Trabalho Prático 1
Programação Genética

Bernardo de Almeida Abreu[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Disciplina: DCC191 - Computação Natural 2017/2

bernardoabreu@dcc.ufmg.br

1. Introdução

Um algoritmo de programação genética é um algoritmo evolutivo, classe de algoritmos que, através de um processo probabilístico, transforma populações de indivíduos em, possivelmente, uma população melhor. [Poli et al. 2008] Os indivíduos que, ao longo da execução do algoritmo, sofrerão adaptações, são comumente programas de computador hierárquicos de tamanho e forma dinamicamente variáveis. [Koza 1992]

Nesse trabalho, a programação genética é utilizada para realizar uma regressão simbólica a partir de alguns conjuntos de dados, de forma a encontrar funções que aproximam os valores desses conjuntos. Ou seja, dado um conjunto de m amostras provenientes de uma função desconhecida $f : \mathbb{R}^n \mapsto \mathbb{R}$, representadas por uma dupla $\langle X, Y \rangle$ 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 ajusta as amostras fornecidas.

Uma regressão simbólica consiste em encontrar uma expressão matemática, em forma simbólica, que provê um bom encaixe entre uma amostra finita de valores de variáveis independentes e os valores associados de variáveis dependentes. Esse encaixe entre os valores pode, além de ser bom, ser o melhor possível ou mesmo perfeito. Portanto, a regressão simbólica envolve encontrar o modelo que corresponde a uma dada amostra de dados. A expressão matemática sendo buscada pode ser vista como um programa de computador que recebe os valores das variáveis independentes como entrada e produz os valores das variáveis dependentes como saída. [Koza 1992]

Foram utilizados três bases de dados para realizar os trabalhos, em que duas delas são bases sintéticas, representando uma função conhecida, enquanto a outra representa uma base real.

2. Implementação

2.1. Individuo

Um indivíduo nesse trabalho representa uma expressão matemática válida que é uma solução para a regressão simbólica. Essas funções são representadas por meio de uma árvore, na qual as folhas representam constantes e variáveis da função e os nós restantes representam funções. A figura 1 apresenta uma árvore para a expressão $X + 1$. Operações mais complexas também podem ser representadas.

As operações matemáticas suportadas são:

- Adição

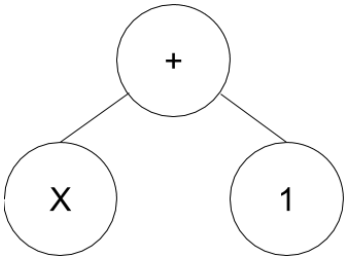

Figure 1. Árvore representativa da expressão $X + 1$.

- Subtração
- Multiplicação
- Divisão
- Raiz quadrada
- Potenciação
- Seno
- Cosseno

Cada nó da árvore foi implementado de forma que o mesmo possua dois filhos, que assumem um valor nulo quando não existem, e o conteúdo do nó, que pode ser um terminal (variável ou constante) ou uma função. Como cada nó é na realidade um nó raiz de uma sub-árvore, ele também guarda a altura da sub-árvore e uma *string* que representa a equação matemática da mesma.

Um indivíduo é implementado como um objeto que possui uma árvore e um valor de *fitness*.

2.2. Avaliação do Indivíduo

Um indivíduo é avaliado utilizando a raiz quadrada do erro quadrático médio (RMSE), mostrado na equação 1.

$$f(Ind) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Eval(Ind, x) - y)^2} \tag{1}$$

Ind é o indivíduo sendo avaliado pela função Eval(*Ind*, *x*) no conjunto de entrada fornecido *x*, *y* e a saída correta da função para a entrada *x*, e *N* é o numero de exemplos fornecidos.

Para calcular a *fitness* de um indivíduo, deve-se obter o valor da expressão da árvore para os dados de entrada da base de dados, o que é representado na equação 1 pela função *Eval*. Para isso caminha-se na árvore em pós ordem, retornando o valor de cada sub-árvore, de modo que ao retornar o valor da raiz, obtenha-se o valor da equação. O valor de cada nó é dado da seguinte forma:

Terminal Caso seja uma constante, o valor da mesma, caso contrário o valor da variável dado pelo conjunto de dados da entrada.

Função O resultado da operação com os filhos como argumentos para a mesma.

Quando alguma operação é inválida, como raiz quadrada de um número negativo ou divisão por 0, é retornado o valor 0. Indivíduos cuja árvore possui uma altura maior do que 7 são penalizados adicionando uma valor de 10.000.000, atingindo até no máximo o valor máximo para inteiros conforme determinado pela linguagem *Python*, o que equivale a 9.223.372.036.854.775.807. Isso é feito para tentar evitar indivíduos não cresçam demasiadamente, causando *bloating*. Além disso, quando quando algum erro ocorre no cálculo da *fitness*, como um *overflow* ou alguma outra exceção, a *fitness* do indivíduo também é definida como máximo valor inteiro. Essa decisão foi tomada para tratar os erros que podem ocorrer caso a linguagem não consiga calcular o resultado de alguma operação muito grande, como resultados de potenciação, ou caso o limite de recursão seja atingido em algum momento.

2.3. Operadores Genéticos

Os operadores genéticos escolhidos foram Cruzamento, Mutação e Reprodução. Esses operadores são aplicados de forma paralela e não sequencial, ou seja, a soma das probabilidades de todos eles deve ser 100% e só um será aplicado a cada seleção.

2.3.1. Cruzamento

Para selecionar o nós a serem trocados de cada indivíduo é criada uma lista para cada árvore com os elementos da mesma. Um nó é então selecionado com probabilidade igual para funções e terminais. Uma árvore é então percorrida para encontrar esse nó, que é então substituído pelo nó selecionado do outro indivíduo. O cruzamento retorna os dois filhos gerados por ele.

2.3.2. Mutação

A mutação realizada é a mutação de ponto. Um nó do indivíduo pai é selecionado da mesma forma que no cruzamento, é feita uma lista com os nós da árvore e um deles é selecionado com probabilidade igual para todos. Então é selecionado um terminal ou função, que não seja igual ao elemento do nó, para substitui-lo. Se o elemento escolhido for do mesmo tipo que o já presente, ou seja, os dois são funções ou terminais, então o novo elemento substitui o antigo conteúdo do nó. Caso um função substitua outra com aridade diferente será necessário eliminar uma das sub-árvores filhas ou então criar uma nova, dependendo da aridade de nó que substitui e do que é substituído.

Caso um função seja escolhida para substituir um terminal, ou vice-versa, um nó não pode ter seu conteúdo substituído, mas sim o próprio nó deve ser substituído. Dessa forma, ao substituir um terminal por uma função, novos filhos devem ser criados, e ao substituir uma função por um terminal, os filhos devem ser removidos.

Novos nós que devem ser criados, tanto para substituições de funções de aridades diferentes, quanto para substituições de terminais por funções, possuem profundidade máxima de 1 e são criados com o método *grow*.

2.4. Decisões Adicionais

A população inicial para o algoritmo é criada utilizando o método *ramped half-and-half*.

As estatísticas podem ser salvas em arquivos de texto e são calculadas toda vez que um novo filho é criado. A população é ordenada pela *fitness* a cada geração, de modo a se pegar a melhor *fitness*, a pior *fitness* e a mediana. Além disso, ao manter a população ordenada é possível pegar *n* elementos, de acordo com o elitismo definido, para a próxima geração.

3. Execução

O programa do trabalho foi implementado usando a linguagem Python, para executar o programa é necessário utilizar *Python 2* ou *Python 3*. O nome do arquivo principal é *main.py*

Um exemplo de comando de execução está a seguir.

./main.py --train keijzer-7-train.csv --**test** keijzer-7-**test**.csv

Os parâmetros *–train* e *–test* são obrigatórios e correspondem aos arquivos contendo a base de dados de treino e teste, respectivamente. Exitem outros argumentos que podem ser utilizados para executar o programa, porém todos aqueles exceto *train* e *test* são opcionais.

train Arquivo de base de dados de treino.
test Arquivo de base de dados de teste.
crossover Probabilidade de cruzamento.
mutation Probabilidade de mutação.
elitism Número de indivíduos a serem usados no elitismo. Caso o número seja 0, não há elitismo.
pop_size Tamanho da população.
gen Número de gerações.
tournament Número de indivíduos a serem usados no torneio.
seed Semente para geração de números aleatórios.
stats Arquivo para salvar as estatísticas.
test_out Arquivo para salvar o resultado da base de dados de teste.

Quando se escolher um nome de arquivo para salvar as estatísticas, é criado um arquivo para cada estatística a ser salva. Caso o arquivo já exista, os novos dados serão inseridos ao final do arquivo, sem apagá-lo. Para cada métrica é inserida uma única linha, com os valores das métricas para cada geração, separadas por vírgulas. O arquivo para o resultado da base de dados de teste funciona da mesma maneira.

4. Experimentos

O experimentos foram realizados em um computador com as seguintes configurações:

Processador: Intel Core i7 4ª geração.
Memória: 16GB DDR3 166MHz
Sistema Operacional: Ubuntu 16.04

4.1. Metodologia

Foram realizados diversos experimentos de forma a encontrar os melhores parâmetros para a execução do programa. Para cada teste, foi definido um parâmetro a ser variado e todos os outros foram fixados em um valor. Para cada variação desse parâmetro, o programa foi executado 30 vezes, cada uma com uma *seed* diferente para a geração dos números aleatórios. Ao término de um teste, um valor de parâmetro foi definido e passou a ser o padrão nos testes seguintes. A *seed* usada para cada execução foi o número entre 1 e 30 relativo aquela execução. Os gráficos são construídos a partir da média dos valores da métrica analisada para as 30 *seeds*.

4.2. Testes

4.2.1. Variação da população

O objetivo do primeiro teste realizado foi analisar as implicações derivadas das mudanças no tamanho da população e, dessa forma, definir um valor para o parâmetro. Os parâmetros de execução foram os seguintes:

Número de Gerações 50
Probabilidade de Cruzamento: 0.9
Probabilidade de Mutação: 0.05
Tamanho do Torneio: 2
Tamanho do Elitismo: 1

Os valores testados para o tamanho da população inicial foram 50, 100 e 500.

É possível observar nas figuras 2 e 3 que com uma maior população, o programa converge mais rapidamente. Uma vez que a melhor solução, embora não seja a solução perfeita, é encontrada muito rapidamente, não seria necessário aumentar o tamanho da população. Porém, ao analisar a figura 4, é possível observar que um maior tamanho de população contribui positivamente e significativamente para uma melhora da melhor *fitness*. Portanto, o tamanho da população escolhido para testes futuros foi 500.

4.2.2. Variação do número de gerações

O objetivo desse teste realizado foi analisar as implicações derivadas das mudanças no número de gerações. Os parâmetros de execução foram os seguintes:

Tamanho do População Inicial: 500
Probabilidade de Cruzamento: 0.9
Probabilidade de Mutação: 0.05
Tamanho do Torneio: 2
Tamanho do Elitismo: 1

As figuras 5 e 6 mostram que com um número maior de gerações a melhor *fitness* para *keijzer-7* e *keijzer-10* apresentam uma melhora, porém não significativa. A figura 7 também mostra que, a partir de 100 gerações, ocorre uma melhora na melhor *fitness*. Essa melhora, porém é muito pequena, não sendo significativa o suficiente para justificar um número maior de gerações. O número de gerações escolhidas para execuções futuras foi de 100 gerações.

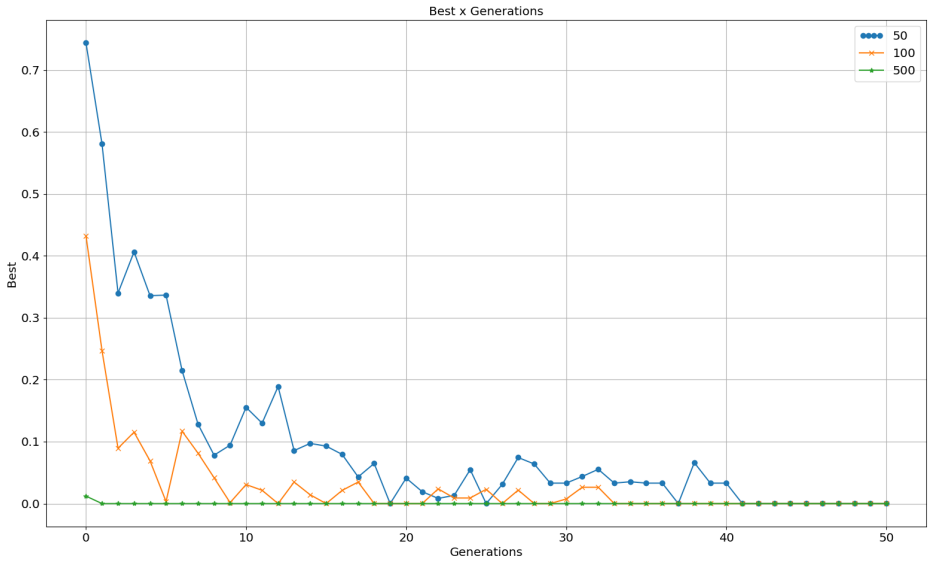

Figure 2. Melhor *fitness* para *keijzer-7* variando tamanho da população.

4.2.3. Variação das probabilidades de cruzamento e mutação

O objetivo do terceiro teste realizado foi analisar as implicações derivadas do aumento da probabilidade de mutação e diminuição da probabilidade de cruzamento. Os parâmetros de execução foram os seguintes:

Tamanho do População Inicial: 500
Número de Gerações 100
Tamanho do Torneio: 2
Tamanho do Elitismo: 1

Os valores testados foram probabilidade de 0.9 de cruzamento e e 0.05 de mutação e, em seguida, probabilidade de 0.6 de cruzamento e e 0.3 de mutação.

É possível observar nas figuras 8, 9 e 10 que os valores de pior *fitness* e *fitness* média pioram para uma probabilidade de mutação de 0.3 e de cruzamento de 0.6. Os valores de melhor *fitness* por sua vez não apresentam uma grande melhora.

Um maior valor de mutação também influenciou no números de indivíduos repetidos na população, diminuindo esse valor, como se pode ver na figura 11. Para se decidir se um indivíduo está repetido ou não, leva-se em conta somente seu genótipo, e não seu fenótipo. Isso pode levar a casos em que dois indivíduos, apesar de parecerem diferentes, são iguais.

Após feitas as análises, foram mantidas as probabilidades de 0.9 para cruzamento e 0.05 para mutação para execuções futuras.

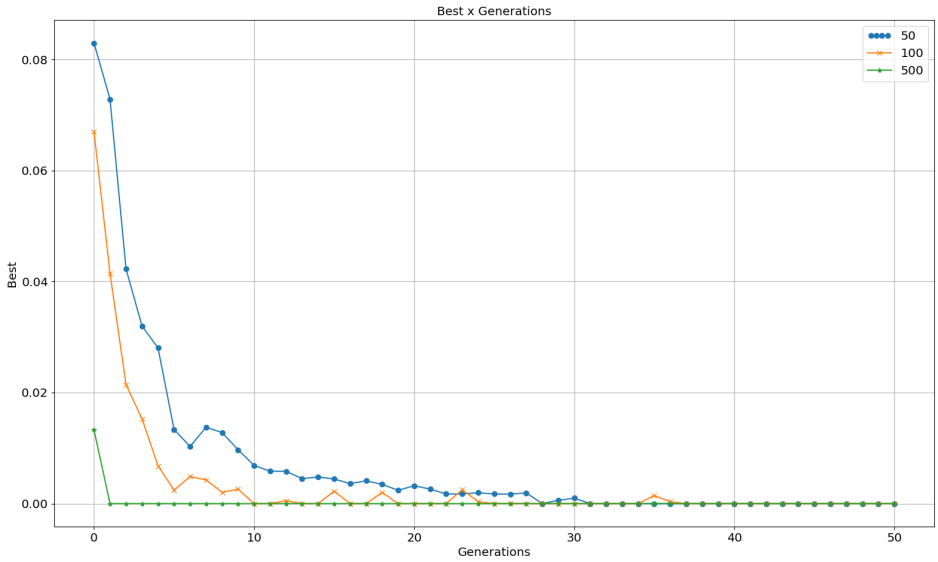

Figure 3. Melhor *fitness* para *keijzer-10* variando tamanho da população.

4.2.4. Variação do tamanho do torneio

A partir desse teste, o tamanho do torneio foi variado. Os parâmetros de execução foram os seguintes:

Tamanho do População Inicial: 500
Número de Gerações 100
Probabilidade de Cruzamento: 0.9
Probabilidade de Mutação: 0.05
Tamanho do Elitismo: 1

Os valores testados para o tamanho torneio foram 3 e 7.

A influencia de diferentes valores de torneio no conjuntos de dados *keijzer-7* e *keijzer-10* não é grande, como é mostrado nas figuras 12 e 13. A figura 14 mostra que, com um valor de torneio de 7, a média da *fitness* piora em relação ao torneio de 3 após a geração 40, aproximadamente. Também é possível ver o impacto da pressão seletiva e do maio valor de torneio, que fazem com que o número de indivíduos repetidos na população aumente, como é visto na figura 15.

Ao se observar a figura 16 porém, nota-se que com um tamanho de torneio igual a 7 a melhor *fitness* do conjunto de dados de *house* melhora. Dessa forma, definiu-se o valor do torneio como 7. A figura 17 que a variação da melhor *fitness* para diferentes *seeds* é grande nas primeiras gerações, mas converge em torno de um mesmo valor nas gerações finais.

Ao comparar os resultados obtidos com os dados de treino com os resultados obtidos com os dados de teste para um torneio de 7 na figura 18, observa-se que não ocorre *overfitting* e que os resultados dos testes acompanham os resultados do treino.

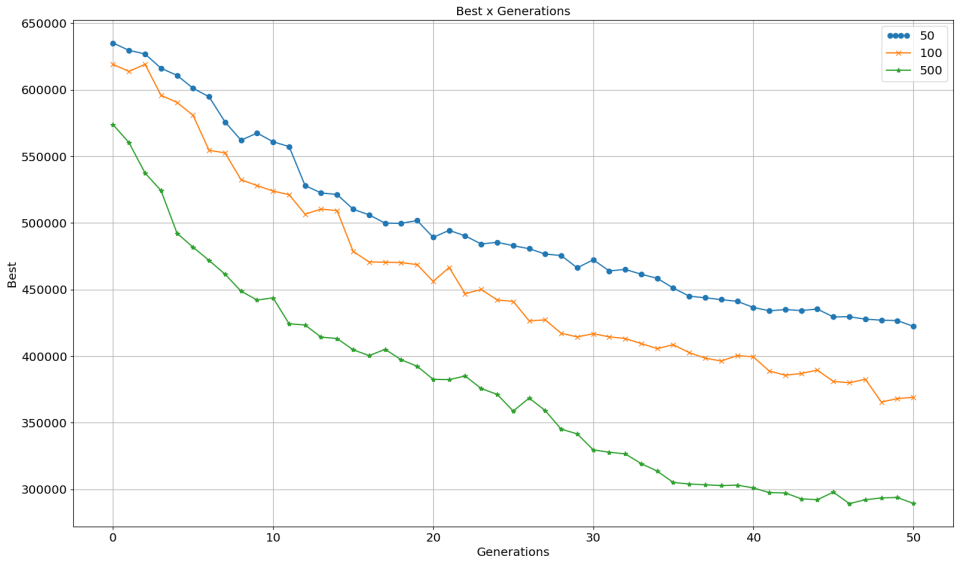

Figure 4. Melhor *fitness* para *house* variando tamanho da população.

4.2.5. Remoção do Elitismo

No teste final, foi removido o elitismo. Os parâmetros de execução foram os seguintes:

Tamanho do População Inicial: 500
Número de Gerações 100
Probabilidade de Cruzamento: 0.9
Probabilidade de Mutação: 0.05
Tamanho do Torneio: 7
Tamanho do Elitismo: 0

Ao se remover o elitismo, é possível observar uma leve diminuição na melhor *fitness* de *house*, como se observa na figura 19. Além disso, a figura 20 mostra que a variação dos valores de melhor *fitness* ao final das gerações é maior do que quando se tem elitismo.

5. Conclusão

A escolha correta dos parâmetros para o algoritmo influenciam drasticamente em seu desempenho. A tarefa de realizar diferentes experimentos e assim determinar os melhores parâmetros é árdua e não-trivial. Muitas vezes não se está claro qual é realmente o parâmetro mais adequado e, as escolhas do mesmo afetam os resultados posteriores do programa.

Os conjuntos de dados artificiais, *keijzer-7* e *keijzer-10* possuíam respostas mais simples, o que fez com que sua evolução não fosse tão notável. A resposta para esses problemas convergia muito rapidamente, já com o primeiro ajuste nos parâmetros. Dessa forma, ajustes posteriores não faziam muito efeito na execução do problema, que constantemente encontrava um resultado muito bom.

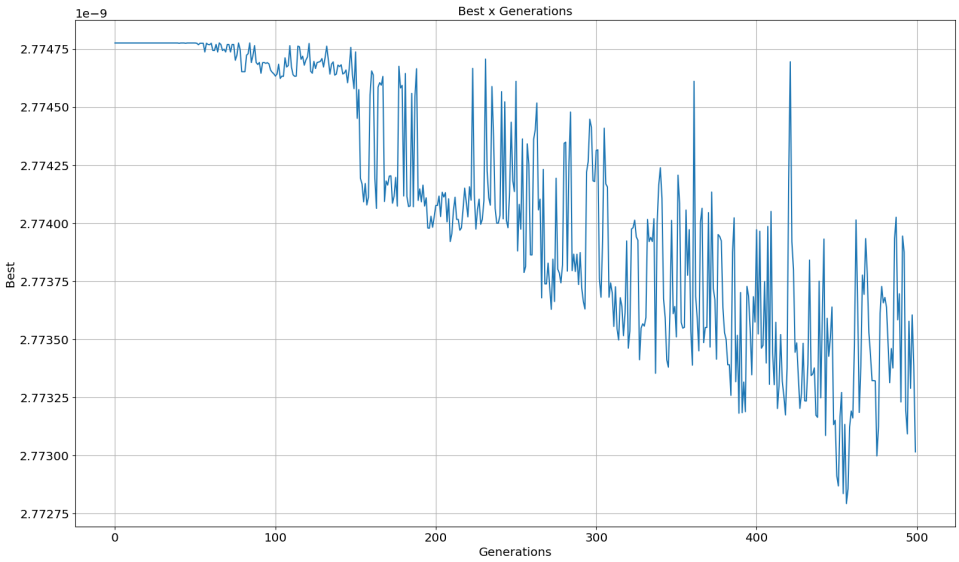

Figure 5. Melhor *fitness* para *keijzer-7*.

O conjunto de dados real, *house*, não atingiu um resultado perfeito. A melhor *fitness* encontrada para o mesmo ficou em torno de 230.000, resultado muito diferente dos encontrados para os outros conjuntos de dados. A evolução do algoritmo para esse conjunto de dados porém, foi notável. A cada novo teste foi possível ver a melhor *fitness* alcançada melhorar, de forma que ficou evidente a importância de uma boa escolha de parâmetros.

6. Bibliografia

References

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.

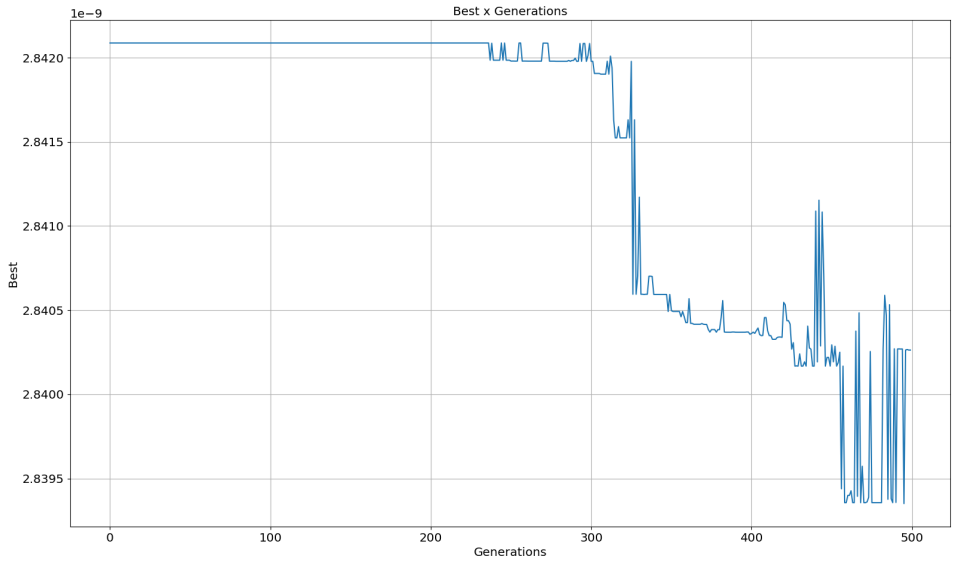

Figure 6. Melhor *fitness* para *keijzer-10*.

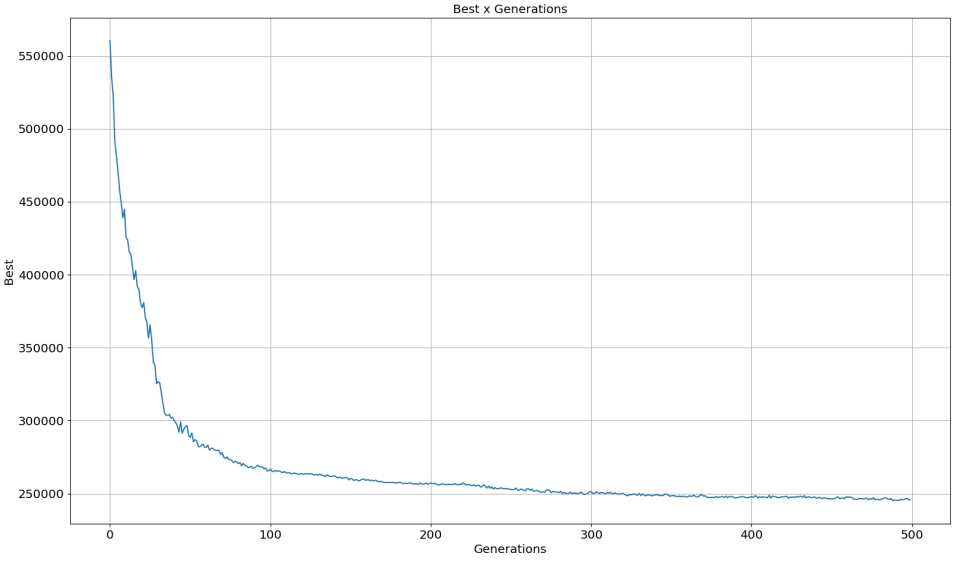

Figure 7. Melhor *fitness* para *house*.

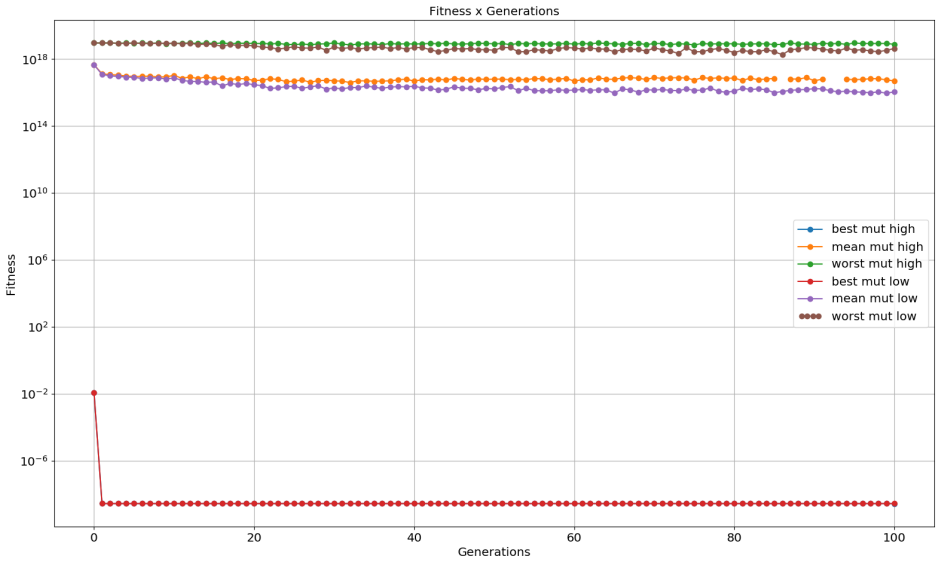

Figure 8. Comparação de diferentes valores de probabilidade para operadores genéticos na *fitness* de *keijzer-7*.

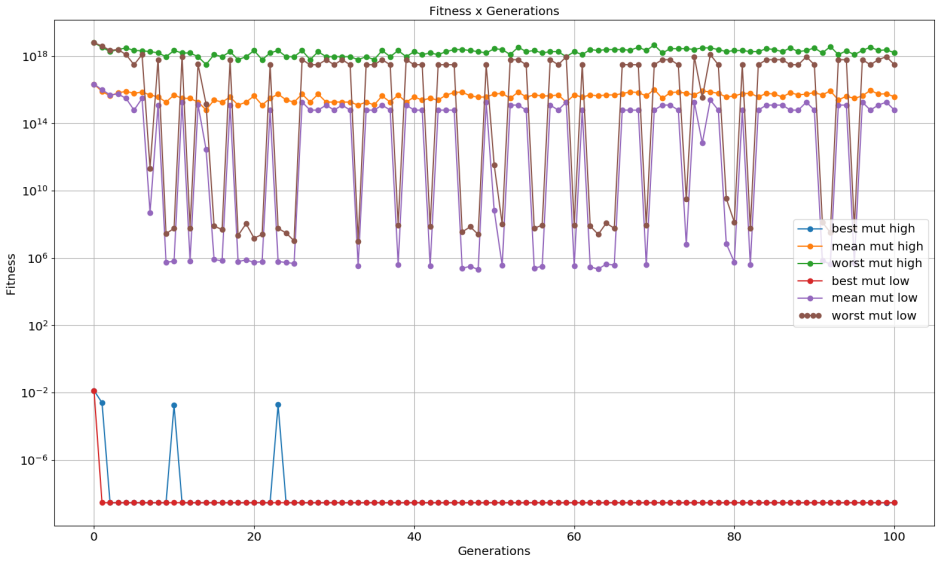

Figure 9. Comparação de diferentes valores de probabilidade para operadores genéticos na *fitness* de *keijzer-10*.

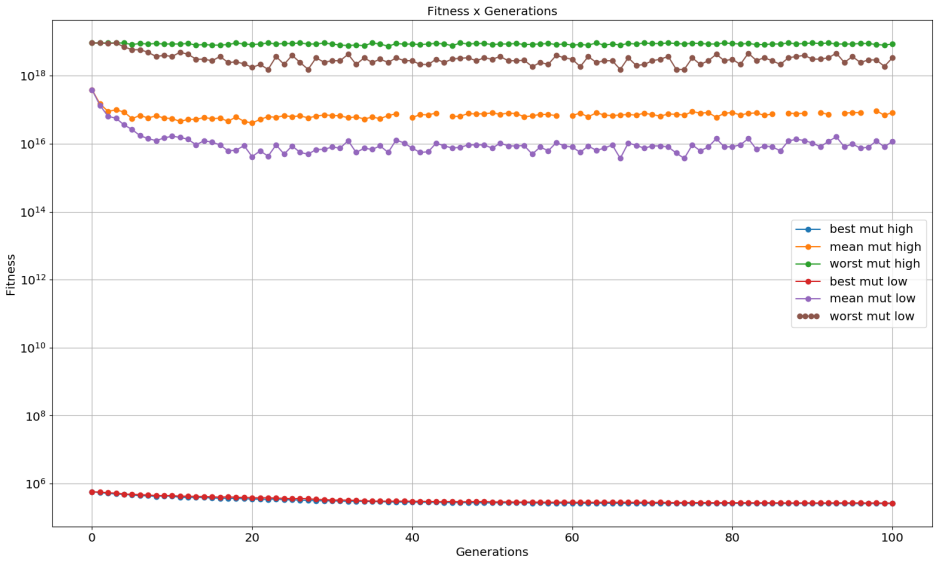

Figure 10. Comparação de diferentes valores de probabilidade para operadores genéticos na *fitness* de *house*.

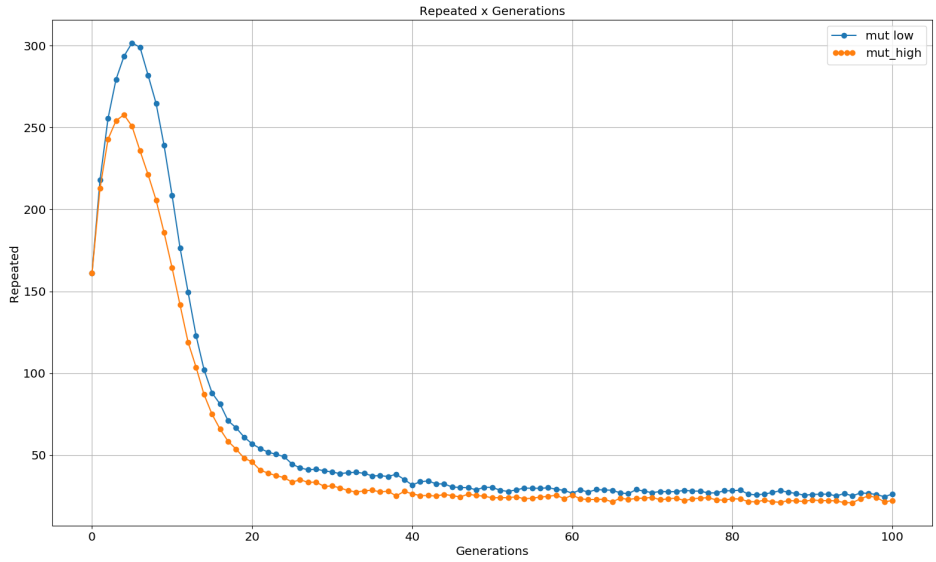

Figure 11. Indivíduos repetidos para *house* com diferentes probabilidades para operadores genéticos.

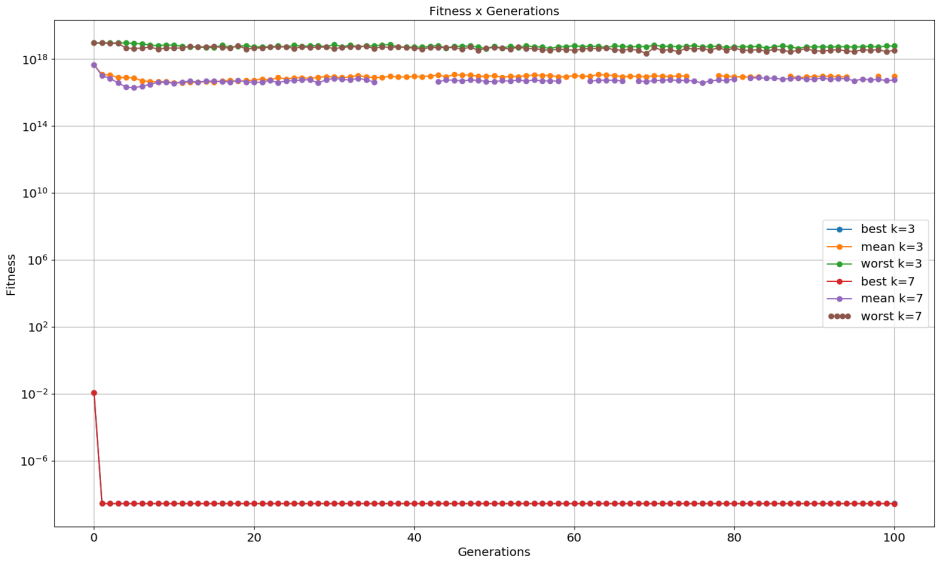

Figure 12. Comparação de diferentes valores de torneio na *fitness* de *keijzer-7*.

Figure 13. Comparação de diferentes valores de torneio na *fitness* de *keijzer-10*.

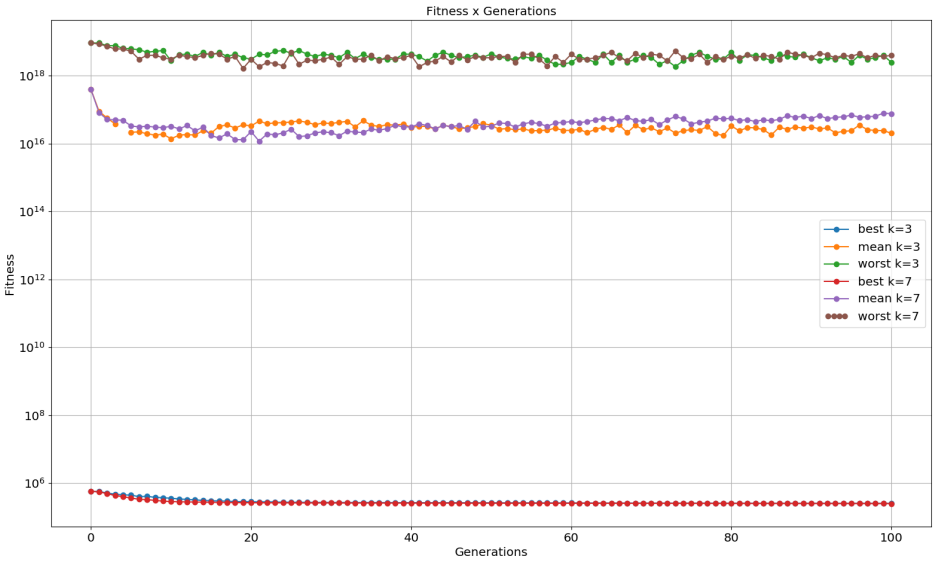

Figure 14. Comparação de diferentes valores de torneio na *fitness* de *house*.

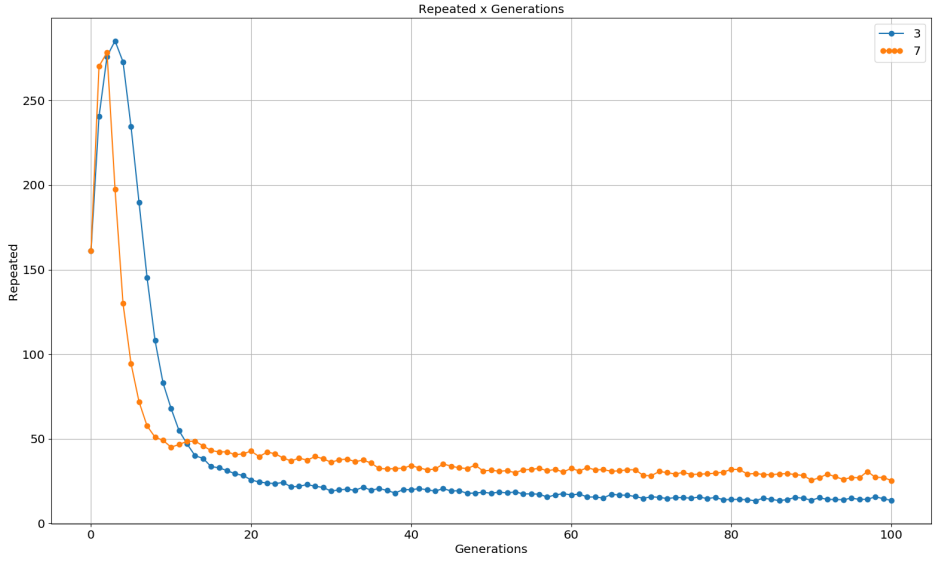

Figure 15. Indivíduos repetidos para *house* com diferentes valores de torneio.

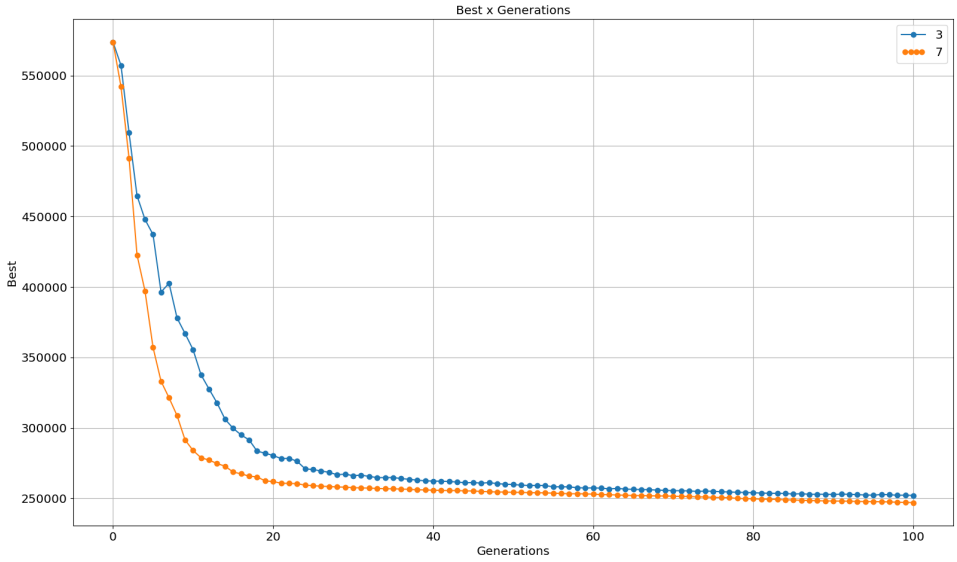

Figure 16. Comparação de diferentes valores de torneio na melhor *fitness* de *house*.

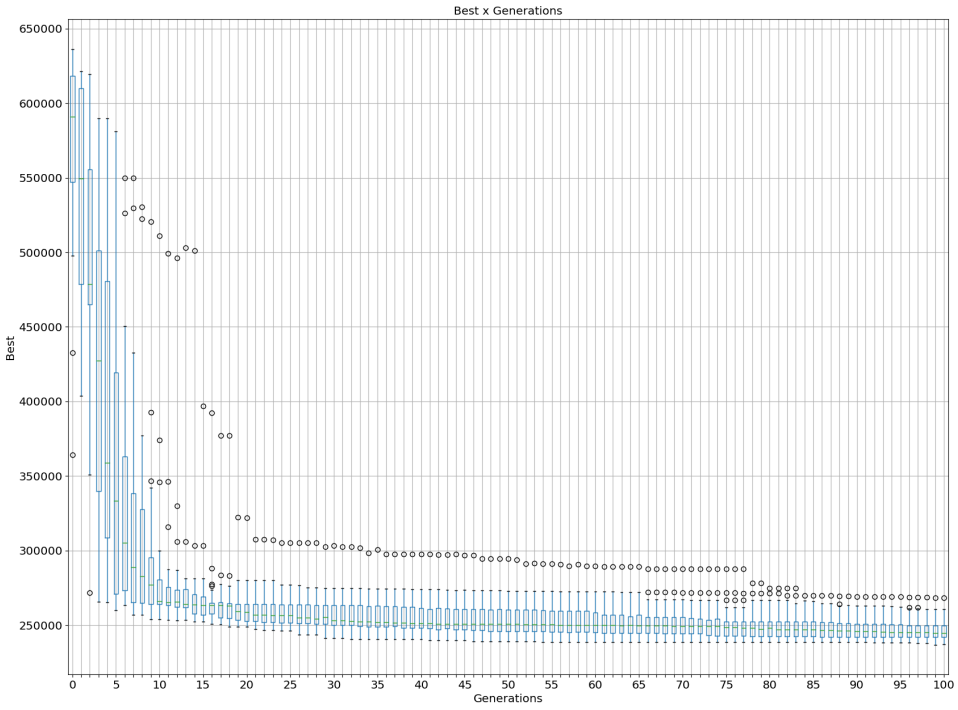

Figure 17. Melhor *fitness* de *house* com os melhores parâmetros.

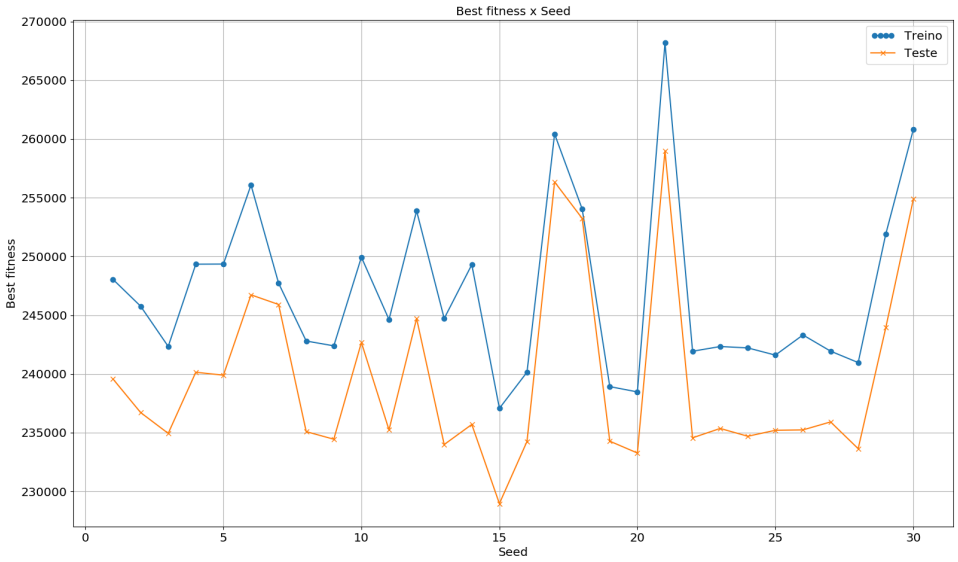

Figure 18. *fitness* de *house* para treino e para teste.

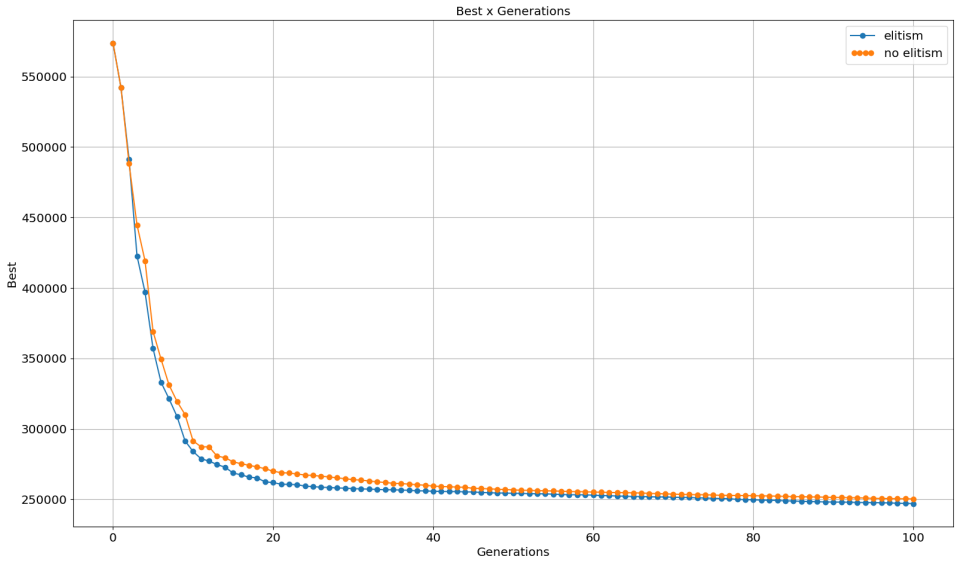

Figure 19. Comparação da melhor *fitness* de *house* com e sem elitismo.

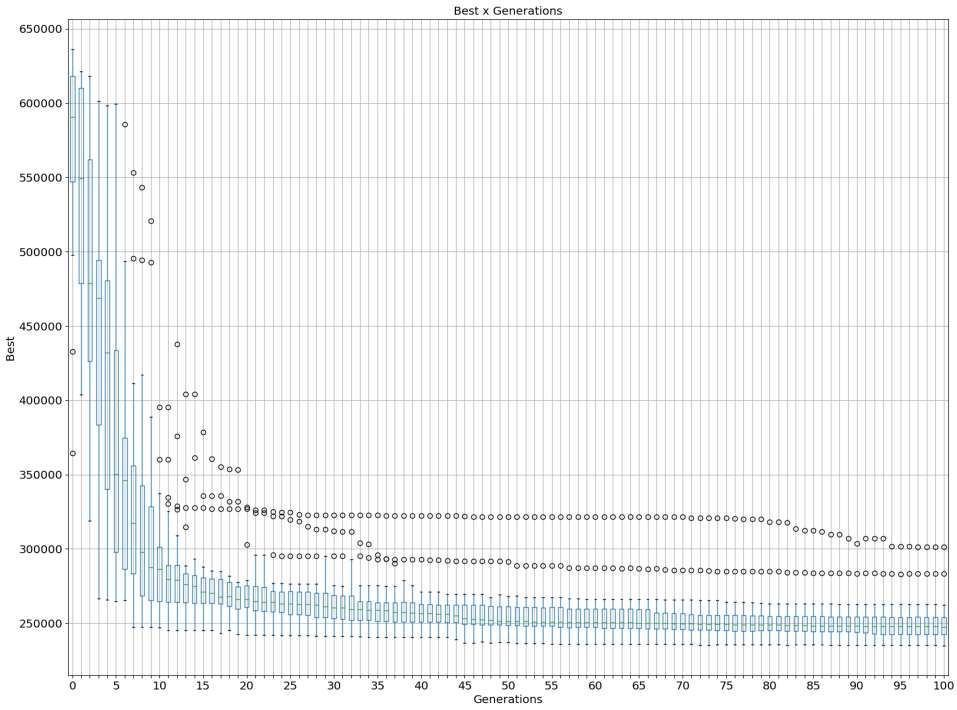

Figure 20. Melhor *fitness* de *house* com os melhores parâmetros e sem elitismo.