Regressão Simbólica com Programação Genética
TP1 – Computação Natural – 2018/2

Alexandre Maros[1]

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

alexandremaros@dcc.ufmg.br

1. Introdução

Algoritmos genéticos vem sendo utilizados para resolver inúmeros problemas de otimização ao longo das últimas décadas e conta com inúmeras sub-áreas. Uma dessas sub-áreas dos algoritmos genéticos que também vem sendo muito utilizada é a programação genética [Altenberg 1994]. Um dos grandes problemas contornados pela programação genética é a de não exigir que os genótipos dos indivíduos sejam representados por modelagens de tamanho fixo, permitindo assim a evolução de indivíduos com tamanhos diferentes.

A ideia da programação genética permite que inúmeros problemas sejam resolvidos através dessa ideia, desde a modelagem de antenas [Hornby et al. 2006] até a criação de indivíduos que conseguem jogar jogos de videogame, como o jogo Tetris [Yurovitsky 1995].

Um outro problema que será abordado neste trabalho e que também já foi muito estudado na literatura é a de regressão simbólica [Searson et al. 2010, Augusto and Barbosa 2000, Uy et al. 2011]. A regressão simbólica consiste em, dado um conjunto de pontos de treino, estimar a função que gera aqueles pontos. Um exemplo do problema pode ser visualizado na Figura 1.

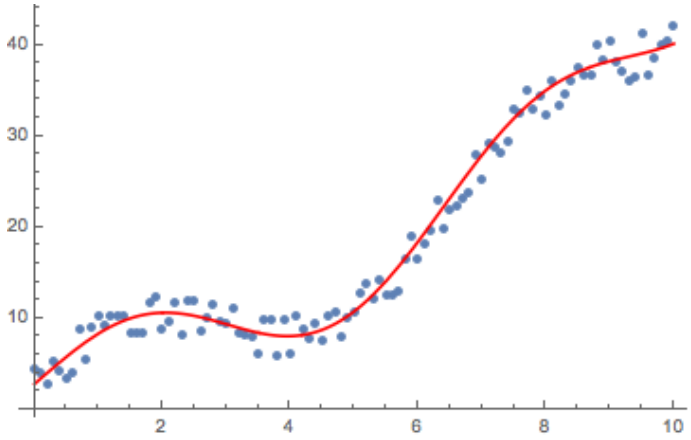


Figura 1. Exemplo de regressão simbólica. Dado um conjunto de pontos (em azul) é necessário estimar a função que gerou esses pontos (em vermelho). [Stork 2015]

A Figura 1 ilustra o problema da regressão simbólica. Os pontos em azul são os pontos de treino e a função a ser estimada é descrita pela curva em vermelho. É um

problema complexo e muito propenso a *overfitting* (caso em que a curva se torna tão específica para os pontos de treino que não há nenhuma generalização e acaba não se adequando a curva real).

Este trabalho tem como principal objetivo explorar o problema de regressão simbólica e propor uma modelagem para resolver o problema. A Seção 2 faz a definição formal do problema e a modelagem do algoritmo proposto, a Seção 3 descreve os *datasets* aqui abordados para testar a modelagem da seção anterior, a Seção 4 comenta os experimentos realizados e os resultados obtidos, e por fim, na Seção 5 é feito a conclusão e as considerações finais sobre o trabalho.

2. Definição e Modelagem do problema

Nesta Seção é feito a definição formal do problema seguido da modelagem proposta para resolve-lo. Na Seção 2.1 é discutido a definição formal do problema e como é avaliado a qualidade de solução de um indivíduo e na Seção 2.2 é discutido como os indívíduos são modelados.

2.1. Definição

O problema de regressão simbólica consiste em, dado um conjunto de m amostras, ou pontos no espaço, provenientes de uma função desconhecida $f : \mathbb{R}^n \rightarrow \mathbb{R}$, representado por uma dupla (X, Y) onde $X \in \mathbb{R}^{m \times n}$ e $Y \in \mathbb{R}^m$, o objetivo é encontrar a expressão simbólica de f que melhor se ajuste às amostras fornecidas. Vale ressaltar que a função é original é desconhecida, há apenas uma pequena quantidade de pontos disponíveis para estimar a curva.

Para validar o quão boa uma solução (ou nesse caso, um indivíduo) realmente é, torna-se necessário uma função de erro para estimar o quanto o modelo está longe dos pontos de treinamento, asssim podendo avaliar de uma forma numérica a qualidade da solução. Para isso, é utilizado a raiz quadrada do erro quadrático médio normalizado (NRMSE), definido na Equação 1, onde N é o número de instâncias, ou pontos de treinamento, Ind é o indivíduo, ou a solução sendo avaliada, $EVAL(Ind, x_i)$ é a avaliação ou previsão da função para o ponto x_i, y_i é a saída esperada para a entrada x_i e $\hat{Y}$ é a média do vetor de saídas Y.

$$fitness(Ind) = \sqrt{\frac{\sum_{i=1}^{N} (y_i - EVAL(Ind, x_i))^2}{\sum_{i=1}^{N} (y_i - \hat{Y})^2}} \tag{1}$$

Para avaliar a qualidade da solução de um individuo, podem ser levadas em consideração inúmeras métricas, como o RMSE, ou o MAPE, entretanto, o NRMSE é uma boa métrica a ser considerada pois facilita a comparação de resultados entre *datasets* diferentes, já que o intervalo dessa métrica se torna parecido entre um *dataset* e outro.

A execução do algoritmo de programação é dado pela máquina de estados da Figura 2. Inicialmente é criado uma população inicial, seguido da avaliação de *fitness* de cada indivíduo, após isso é feito a seleção dos melhores indivíduos, seguido da aplicação de operadores genéticos e a criação da nova solução. O algoritmo para quando o número máximo de iterações é atingido. Nas próximas subseções cada etapa é explicada detalhadamente.

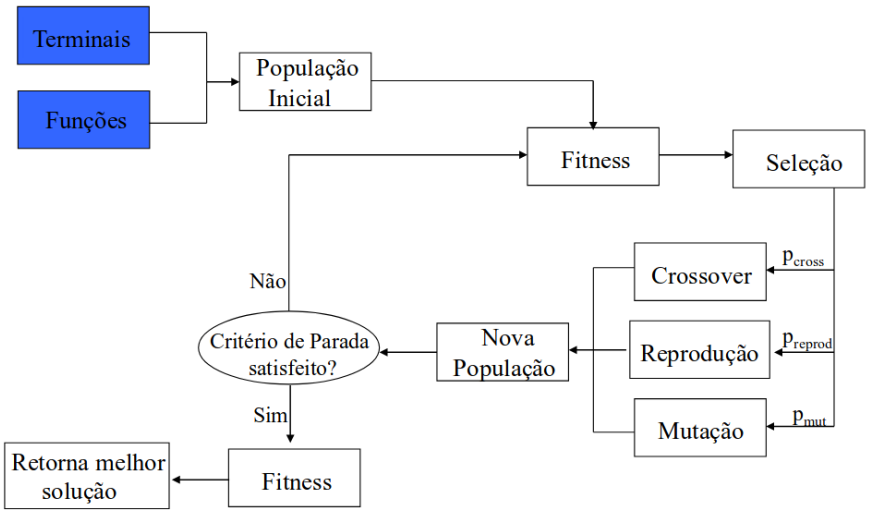


Figura 2. Máquina de estados do algoritmo

2.2. Modelagem do indivíduo

A modelagem do indivíduo foi definido pela representação através de uma árvore. A Figura 3 mostra a modelagem do indivíduo para melhor visualizar como o mesmo é representado.

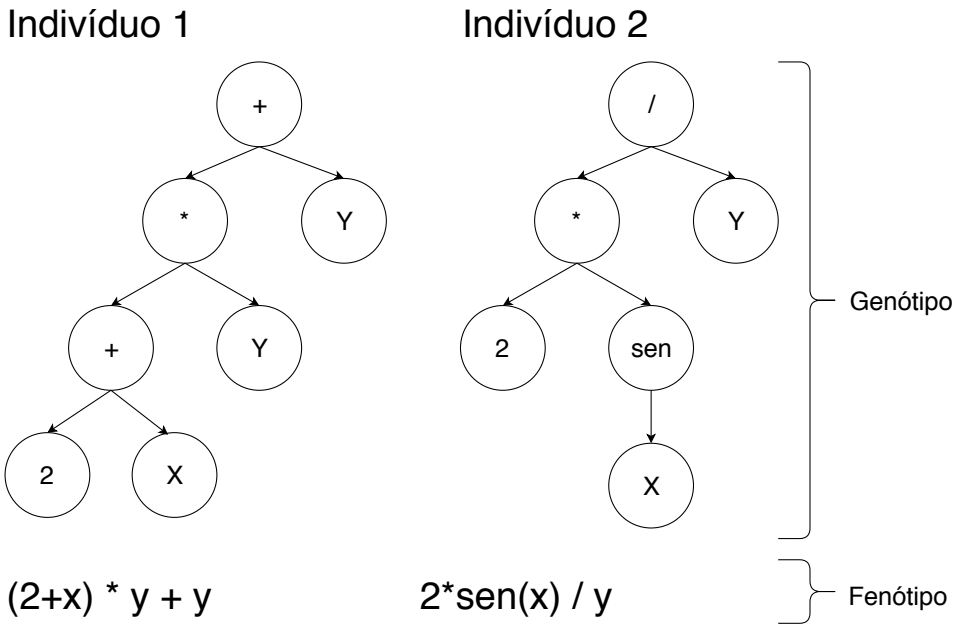


Figura 3. Loop de execução do algoritmo

Cada indivíduo é representado através de uma árvore binária (a árvore não necessariamente necessita ser binária mas facilita a visualização e modelagem do problema). Cada nó representa um terminal ou uma funão. Para o caso dos terminais, há três tipos:

- **Variável**: Retorna os valores de uma variável dado os pontos de entrada

$(x_1, x_2, x_3, ..., x_n)$. A variável escolhida para ser retornada é definida no momento do instanciamento do nó;

- **Valor**: Um valor escolhido aleatoriamente entre -2.0 e 2.0, os valores foram escolhidos apenas através de testes empíricos;
- **Seno**: Embora seja uma função aqui foi considerado como um terminal por possuir apenas um nó folha. A variável escolhida para a função é escolhida aleatoriamente durante seu instanciamento. **O seno somente é considerado nas bases *synth2* e *concrete* discutidas mais adiante.**

Para o caso das funções há quatro tipos:

- **Adição**: Necessita de dois nós folhas;
- **Subtração**: Necessita de dois nós folhas;
- **Multiplicação**: Necessita de dois nós folhas;
- **Divisão**: Necessita de dois nós folhas. Além disso, a operação de divisão é tratada de duas formas: Se acontecer uma divisão por 0, é retornado o valor 1. Também é feito o arredondamento do divisor para duas casas decimais. Tal tratamento foi feito pois estava ocorrendo de indivíduos retornarem valores absurdamente grandes em divisões por números muito pequenos.

O tamanho (altura) da árvore é um hiper-parâmetro, mas foi definido um valor máximo fixo de 7. No código fonte está como tamanho 6 pois o nó raiz é considerado como altura 0, logo a altura vai de 0 a 6.

2.2.1. Inicialização da população inicial

A inicialização da população inicial foi feita através do método *ramped half-and-half* por ser conhecido como uma boa inicialização para gerar indivíduos diversificados [Walker 2001]. A inicialização consiste em gerar metade dos indivíduos com o método *grow* e *full* variando o tamanho máximo da árvore até o tamanho máximo.

A técnica *grow* vai gerando a árvore recursivamente, podendo pegar nós tanto do conjunto de terminais como o de conjunto de funções. Caso seja pego nós do conjunto de terminais, aquele nó acaba nao podendo ser expandido até o limite, podendo gerar árvores menores.

Já a técnica *full* cria indivíduos até o tamanho máximo permitido, pegando apenas nós do conjunto de funções. Quando o limite máximo é atingido, as folhas então são geradas pegando nós do conjunto de terminais.

2.2.2. Seleção

A seleção é feita por torneio. k indivíduos são escolhidos aleatoriamente da população e o melhor dentre eles (o com a melhor *fitness*) é declarado o ganhador e passa para a fase de aplicação de operador genético. O valor de k é um hiper-parâmetro.

2.2.3. Operadores Genéticos

Há três operadores genéticos que podem ser aplicados em cima dos indivíduos. O *crossover*, a *reprodução* e a *mutação*. A probabilidade de cada um deles ocorrerem é definido através de hiper-parâmetros, embora neste trabalho só foi feito a variação dos operadores de *crossover* e *mutação*. A probabilidade de operador de *reprodução* ocorrer foi mantida em 0.01%, enquanto os dos dois outros foram estudados na parte de experimentação.

O operador de *crossover* funciona da seguinte forma: dois indivíduos são escolhidos através da seleção. Dois nós aleatórios de cada um dos indivíduos são escolhidos e são trocados. Caso um nó (subarvore) exceda o tamanho máximo permitido para o filho, dois outros nós aleatórios são escolhidos.

O operador de *mutação* funciona da seguinte forma: Um indivíduo aleatório é escolhido através da seleção. Após isso um nó é escolhido e retirado do individuo. Um novo nó é gerado a partir do método *grow* e posto no lugar desse nó, tomando cuidado para o indivíduo não exceder a altura máxima.

O operador de *reprodução* simplesmente passa o indivíduo escolhido para a nova população.

Também foi feita a implementação elitista desses operadores. Onde os filhos só são adicionados a nova população somente caso sua *fitness* é a melhor de que a de seus pais, caso contrário o melhor pai é adicionado a nova população.

2.2.4. Detalhes da implementação

A implementação foi feita utilizando a linguagem de programação Python 3.6, com auxilio de bibliotecas manipulação de números numpy. Além disso foi utilizado a biblioteca matplotlib para fazer os gráficos. Todas as bibliotecas utilizadas estão no arquivo *requirements.txt*, no diretório raiz. Atenção, para os gráficos, também é necessário ter a biblioteca tkinter instalada. Tal biblioteca foi utilizada como *backend* do matplotlib para poder gerar gráficos sem um *display* e necessita ser instalado via apt-get no Linux (apt install python3-tkinter).

Além disso, embora não tenha sido feito uma implementação paralela do algoritmo, diferentes execuções do algoritmo são executados em paralelo. Foi tomado o devido cuidado com o gerador de números aleatórios para não haver interferência entre uma execução e outra.

Por fim, um problema enfrentado na avaliação dos indivíduos foi de ter soluções que tinha um erro significativamente maior que os outros. Tais erros eram causados por indivíduos que contem operações de divisões encadeadas com números pequenos, resultando em números extremamente grandes. Esses indivíduos que "explodem"foram retirados do cálculo de média da *fitness*. Por esse motivo, também não foi considerado a métrica dos piores indivíduos, pois a média ficava extremamente grande, impossibilitando a análise dos indivíduos.

3. Descrição dos *datasets*

As bases de dados consideradas estão dispostos na Figura 4. Há dois *datasets* sintéticos, definidos pelas tags *synth1* e *synth2* e um *dataset* real, definido pela tag *concrete*.

Base	# atributos	# instâncias		Tipo
		Treino	Teste	
synth1	3	60	600	Sintética
synth2	3	300	1000	Sintética
concrete[1]	9	824	206	Real

Figura 4. *Datasets* estudados

Para o primeiro *dataset* não foi considerado a função seno como um possível terminal por não apresentar tal comportamento. Nas duas bases subsequentes foi considerado tanto a inclusão como a não inclusão da função seno, para verificar se tal comportamento estava presente. As experimentações mais extensivas foram realizadas sobre a base *synth1*.

4. Experimentação

Os testes foram conduzidos da seguinte forma. Para a base de dados *synth1* foi estudado os efeitos dos seguintes hiper-parâmetros: probabilidade dos operadores genéticos, a quantidade de gerações, o tamanho da população, o tamanho do torneio (k), e também o efeito de operadores elitistas. Também foi feito uma análise do indivíduo médio e o melhor indivíduo com a melhor combinação de hiper-parâmetros.

Para cada teste, foram feitas 30 execuções distintas para retirar a média e desvio-padrão. Além disso, a semente 814151623 foi fixada para não haver variação entre a população inicial entre cada teste. Os parâmetros que não foram analisados em cada caso de teste seguem esse padrão: 200 gerações, k = 5, população de 100 indivíduos, probabilidade de cruzamento de 70%, probabilidade de mutação de 0.29% e reprodução de 1%.

4.1. Teste no *dataset synth1*

4.1.1. Tamanho da população

Para a variação do número de indivíduos, foi escolhido quatro valores distintos: 50, 100, 200 e 500 indivíduos. A média dos indivíduos de cada geração pode ser conferida na Figura 5. O que se pode ver no gráfico é que, com 500 indivíduos, rapidamente se atinge convergência, já com 50 indivíduos essa convergência demora mais a chegar, sendo que os outros dois valores estão no meio desses dois extremos. Além disso, a população com 50 indivíduos possui uma maior variação entre cada geração do que os outros. A população com 100 e 200 indivíduos se mantiveram mais parecidas.

Na Figura 6 é possível verificar a média da melhor solução da última geração para a base de treino e de teste. O desvio padrão não foi aqui considerado por seus intervalos

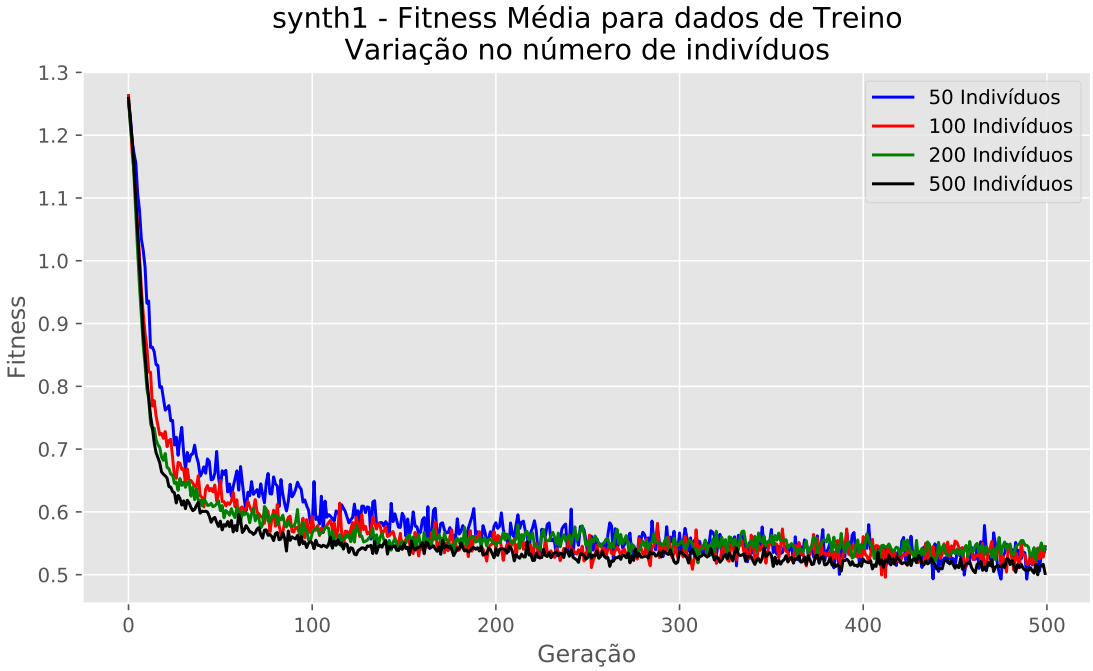


Figura 5. *synth1* variação tamanho da população

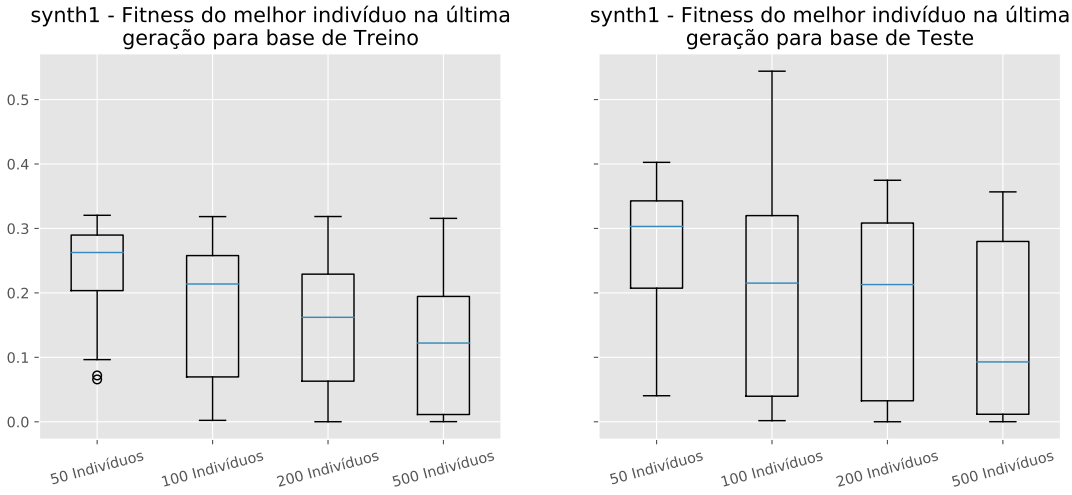


Figura 6. *synth1* variação tamanho da população, avaliação do último indivíduo

estarem se sobrepondo em todos os caoss. Na base de treino a melhor solução é a de 200 e 500 indivíduos (os intervalos das duas soluções se sobrepoem mais que as outras, logo tendo uma semelhança estatística). A pior solução foi a de 50 indivíduos. Para a base de teste, as duas melhores soluções foram as com 200 e 500 indivíduos também. Como a computação com 200 indivíduos é consideravelmente mais rápida, a sugestão do autor é de utilizar 200 indivíduos (500 indivíduos também parece convergir muito rapidamente).

4.1.2. Número de gerações

Para a avaliação do número de gerações foi testado os valores de 50, 100, 200 e 500 gerações. A Figura 7 mostra a curva das gerações (as curvas se sobrepõem e apenas a de 500 gerações se mantém a mesma nesse caso, devido a semente de cada teste ser a mesma).

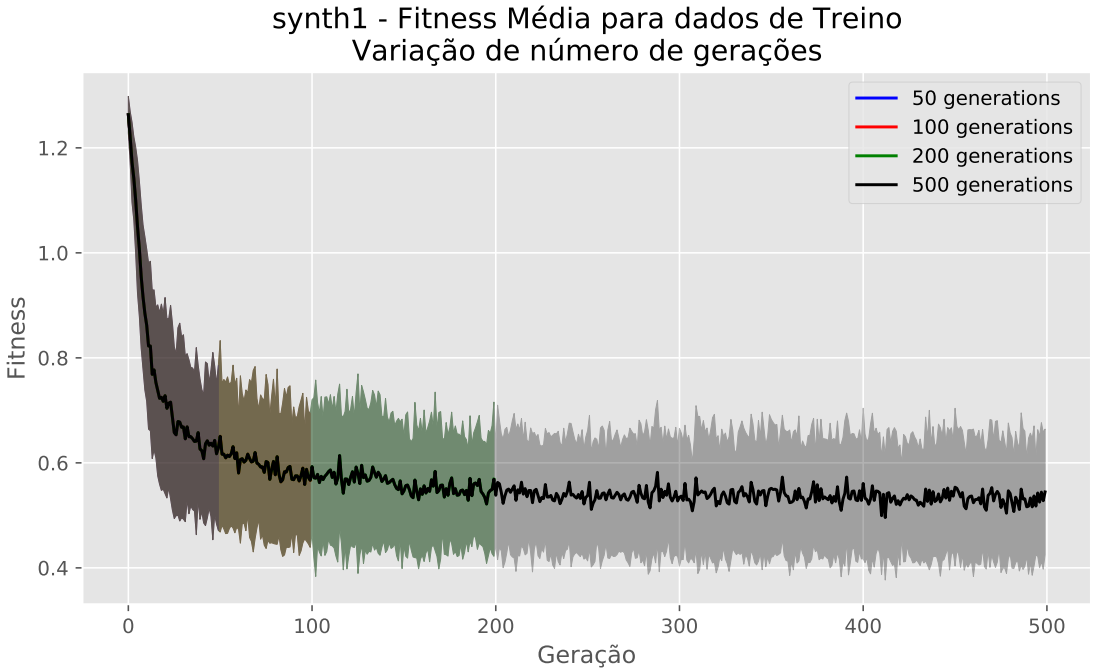


Figura 7. *synth1* variação do número de gerações

O gráfico mais interessante para esse caso de teste pode ser visto na Figura 8 onde é avaliado a solução do melhor indivíduo para a base de treino e teste.

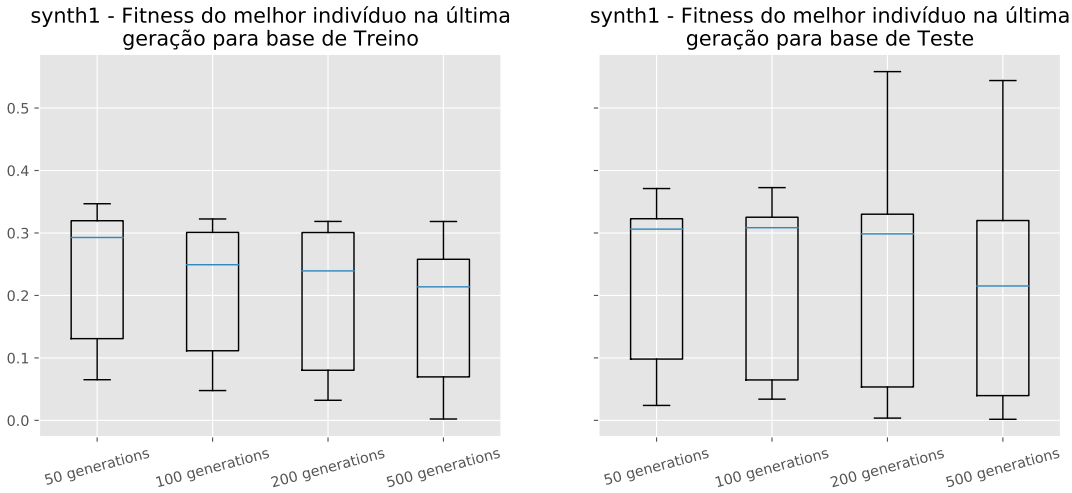


Figura 8. *synth1* variação do número de gerações, avaliação do último indivíduo

No gráfico da Figura 8 é possível notar que os melhores indivíduos para a base de treino foram gerados utilizando 500 gerações. Entretanto, para a base de teste, percebe-se que os indivíduos utilizando 500 gerações possuem um desvio padrão maior que os outros casos. Isso pode ser explicado pelo *overfitting* que o alto número de gerações traz. Para o resto dos teste foi utilizado o número de gerações de 200, que, embora também tenha um desvio padrão mais elevado, possui um menor percentil de 25% e parece ser um bom comprometimento entre uma maior busca, cuidando com o *overfitting*.

4.1.3. Probabilidade dos operadores genéticos

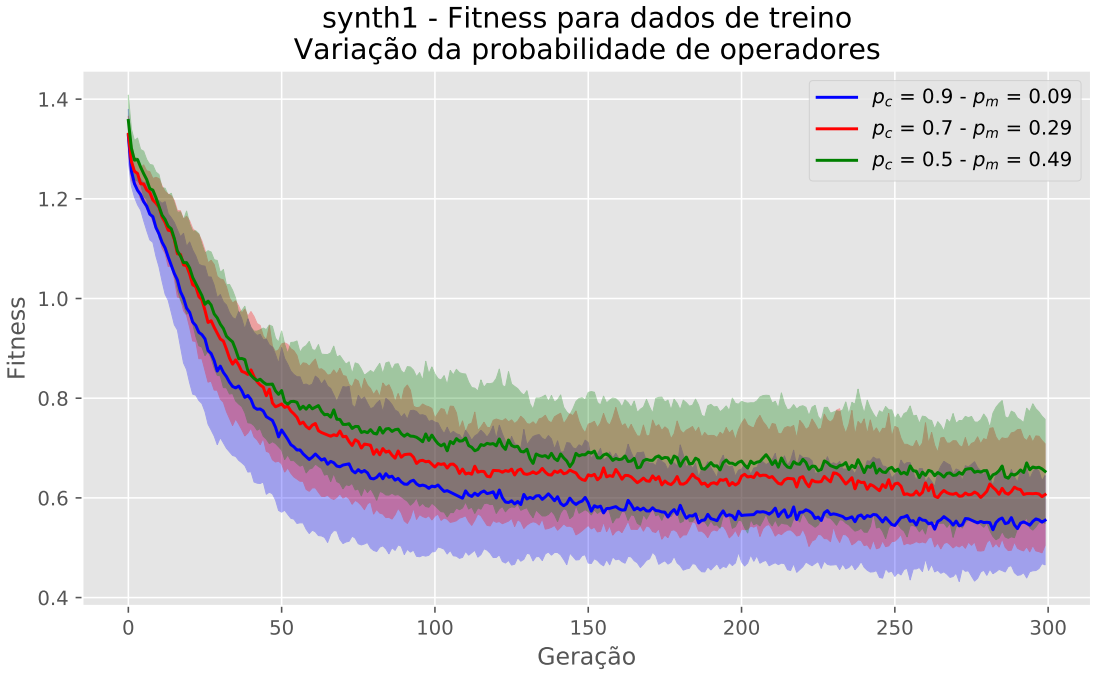


Figura 9. *synth1* variação do número de probabilidade dos operadores genéticos

Para a avaliação das probabilidades dos operadores genéticos foi utilizada as seguintes probabilidades: p_c de 0.9 e p_m de 0.09; p_c de 0.7 e p_m de 0.29; e p_c de 0.5 e p_m de 0.49. A probabilidade de reprodução é de 0.01 em todos os casos. A Figura 9 mostra as curvas para os três casos ao longo das gerações. Para esse caso em específico, foi considerado 300 gerações para avaliar o comportamento um pouco mais adiante do que foi sugerido no último teste (200 gerações). As três taxas ficaram parecidas, sendo que a taxa de cruzamento de 50% atingiu uma convergência mais rápida e a taxa de cruzamento de 90% aparentemente obteve a melhor *fitness*. Os desvio padrões dos três casos se sobrepõem em sua maioria, exigindo uma analise mais detalhada de cada caso.

Na Figura 10, na base de treino, a melhor solução foi encontrada com cruzamento de 90% e mutação de 0.09%, entretanto possui uma variação um pouco maior que as outras. Para a base de treino, o mesmo comportamento foi observado, sendo que a taxa de cruzamento de 90% aparenta ter o melhor comportamento.

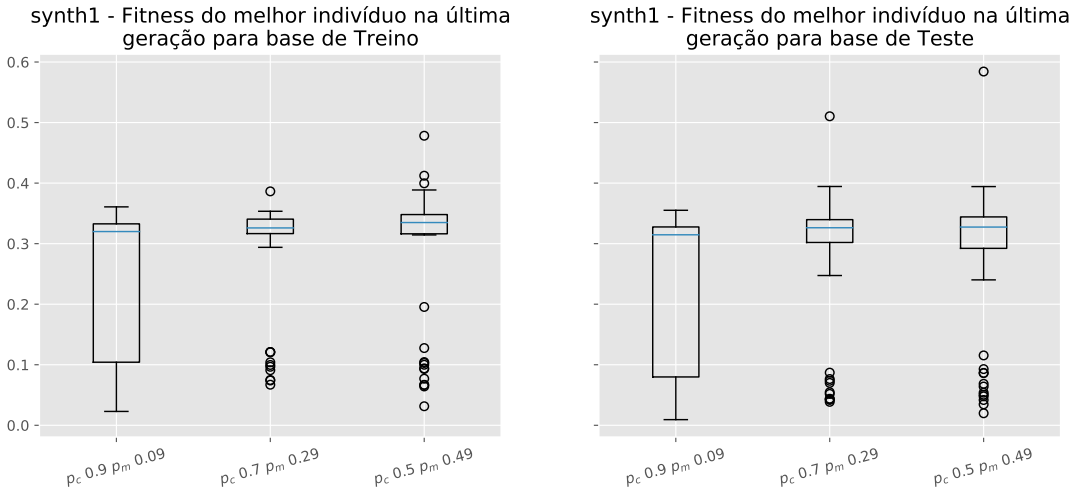


Figura 10. *synth1* variação do número de probabilidade dos operadores genéticos, avaliação do último indivíduo

4.1.4. Tamanho do torneio

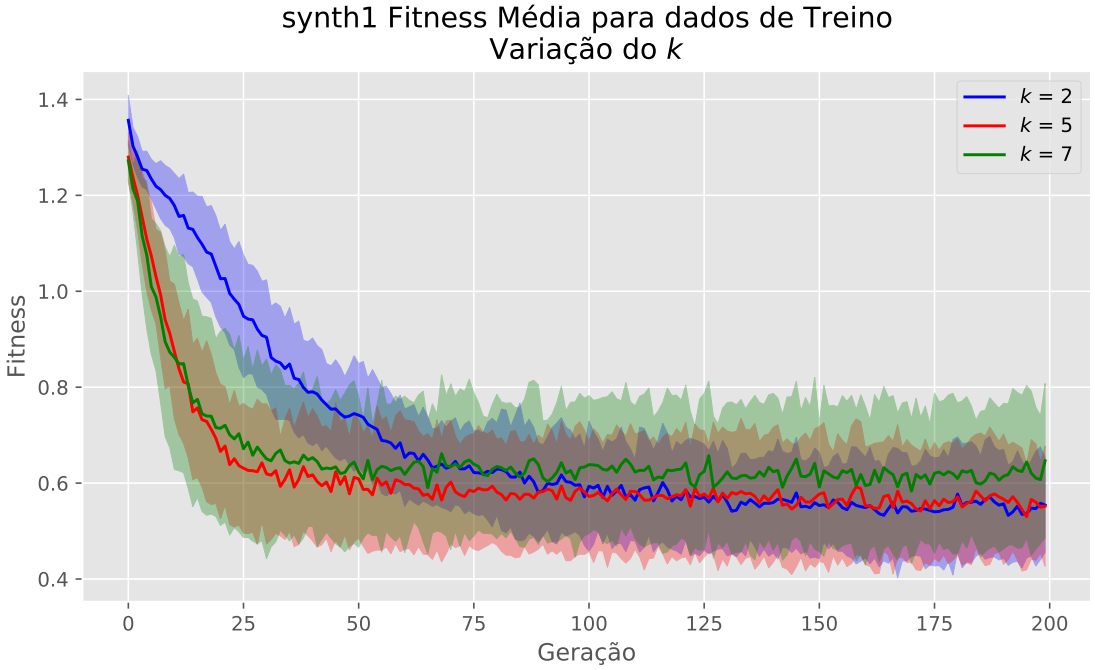


Figura 11. *synth1* variação do k no torneio

Para a experimentação do tamanho do torneio, foi selecionado um k de 2, 5 e 7. A Figura 11 mostra a curva da *fitness* média para cada caso. É possível identificar uma diferença significativa entre cada curva. Com $k = 2$ têm se uma exploração muito maior no início das gerações e atinge uma convergência apenas no final. Para $k = 5$ ou 7 a convergência é consideravelmente mais rápida, sendo que $k = 7$ atingiu uma pior *fitness* que os outros dois. Além disso, $k = 7$ possui uma variância mais elevada que os outros

dois casos. $k = 2$ possui o menor desvio padrão dos três.

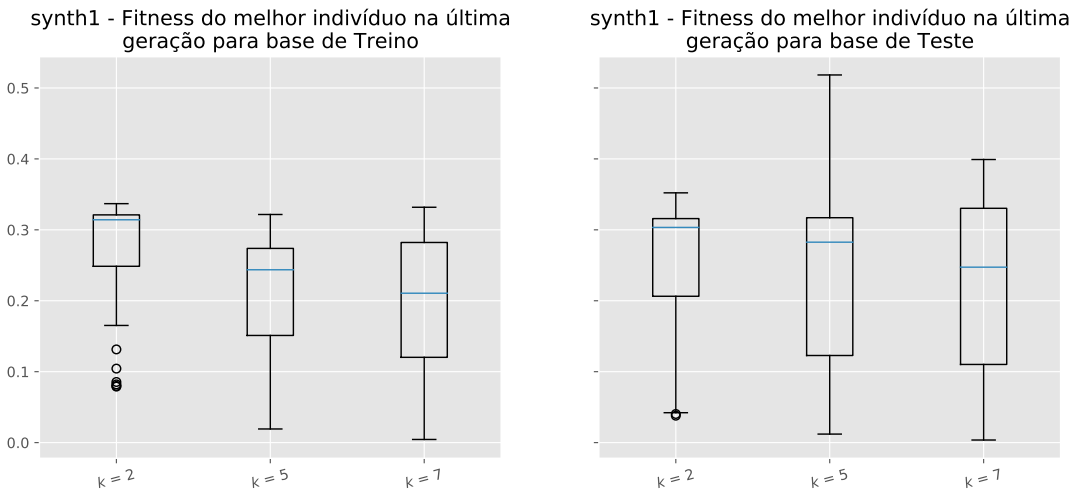


Figura 12. *synth1* variação do k no torneio, avaliação do último indivíduo

Para a Figura 12, é possível notar que k teve um desvio padrão menor tanto na base de treino quanto na base de testes. A mediana dos três testes se manteve parecidas. A sugestão é de utilizar um $k = 2$, para se ter um nível de exploração maior no início do algoritmo e um desvio padrão menor do que nos outros casos.

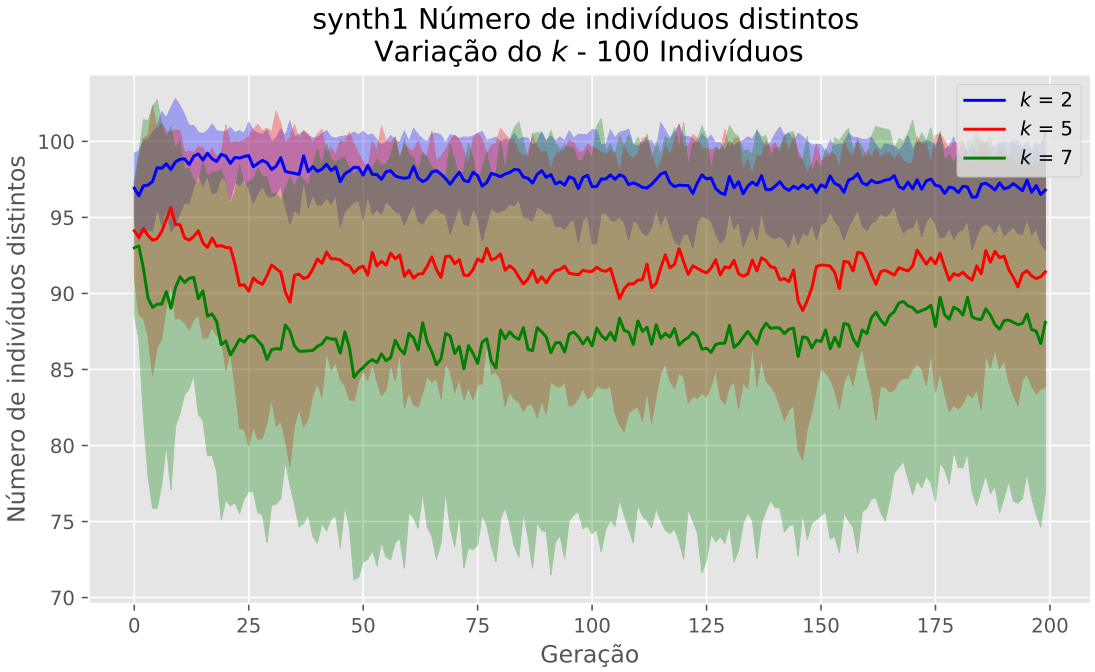


Figura 13. *synth1* média número de indivíduos distintos par cada k

Outra coisa interessante a ser analisada é a média do número de indivíduos distintos em cada geração para um certo k. Indivíduos são considerados repetidos caso tenham o mesmo fenótipo (e consequentemente, mesmo genótipo). A Figura 13 representa essa

métrica. É possível perceber pela figura que, quanto menor o k, mais indivíduos distintos são gerados. Tal informação faz sentido, já que com um k menor têm-se uma pressão seletiva menor logo abre mais espaço para a exploração do espaço de busca.

4.1.5. Operadores Elitistas

O último teste é em relação aos operadores elitistas. Na Figura 14 é possível visualizar o impacto desses operadores. Ao utilizar operadores elitistas, a convergência é extremamente mais rápida e o erro atingido na base de treino é significativamente menor que no outro caso.

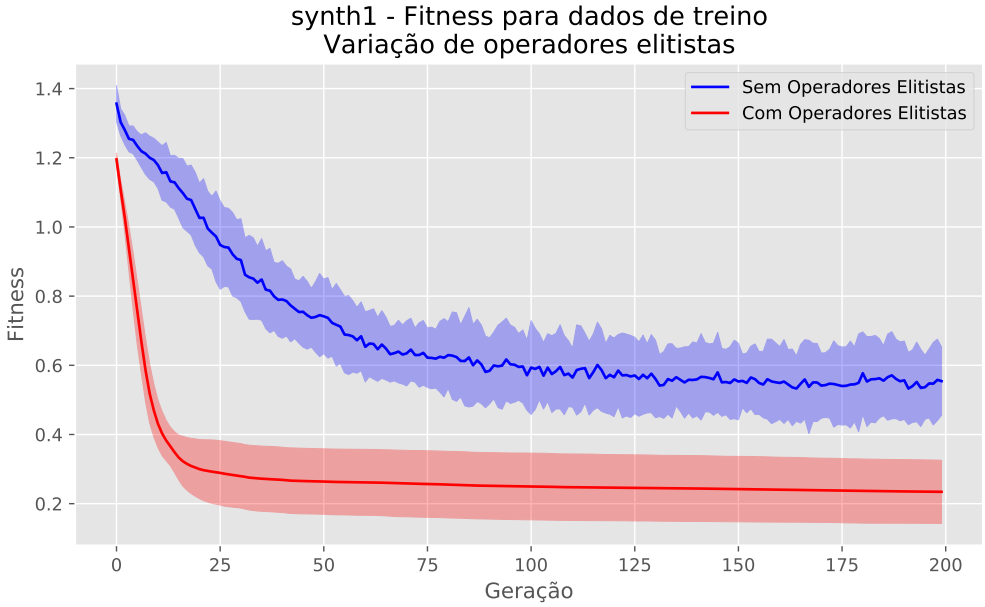


Figura 14. *synth1* variação da utilização de operador elitista

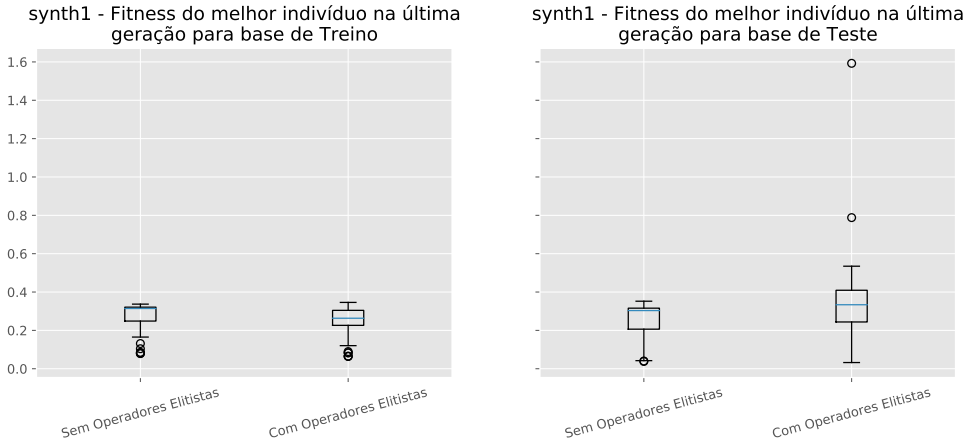


Figura 15. *synth1* variação da utilização de operador elitista, avaliação do último indivíduo

Entretanto, ao visualizar o efeito disso na base de testes na Figura 15, é possível identificar o *overfitting* acontecendo ao utilizar operadores elitistas. O erro na base de treino acaba chegando a níveis muito baixos, entretanto na base de testes há inúmeros *outliers* e a mediana é maior que no primeiro caso.

A indicação ao utilizar esses operadores é utiliza-los de uma forma mais inteligente. A escolha de apenas aceitar soluções melhores acaba diminuindo a diversidade da população drasticamente e acaba gerando funções extremamente específicas que passam pelos pontos de treino mas não conseguem generalizar para o *dataset* completo.

4.1.6. Melhor, indivíduo médio e pior indivíduo

Por fim, resta analisar o melhor e o indivíduo médio utilizando os melhores hiperparâmetros estudados nas seções anteriores. Os parâmetros são os seguintes: 200 gerações, $k = 2$, 200 indivíduos, *crossover* de 0.5 e mutação de 0.49.

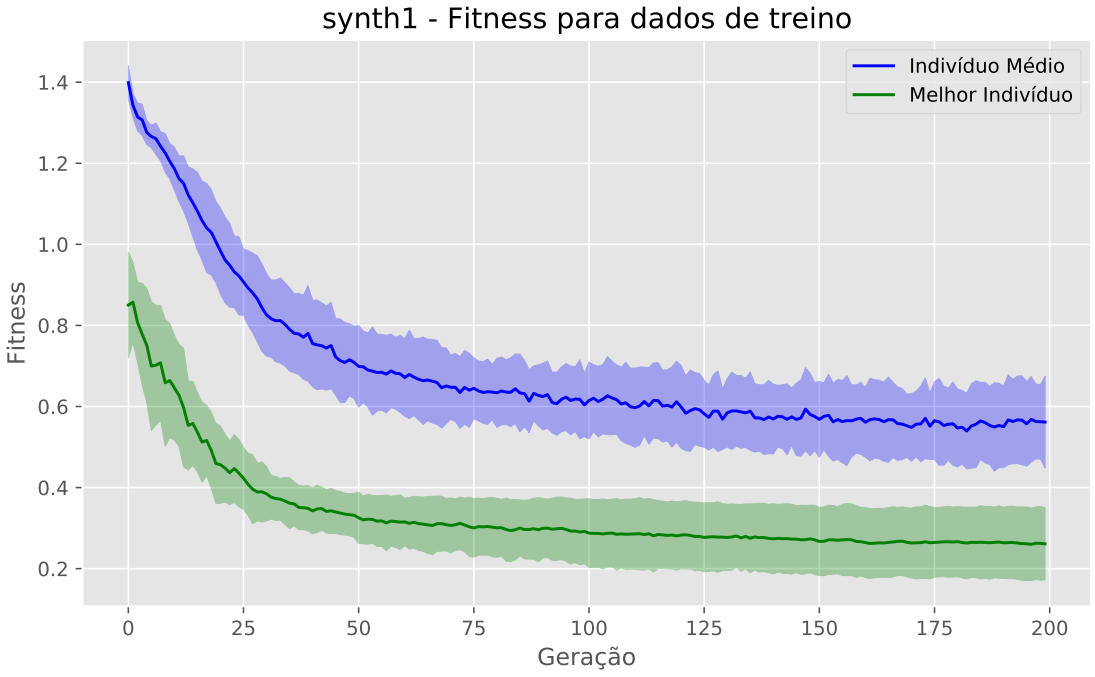


Figura 16. *synth1* indivíduo médio e o melhor indivíduo de cada geração

A Figura 16 mostra a média dos indivíduos e o melhor indivíduo de cada geração junto com o desvio padrão. O melhor indivíduo atinge, em média um valor de *fitness* de 0.28 com um desvio padrão próximo de 0.05. O indivíduo médio atingiu uma *fitness* um pouco a baixo de 0.6.

Como foi dito anteriormente, um dos problemas enfrentados no decorrer do projeto foi em que uma pequena parcela de indivíduos possuíam *fitness* extremamente ruins, seja por divisões em cadeia por números muito pequenos ou muita multiplicações por números/variáveis grandes. Para contornar esse problema, foram feitos duas coisas: arrendondar os números dos divisores na divisão e retirar indíviduos que tinham *fitness*

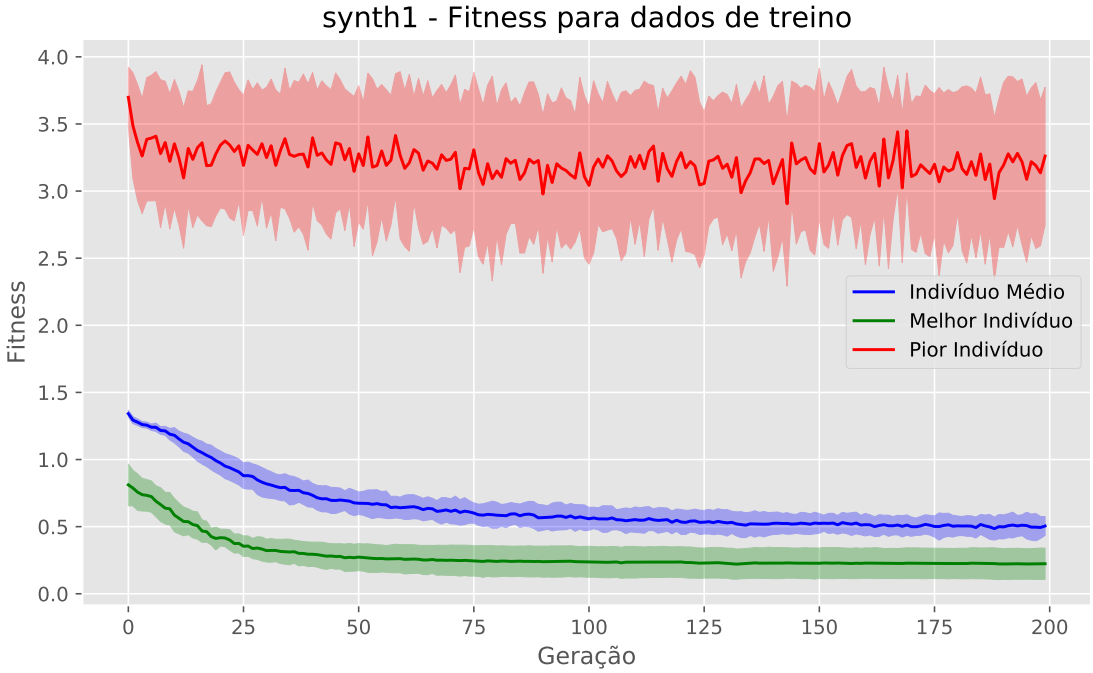
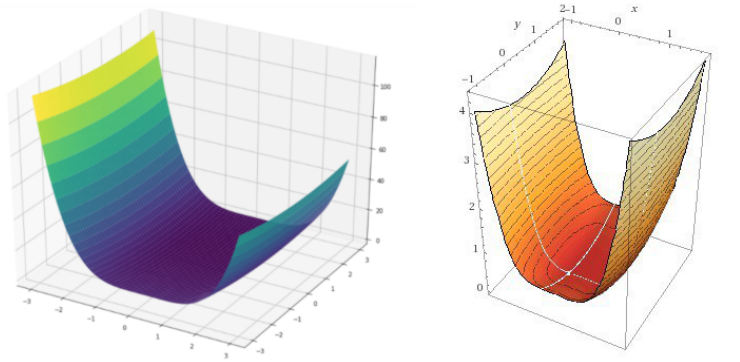


Figura 17. *synth1* indivíduo médio, melhor e pior indivíduo

maior que 4 da média (com eles, era impossível interpretar os gráficos). A Figura 17 ilustra um pouco o problema da interpretação com o pior indivíduos. Para os casos seguintes, o pior indivíduo não será considerado.



(a) Gráfico da Função Original

(b) Gráfico da Função Obtida

Figura 18. Comparação da forma das equações

O melhor indivíduo gerado, com uma *fitness* para a base de teste próximo a 0.02 foi o com o seguinte fenótipo:

```
((((X0 − 0.56) ∗ X0) ∗ ((X0 − 0.56) ∗ X0)) − (((1.0 ∗ (0.26
    + −0.05)) ∗ (((X1 − 0.7) − (X1 ∗ X1)) − −0.05)) + ((X0
  / ((0.13 + 0.85) − (X1 / 0.02))) ∗ (((X0 / X0) ∗ X1) /
  ((0.03 − X0) + (X0 / −0.03))))))
```

Que simplificando gera a Equação 2.

$$-\frac{xy}{(0.03 - 34.3333x) * (0.98 - 50y)} + (x - 0.56)^2 x^2 - 0.21(-y^2 + y - 0.65) \quad (2)$$

A Figura 18 mostra, na esquerda, o gráfico da função original e na direita o gráfico da função obtida. Embora as escalas estejam diferentes (a função original não foi fornecida para gerar os gráficos nas mesmas escalas), a forma da função se mantém parecida.

4.2. Teste no *dataset synth2*

Para o *dataset synth2*, será feito apenas um teste novo dos anteriores: o teste permitindo a adição da função seno. Quanto aos outros hiper-parâmetros será mantido a melhor combinação encontrada no cenário anterior, com exceção do número de gerações que foi colocada como 500 para ver seu comportamento de convergência.

4.2.1. Uso da função seno

A Figura 19 mostra a fitness média da população a cada geração com e sem a utilização da função seno. Inicialmente é possível identificar que a *fitness* média atingida nesse caso é maior do que no *dataset synth1*. Outro fator importante a ser observado é a de que a utilização da função seno, no geral, tem uma *fitness* com uma média menor do que quando não é utilizado a função.

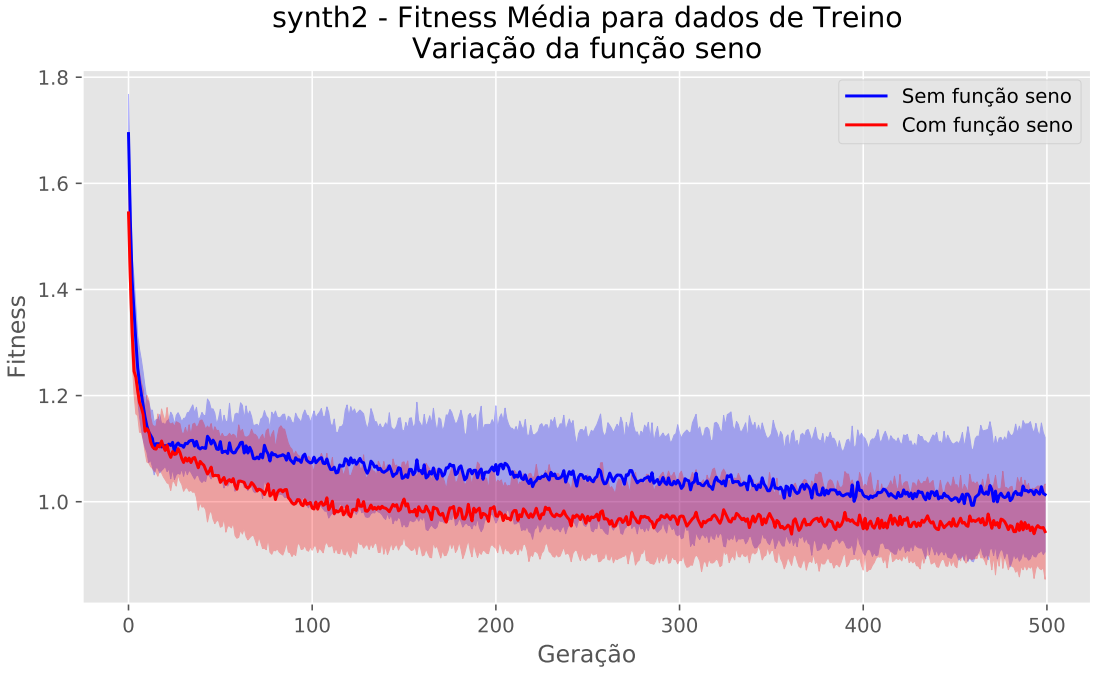


Figura 19. *synth2* variação do uso da função seno

Ao olhar para a Figura 20, nota-se um comportamento diferente do esperado da figura anterior. Aqui é possível notar que na verdade, a melhor solução é geralmente

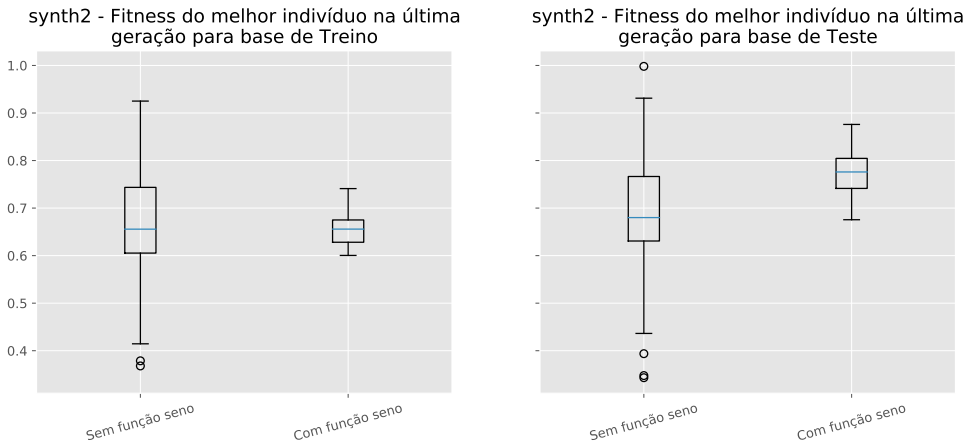


Figura 20. *synth2* variação do uso da função seno, avaliação do último indivíduo

obtida não utilizando a função seno. Entretanto, quando utiliza-se a função seno, o desvio padrão da solução obtida é significativamente menor. Caso deseja-se uma função com menor variância, a inclusão do seno parece ser uma boa alternativa. Faz-se necessário um estudo mais aprofundado desse *dataset*, já que pela forma da função fornecida em sala de aula, parecia que a função seguia esse comportamento, mas o algoritmo não conseguiu encontrar uma solução tão precisa quanto ao *dataset* anterior.

4.2.2. Melhor e indivíduo médio

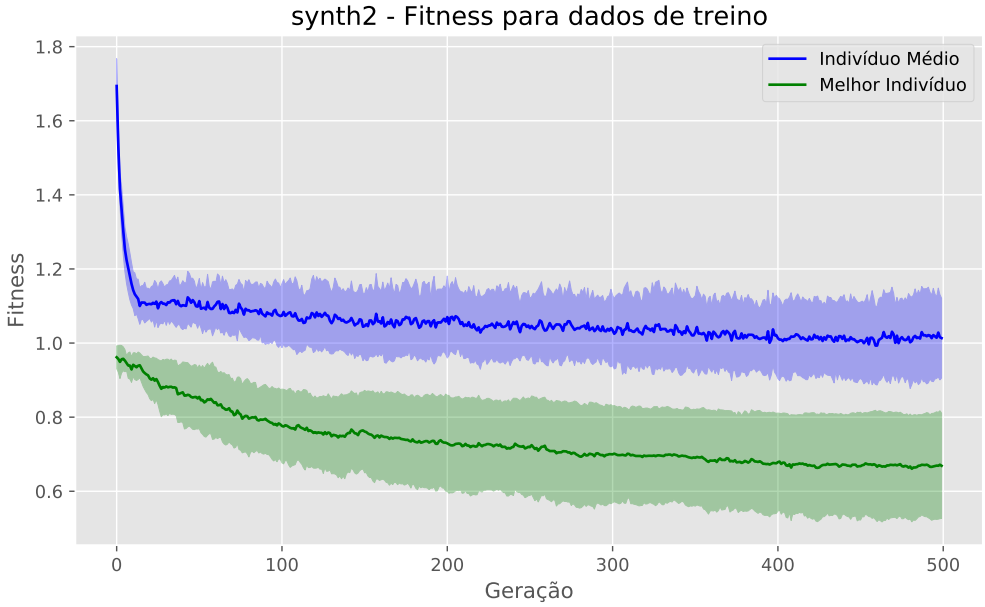


Figura 21. *synth2* indivíduo médio e o melhor indivíduo de cada geração

A Figura 21 mostra o comportamento do indivíduo médio da população e o melhor indivíduo. É importante notar aqui, que os indivíduos podem utilizar a função seno.

É interessante notar nesse caso que a convergência é extremamente rápida, precisando de apenas algumas gerações para atingir o valor mais baixo. As gerações subsequentes conseguem melhorar a *fitness* mas apenas um pouco. Talvez é necessário aumentar a diversidade dos indivíduos nesse caso para se obter melhores soluções.

O melhor indivíduo com *fitness* de 0.675 gerado foi o seguinte:

```
(−0.1 ∗ ((((((sin(X1)) − 1.82) − X0) + (X0 − X0)) − (((X0 −
    X1) ∗ (X1 − X0)) − (−1.82 + −1.82))) − ((((−1.82 − X1)
  + (X1 − (sin(X1)))) − (1.82 + X1)) ∗ (((X0 − 1.82) + (X0
    − X1)) ∗ (sin(X1))))))
```

4.3. Teste no *dataset concrete*

Por fim, no *dataset* concrete, algumas métricas foram revisitadas para serem estudadas. Foi escolhido a investigação da função do seno, junto com a probabilidade dos operadores e o tamanho do k. Os resultados estão dispostos nas próximas subseções.

4.3.1. Probabilidade dos operadores genéticos

Pela Figura 22, é possível observar que o comportamento da mudança das probabilidades dos operadores genéticos se mantém parecido com o que foi visto no *dataset synth1*, com a probablidade de cruzamento de 90% se mantem como a possivelmente melhor (embora não de para afirmar completamente devido a sobreposição dos intervalos).

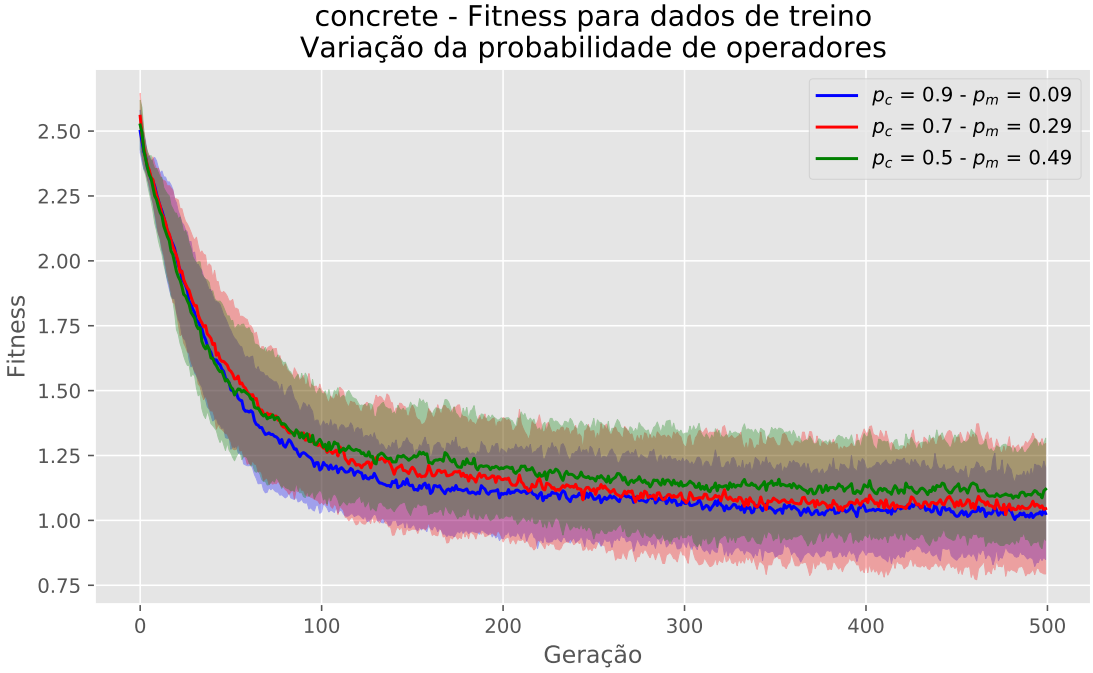


Figura 22. *concrete* variação do número de probabilidade dos operadores genéticos

Para a Figura 23, também é identificado o mesmo comportamento da análise da base de testes *synth1*. Entretanto, nessa base houveram alguns *outliers*, especialmente

para os casos de *crossover* de 70 e 50% (a outra probabilidade também teve mas foi menor). A melhor escolha para a base continua sendo com a probabilidade de *crossover* de 90%.

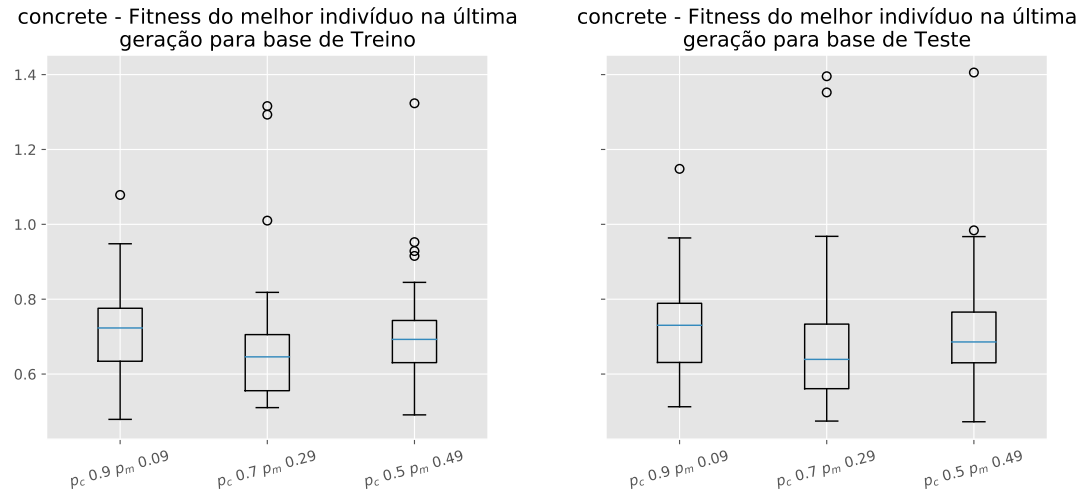


Figura 23. *concrete* variação do número de probabilidade dos operadores genéticos, avaliação do último indivíduo

4.3.2. Tamanho do torneio

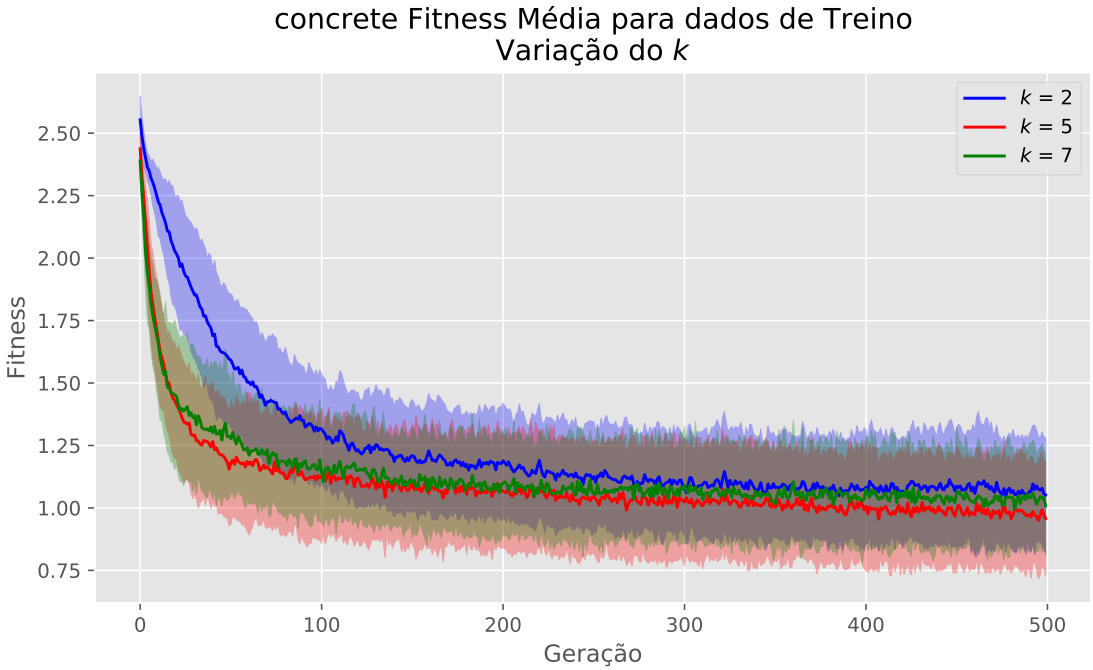


Figura 24. *concrete* variação do k no torneio

A Figura 24 mostra o comportamento que variação do tamanho do torneio tem sobre a solução.

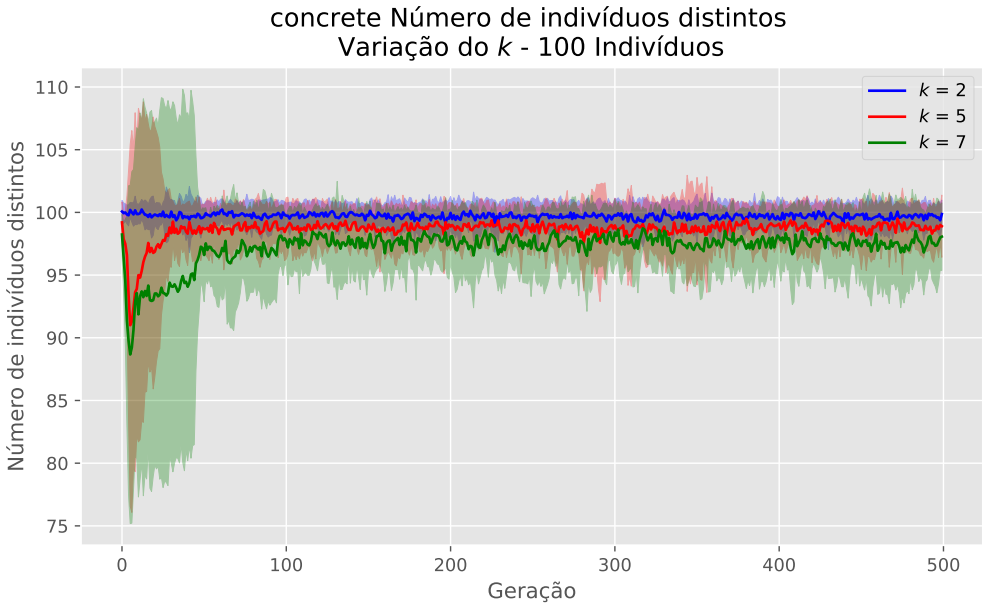


Figura 25. *concrete* média número de indivíduos distintos par cada k

Novamente, o comportamento é muito parecido com o que foi visto na base *synth1*: um k menor deixa os indivíduos explorarem mais devido a maior diversidade, enquanto que um k maior converge mais rápido. A verificação do último indivíduo neste caso não foi mostrada, também por seguir o comportamento já visto anteriormente.

A Figura 25 mostra quanto indivíduos distintos existem em cada geração. Novamente, o mesmo comportamento seguiu, mas dessa vez quase todos os indivíduos foram distintos para essa base com o $k = 2$. Para os outros valores, também houve um aumento no número de indivíduos em relação a outra base.

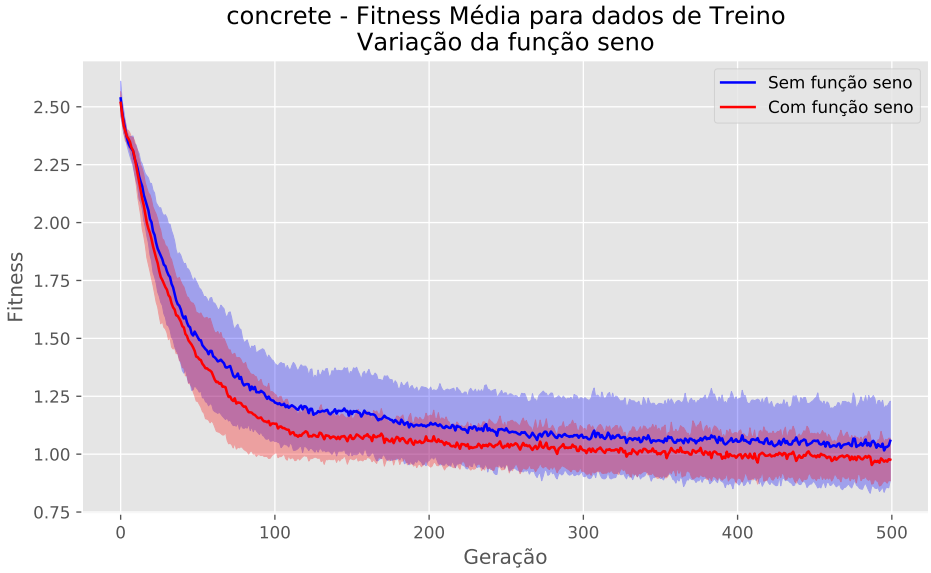


Figura 26. *concrete* variação do uso da função seno

4.3.3. Uso da função seno

A Figura 26 mostra que, aparentemente, a função seno ajuda a obter uma *fitness* média melhor. Não há muitas conclusões além dessa que podem ser tiradas já que os intervalos do desvio padrão se sobrepõem.

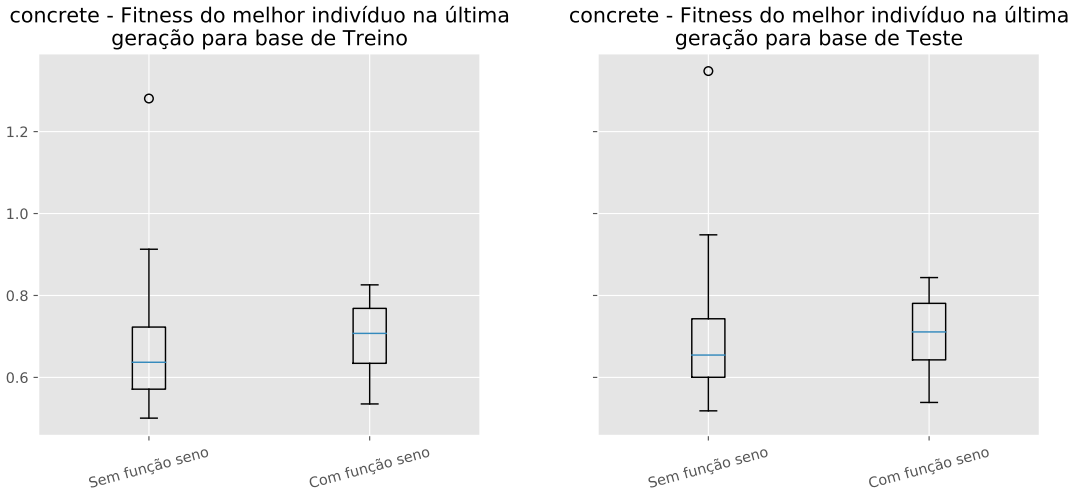


Figura 27. *concrete* variação do uso da função seno, avaliação do último indivíduo

Para a Figura 27, é possível notar que a função de seno realmente ajudou a melhorar a solução para a base de testes. Embora a mediana da solução esteja um pouco cima da função sem o seno, o desvio padrão é menor e não apresentou *outliers*.

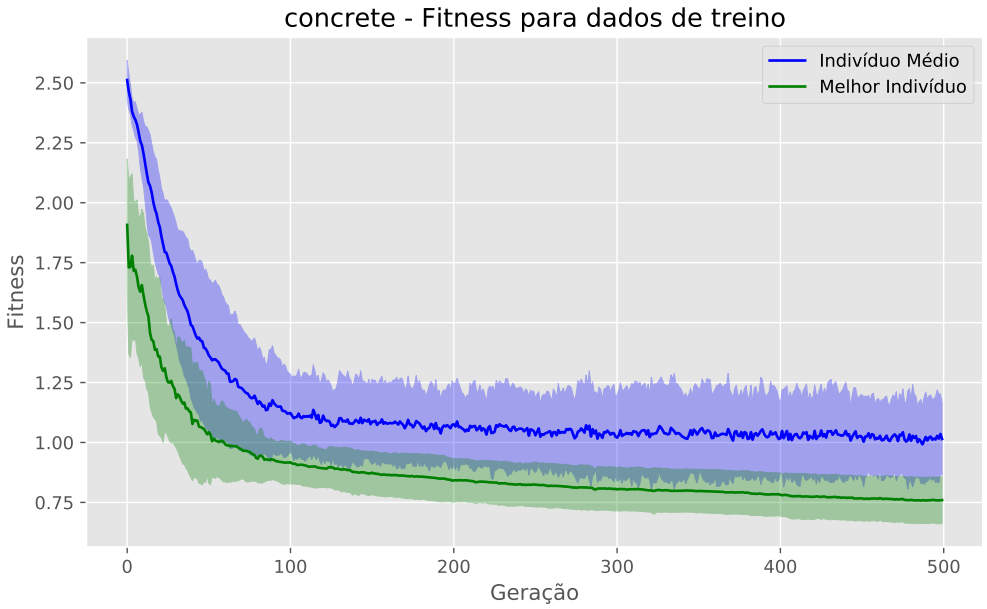


Figura 28. *concrete* indivíduo médio e o melhor indivíduo de cada geração

4.3.4. Melhor e indivíduo médio

A Figura 28 mostra o indivíduo médio e o melhor indivíduo para cada geração. A *fitness* média atingia é em torno de 1, enquanto que a melhor se aproxima de 0.75. Os hiperparâmetros para esse caso foram: 500 gerações, $k = 2$, 100 indivíduos, *crossover* de 90%, mutação de 9%, reprodução de 1%.

O melhor indivíduo com uma *fitness* de 0.65, foi dado pela equação:

```
(((((0.06 / X7) + ((X0 + X0) - ((sin(X7)) * X2))) + (X7 -
    ((X1 + X7) / -1.36))) * 0.06) - (((X7 - 0.06) / (((0.06
    / 0.06) * -1.36) * ((X0 * X4) * (X7 + X2)))) / (((0.06 -
     (-1.36 * X4)) / X4) + (0.06 - ((X7 - X4) / (X7 + X2))))
    ))
```

5. Conclusão

A regressão simbólico se mostrou um interessante método a ser considerado, especialmente para a base sintética 1, onde a *fitness* atingiu um valor próximo a 0.02. Para os outros casos, a *fitness* não se comportou tão bem assim, mas ainda resta uma extensa análise dos hiperparâmetros e indivíduos para assim tentar conseguir um melhor resultado.

A tarefa da busca e análise de hiper-parâmetros ideais foi a mais difícil do trabalho, exigindo longos testes para poder tirar conclusões sobre os resultados. Ao procurar na literatura, é possível identificar que essa área ainda é um grande foco de pesquisa, assim como na maioria das subáreas de inteligência artificial.

Alguns pontos do trabalho ainda precisam ser explorados, como por exemplo, a geração de indivíduos que "explodiam" quando ocorria subsequentes divisões por números muito pequenos, o que atrapalhou a análise dos piores indivíduos e também dificultou a análise da média dos indivíduos. Esse ponto ainda deve ser explorado em trabalhos futuros.

Uma questão que não foi muito analisada nos indivíduos é a questão do *bloating*, que ocorre quando certas regiões da árvore não possuem efeito nenhum sobre a solução (como é o caso de uma multiplicação por 0, por exemplo). Ha muita discussão sobre a vantagem ou desvantagem desses nós. Alguns autores argumentam que estas regiões "inuteis" servem para proteger de *crossovers* drásticos, enquanto outros acreditam que remove-los é o melhor caminho [Luke 2000, Poli 2003]. Nos indivíduos aqui estudados foi identificado a questão do *bloating*. Embora não seja difícil realizar a verificação desses nós, ela se torna computacionalmente cara (especialmente para indivíduos com árvores maiores do que as aqui utilizadas), já que para cada sub-arvore deveria executar a avaliação de um conjunto de entradas e verificar se o retorno daquela subárvore é 0. Com muitos indivíduos, isso se torna custoso e não foi extensivamente analisado aqui.

Ainda restam inúmeras questões a serem analisadas para trabalhos futuros, dentre as principais sugestões estão:

- Refatoração do código para eliminar buscas na árvore que já haviam sido calculadas (busca de nós da árvore);
- Paralelização da avaliação de *fitness*;

- Estudo aprofundado do *bloating* nos indivíduos e verificar se remover sub-árvores inúteis geram melhores resultados;
- Estudo de outras funções, como log, exponenciação, etc;
- Estudo em outros *datasets*

Referências

Altenberg, L. (1994). Advances in genetic programming. chapter The Evolution of Evolvability in Genetic Programming, pages 47–74. MIT Press, Cambridge, MA, USA.

Augusto, D. A. and Barbosa, H. J. (2000). Symbolic regression via genetic programming. In *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*, pages 173–178. IEEE.

Hornby, G., Globus, A., Linden, D., and Lohn, J. (2006). Automated antenna design with evolutionary algorithms. In *Space 2006*, page 7242.

Luke, S. (2000). *Issues in scaling genetic programming: breeding strategies, tree generation, and code bloat*. PhD thesis, research directed by Dept. of Computer Science.University of Maryland, College Park.

Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In *European Conference on Genetic Programming*, pages 204–217. Springer.

Searson, D. P., Leahy, D. E., and Willis, M. J. (2010). Gptips: an open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International multiconference of engineers and computer scientists*, volume 1, pages 77–80. Citeseer.

Stork, D. G. (2015). Machine learning for discovering formulas. Cross Validated. URL:https://stats.stackexchange.com/q/140410 (version: 2015-03-05).

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., and Galván-López, E. (2011). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119.

Walker, M. (2001). Introduction to genetic programming.

Yurovitsky, M. (1995). Playing tetris using genetic programming. *Genetic Algorithms and Genetic Programming at Stanford*, pages 309–319.