mais positivo na população quando comparado com o cruzamento, uma vez que uma maior porcentagem dos filhos gerados por esse operador têm fitness maiores que os pais. Entretanto, como esse efeito positivo da mutação não resultou em uma grande diferença na fitness obtida ao final do treinamento, o restante dos experimentos foram executados com os parâmetros $p_c = 0.9, p_m = 0.05$.

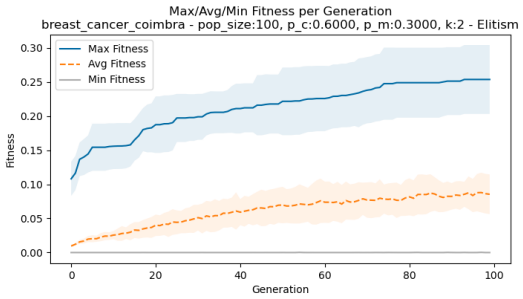
3.3 Tamanho do Torneiro

Com a definição dos parâmetros conforme discutido nas seções anteriores, foi realizado um experimento para avaliar o efeito do tamanho do torneio. Os resultados são apresentados na Tabela 3 e na Figura 3.

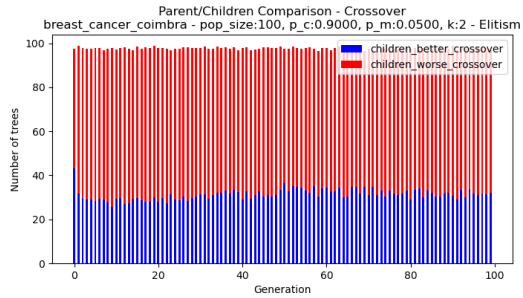
Analisando a Tabela 3, podemos notar que o aumento do tamanho do torneio resultou em um pequeno aumento na fitness do treino e uma pequena queda na fitness de teste. Os gráficos 3a e 3b que o aumento do tamanho do torneio fez com que a fitness máxima



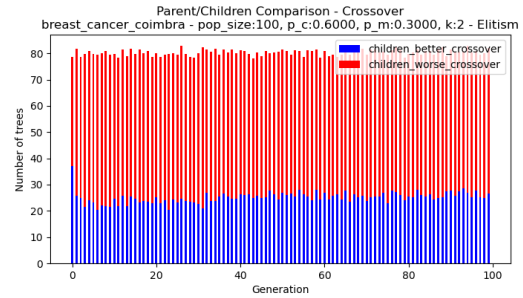
(a) $p_c = 0.9, p_m = 0.05$



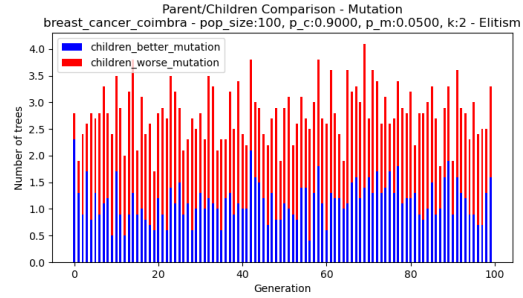
(b) $p_c = 0.6, p_m = 0.3$



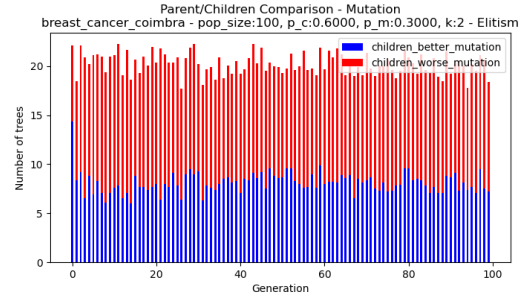
(c) $p_c = 0.9, p_m = 0.05$



(d) $p_c = 0.6, p_m = 0.3$



(e) $p_c = 0.9, p_m = 0.05$



(f) $p_c = 0.6, p_m = 0.3$

Fig. 2: Gráficos com os resultados dos experimentos variando as probablidades de cruzamento e mutação. Em 2a e 2b, são apresentados os gráficos Fitness x Geração. Em 2c e 2d são apresentados os gráficos que mostram o número de filhos com fitness melhor que a média dos pais após o cruzamento e o número de filhos com fitness pior. Em 2e e 2f são apresentados os gráficos que mostram o número de filhos com fitness melhor o pai após mutação e o número de filhos com fitness pior.

encontrada e, principalmente, a fitness média subissem mais rapidamente. Todavia, o valor máximo de fitness encontrado ao fim do treino não sofreu grande alteração.

Os gráficos 3c, 3d, 3e e 3f ilustram bem o efeito de uma maior pressão seletiva. É possível notar claramente que o tamanho do torneio maior fez com que o número de filhos melhores que os pais, tanto na mutação quanto no cruzamento, caísse consideravelmente. Isso pode ser justificado pelo aumento da pressão seletiva, uma vez que, com um maior k, a probabilidade de selecionar os melhores indivíduos da população aumenta, fazendo que seja menos provável que o filho tenha uma fitness melhor que o pai.

Tab. 3: Melhores fitness encontradas na base de dados `breast_cancer_coimbra` variando o tamanho do torneio

	Best Fitness Train	Best Fitness Test
$k = 2$	0.2525 ± 0.0453	0.0986 ± 0.0740
$k = 5$	0.2653 ± 0.0449	0.0758 ± 0.1639

3.4 Efeito do Elitismo

Foi realizado um experimento para avaliar o efeito do elitismo nos resultados obtidos. Os resultados estão dispostos na Tabela 4 e na Figura 4.

Analisando a Tabela 4, podemos notar que a ausência do Elitismo fez com que a melhor fitness encontrada no treino sofresse uma pequena redução. Em contrapartida, a melhor fitness encontrada no teste teve um pequeno aumento.

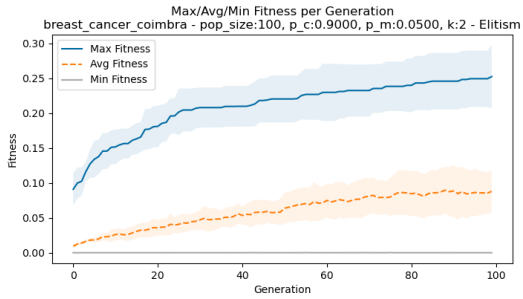
Nos Gráficos 4a e 4b o efeito do elitismo fica mais claro. Quando há elitismo (Gráfico 4a), a curva da fitness máxima é não-decrescente e as curvas no geral são bem comportadas. Já no Gráfico 4b, onde não há elitismo, a curva de fitness máxima decresce em alguns pontos, onde o melhor indivíduo de uma geração é descartado após a aplicação dos operadores genéticos. Também é possível notar que, quando não há elitismo, as fitness máximas assumem um valor menor, ficando mais próxima da fitness média da população.

Tab. 4: Melhores fitness encontradas na base de dados `breast_cancer_coimbra` utilizado ou não o Elitismo

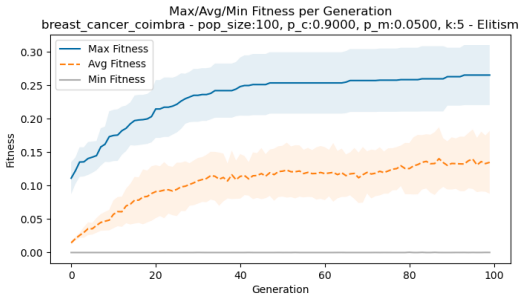
	Best Fitness Train	Best Fitness Test
Com Elitismo	0.2653 ± 0.0449	0.0758 ± 0.1639
Sem Elitismo	0.2248 ± 0.0617	0.1120 ± 0.1296

3.5 Indivíduos Iguais na População

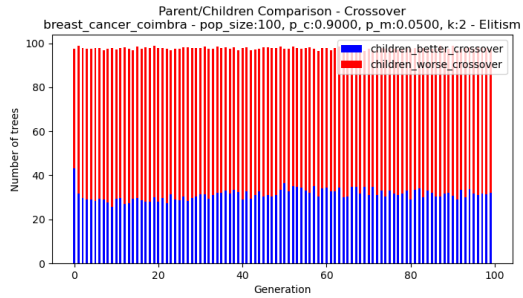
A fim de se verificar a diversidade da população, foi computada a estatística de quantos indivíduos iguais existiam na população. Os resultados estão plotados no Gráfico 5. É possível notar que, de fato, cerca de 5% da população é constituída de indivíduos iguais. Como esse percentual não é muito alto, podemos dizer que há um certo nível de diversidade na população.



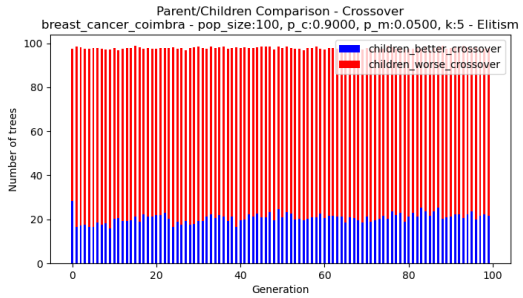
(a) $k = 2$



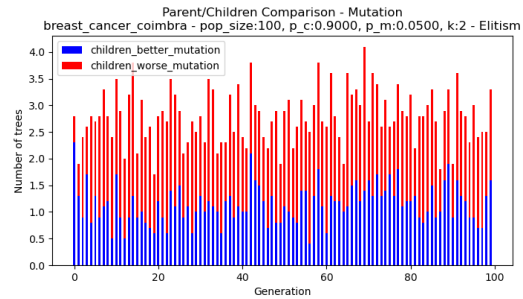
(b) $k = 5$



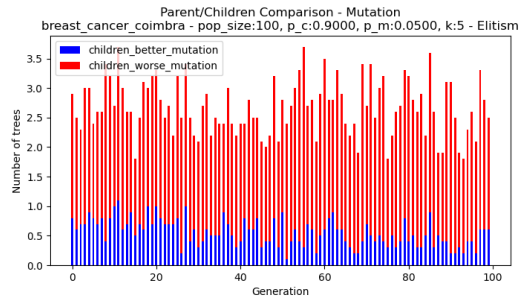
(c) $k = 2$



(d) $k = 5$

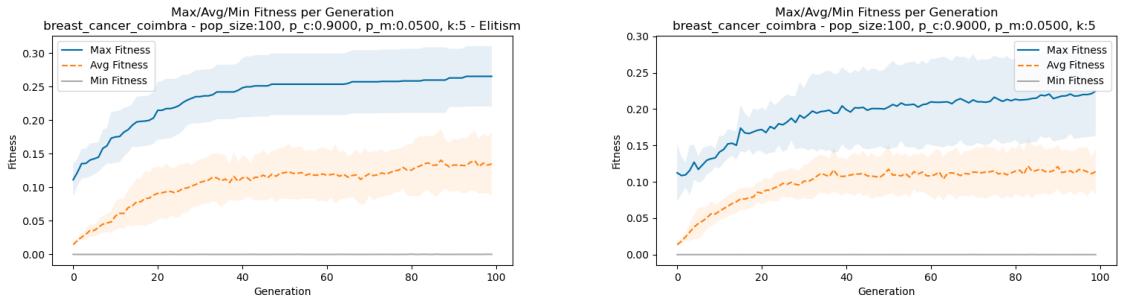


(e) $k = 2$



(f) $k = 5$

Fig. 3: Gráficos com os resultados dos experimentos variando o tamanho do torneio k. Em 3a e 3b são apresentados os gráficos Fitness x Geração. Em 3c e 3d são apresentados os gráficos que mostram o número de filhos com fitness melhor que a média dos pais após o cruzamento e o número de filhos com fitness pior. Em 3e e 3f são apresentados os gráficos que mostram o número de filhos com fitness melhor o pai após mutação e o número de filhos com fitness pior.



(a) Com Elitismo (b) Sem Elitismo

Fig. 4: Gráficos Fitness x Geração com Elitismo, 4a, e sem Elitism, 4b.

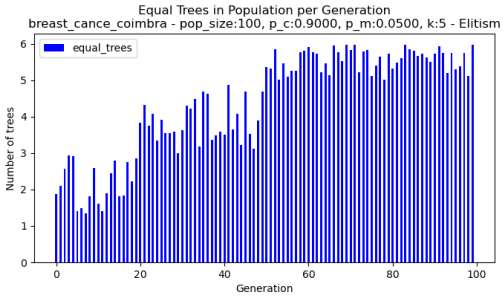


Fig. 5: Gráfico com a quantidade de indivíduos iguais na população de tamanho 100.

3.6 Resultados na base de dados `glass`

Os resultados apresentados nas últimas seções são de experimentos realizados usando a base de dados `breast_cancer_coimbra`. Para verificar o desempenho do programa em outros dados, foi feito um experimento rodando o programa com dados da base `glass` usando os melhores parâmetros obtidos anteriormente. Os resultados desse experimento são apresentados na Tabela 5 e Figura 6.

Analisando esses resultados, podemos notar que o algoritmo alcançou um valor consideravelmente maior de fitness, tanto no treino quanto no teste, nessa base. Isso pode indicar que os dados da base `glass` tem uma complexidade menor, o que torna mais fácil classificá-los.

Tab. 5: Melhores fitness encontradas na base de dados `glass` utilizado os melhores parâmetros encontrados.

Best Fitness Train	Best Fitness Test
0.4635 ± 0.0121	0.4686 ± 0.1105

4 Conclusão

Neste trabalho foi implementado um algoritmo de Programação Genética (GP) que usa a técnica de regressão simbólica para encontrar uma função que mede a distância entre pares

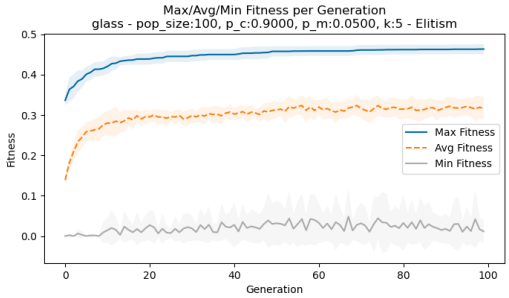


Fig. 6: Gráfico Fitness x Gerações para os dados da base `glass`, usando os melhores valores obtidos para os parâmetros.

de exemplos de uma base de dados. Essa função foi utilizada para ajustar um algoritmo de clusterização, e os clusters obtidos eram então utilizados para classificar as amostras.

Neste documento, foram apresentadas as decisões de implementação tomadas e os resultados de vários experimentos que foram realizados com o programa desenvolvido, a fim de se obter o efeito de cada parâmetro da GP no resultado obtido.

References

[1] Pappa, G. L. Slides sobre programaçao genética e algoritmos evolucionários disponibilizados no moodle, 2021.