UNIVERSIDADE FEDERAL DE ITAJUBÁ
Campus Itabira

Engenharia da ComputaçãoECO012 –

Algoritmos Genéticos

21693 – Emmanuell Nogueira Miqueletti

21197 – Matheus Venturyne Xavier Ferreira

Prof. Dr. Sandro Izidorio

sandroizidoro@unifei.edu.br

Itabira – MG

2 de junho de 2015

Sumário

1 Introdução

O seguinte relatório tem como objetivo demonstrar a implementação de dois problemas clássicos da computação: o problema do caixeiro viajante e o problema da mochila. Os dois problemas são não polinomiais o que quer dizer mesmo utilizando uma boa técnica como por exemplo a programação dinâmica não se consegue resolver em tempo satisfatório, também significa que o número de soluções cresce de forma não polinomial com a entrada. Como abordagem alternativa, foi implementado o algoritmo genético que nos dá uma boa aproximação da solução ótima com complexidade linear. O relatório fara uma breve explicação de programação dinâmica e como ela pode ser usada e depois explicara com detalhes os algoritmos genéticos.

2 Programação Dinâmica

Programação dinâmica é uma técnica usada para otimizar problemas, problemas estes que apresentam duas características: o problema pode ser resolvido em subestruturas ótimas e acontece uma superposição de subestruturas, ou seja, o algoritmo quebra o problema em subproblemas, resolve esses subproblemas e os recombina aproveitando o cálculo de subproblemas equivalentes. Pode ser aplicada por exemplo para otimização de cálculo de Fibonacci, multiplicação de matrizes, o problema do troco e outras infinitas aplicações.

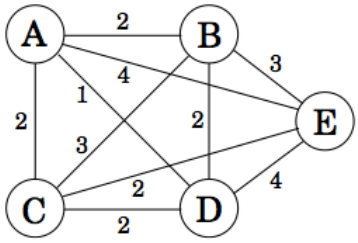
2.1 Caixeiro Viajante com Programação Dinâmica

O problema do caixeiro viajante consiste em N cidades onde deve-se encontrar o ciclo de menor custo em que cada cidade é visitada apenas uma única vez. O custo de uma solução consiste na soma dos custos para se viajar entre cada cidade. O grande desafio é que o numero de rotas possíveis cresce de forma fatorial e faz com que seja inviável calcular todas combinações possíveis visto que por exemplo 13! = 6227020800. Com o poder de processamento que possuímos na atualidade poderíamos gastar séculos caso a entrada seja 100 cidades por exemplo. Vamos usar o seguinte exemplo para ilustrar como é o funcionamento do algoritmo do caixeiro viajante

Na figura **??**, um grafo com 5 cidades possui 4! = 24 ciclos possíveis. Para desenvolver o algoritmo de programação dinâmica considere um grafo $G = (V, E)$ onde $v \in V$ é uma cidade e cada $e = (v, v') \in E$ é uma conexão entre a cidade v e v'. Nos queremos um caminho que comece e termine em um $v \in V$.

Agora, numerando as cidades de 1 até N, suponha que comecemos na cidade 1 estejamos

Figura 1: Exemplo de uma modelagem para caixeiro viajante[2]



na cidade i. Tenhamos de ir em todas as cidades em um subconjunto A de V. Nós podemos escolher qualquer uma das cidades $j \in A$ pagando o custo $c_(i, j)$ para ir da cidade i até j. Definimos então a variável $OPT(i, A)$ que armazena o menor custo possível para o problema de estar na cidade i e haver um conjunto de A cidades para se visitar. Portanto, pode-se definir a seguinte equação de recorrência:

$$OPT(i, A) = \begin{cases} min_{j \in A}\{OPT(j, A - \{j\}) + c_{i,j}\} & : A \neq \emptyset \\ c_{i,1} & : A = \emptyset \end{cases}$$

O caso para $A = \emptyset$ corresponde ao retorno da última cidade à cidade inicial 1.

Esse algoritmo foi implementado utilizando memorização e sua corretude parte do fato de serem investigadas todas as soluções possíveis. Na análise da complexidade considere que há N cidades e existem $O(2^N)$ subconjuntos A de V. Além disso para cada solução do subproblema $OPT(i, A)$ deve-se comparar todas as soluções para o subproblemas $OPT(j, A - \{j\})j \in A$. Portanto a complexidade do algoritmo é $O(n^2 2^n)$.

2.2 Problema da Mochila com Programação Dinâmica

O problema da mochila consiste em dado um conjunto I itens onde cada item $i \in I$ possui um valor v_i e um peso w_i. Deseja-se colocar tais itens em uma mochila de capacidade W de forma a maximizar a soma dos itens inseridos. Este problema é um problema NP-completo clássico da computação em que também pode-se utilizar programação dinâmica para redução de sua complexidade.

Existem diferentes versões para o Problema da Mochila; no entanto, neste trabalho, concentra-se na Mochila 0-1 ou Mochila binária onde dado um item ele está presente ou não está presente na mochila. Outras variações permitem que se repita itens na mochila; no entanto, tais versões podem ser facilmente reduzidas à Mochila binária.

No projeto do algoritmo de programação dinâmica considere um subproblema onde tenhamos o item i e a mochila possua capacidade W. Temos duas escolhas possíveis: adicionamos o item na mochila, reduzindo sua capacidade em w_i unidades e somando v_i ao valor da mochila; ou não adicionar o item na mochila, mantendo o seu valor e peso. Logo a equação de recorrência é dada por:

$$OPT(i, W) = \begin{cases} max\{v_i + OPT(i+1, W - w_i), OPT(i+1, W)\} & : w_i <= W \\ OPT(i+1, W) & : w_i > W \end{cases}$$

A corretude segue do fato de anualizarmos todas as soluções possíveis. Como desejamos computar $OPT(1, W)$ onde W é a capacidade inicial da mochila a complexidade desse algoritmo é $O(NW)$. Apesar de aparentar possuir complexidade linear esse algoritmo é pseudo-polinomial uma vez que W é um parâmetro número do problema, ou seja, a complexidade cresce exponencialmente com a representação em bits de W.

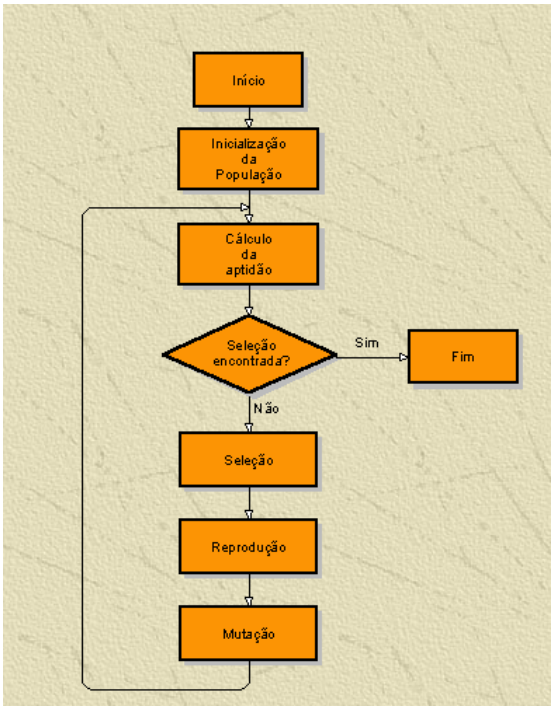
3 Algoritmos Genéticos

Nessa seção faremos uma breve explicação de como funcionam os algoritmos Genéticos. Estes nada mais são que técnicas para a solução de problemas de otimização, como o Problema da Mochila e o Caixeiro Viajante.

É um método inspirado na biologia evolutiva onde possui muitas similaridades com a mesma. Vamos partir do seguinte pressuposto: Todos os seres vivos que existem hoje são evoluções de milhões de anos de suas gerações passadas. Os seres que hoje habitam a terra se adaptaram a diversas situações através de mutações genéticas para garantir a sobrevivência da espécie. Essa analogia pode ser aplicada para fazer um algoritmo computacional, onde os indivíduos são na verdades soluções do problema. Como Darwin teorizou o mais forte sobrevive, logo se uma solução não é "forte"ela tende a "morrer". Para ver como funciona o algoritmo genético veja o fluxograma na Figura 2.

Inicialmente é gerada uma população de indivíduos, possíveis soluções do problema. Em segundo, os indivíduos passam por um cálculo de aptidão também conhecido com função Fitness, a mesma função que na natureza escolhe os mais fortes. É a maneira que escolhemos para seguir em frente com uma solução ou não. Por exemplo, o Fitness do caixeiro viajante é a soma dos custos das arestas da rota. Nesse caso, um baixo valor de Fitness consequentemente tem mais chances de passar para a próxima geração. No problema da mochila, o Fitness é

Figura 2: Funcionamento do algoritmo genético[2]



uma função que soma o valor dos itens inseridos com a restrição de que o peso limite não seja ultrapassado.

O Fitness é uma maneira de garantimos que estamos indo de acordo com o nosso objetivo, é uma função vital nos algoritmos genéticos. Terceiro passo, é testado a condição de parada que pode ser o número de gerações. Caso não aconteça a parada vamos partir para o quarto passo, uma seleção de indivíduos; essa seleção pode ser feita por roleta, torneio ou ainda outros métodos.

Na seleção por roleta é sorteado quem vai para próxima geração de acordo com sua Fitness. No torneio dois indivíduos são sorteados e o indivíduo de maior Fitness vence. No quinto passo ocorre a reprodução onde um pedaço do genoma de dois indivíduos é recombinado. No último passo, ocorre a mutação localmente nos genes. Por exemplo, na mochila um item pode ser retirado ou colado por meio da mutação do gene. Esse ciclo se repete quantas vezes for desejado até atingir a condição de parada.

É importante frisar que caso não conheçamos a melhor solução não podemos afirmar que o algoritmo genético irá encontra-la pois é um método probabilístico.

3.1 Caixeiro Viajante com Algoritmos Genéticos

O problema do caixeiro viajante já foi previamente apresentado. Nessa seção vamos explicar como foi feito o algoritmo genético para o caixeiro viajante.

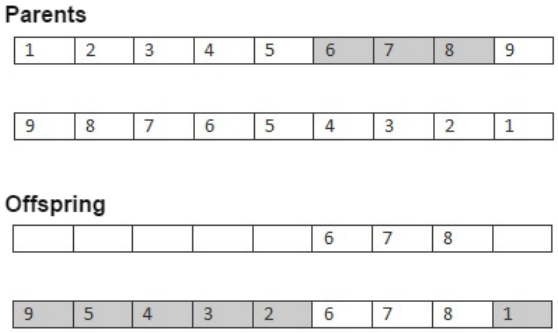
O genoma de uma solução foi representado por um tamanho fixo com uma sequência de números de 1 a N que não se repete. Foi utilizada a seleção por torneio com elitismo e como deseja-se minimizar o custo da rota, quanto menor a Fitness melhor a solução. Para realizar as operações genéticas, é preciso tomar cuidado para não gerar soluções invalidas, ou seja, soluções que repitam uma cidade. Portanto, para a mutação de um gene é escolhido aleatoriamente outro gene e estes são trocados de posição no genoma. Por exemplo,

$genoma = [1, 2, 3, 4]$ $genoma = [1, 3, 2, 4]$

Com a mutação do segundo gene é escolhido aleatoriamente o terceiro gene para trocar de lugar.

Para o crossover, escolhe-se um pedaço do genoma do pai, salvando-o no novo filho e depois o pedaço do genoma restante é completado com os genes do outro pai em sequência desde que esta cidade ainda não pertença ao pedaço já implantado no filho como pode-se ver na Figura 3.

Figura 3: Demonstração do cruzamento usado para o caixeiro viajante[3]



Realizando-se vários testes, obteve-se resultados satisfatórios para uma taxa de mutação de 15%, taxa de cruzamento de 70%, população com 100 indivíduos e executando 100 gerações.

3.2 Problema da Mochila com Algoritmos Genéticos

O problema da mochila possui implementação em algoritmos genéticos mais trivial. O genoma foi representado por uma string de tamanho constante com bits 0s e 1s onde cada bit representa um item do problema. O bit 0 significa que um item não está na mochila e 1 se o item se encontra na mochila.

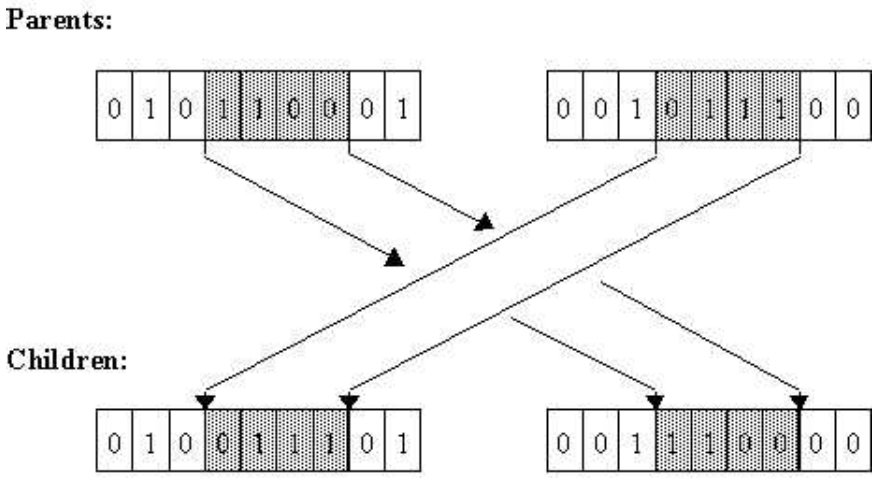
Ao executar os operadores genéticos não se preocupa-se com o limite do peso da mochila; no entanto, ao se calcular a Fitness da solução, quando o peso limite é excedido, itens são removidos aleatoriamente da mochila até que se obtenha uma solução adequada. Para a seleção

foi empregado torneio com elitismo. O operador de mutação consiste apenas em trocar um gene 0 para um gene 1 e um gene 1 para um gene 0. O crossover consiste em definir um breakpoint no genoma dos pais e atribuir aos filhos parte do genoma do primeiro pai e parte do genoma do segundo pai.

Por exemplo, considere um problema com 4 itens com uma solução representada pelo genoma "0110". Essa informação nos diz que o segundo e o terceiro item estão na mochila apenas. O Fitness da solução é a soma dos valores dos itens 2 e 3.

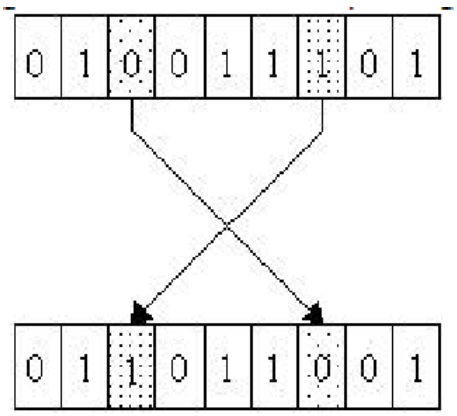
Realizando-se vários testes, obteve-se resultados satisfatórios para uma taxa de cruzamento de 70%, taxa de mutação de 1%, população com 100 indivíduos e executando 100 gerações.

Figura 4: Demonstração de um cruzamento no Problema da Mochila[4].



A figura 4 demonstra como ocorre o cruzamento entre dois indivíduos no problema da mochila. Podemos notar que pegamos uma parte aleatória dos pais e recombinamos nos filhos.

Figura 5: Demonstração da mutação de um indivíduo no Problema da Mochila[4].



A figura 5 mostra como ocorre a mutação de um indivíduo no caso do problema da mochila.

4 Resultados

Foi implementado um framework em C++ para a implementação de algoritmos genéticos. Esse framework foi aplicado ao problema da mochila e ao problema do Caixeiro Viajante. Além disso, foi desenvolvido wrappers para logging automático dos testes e um conjunto de scripts em MATLAB para a geração dos gráficos a partir dos dados obtidos. Para os testes foram implementadas funções para a geração de problemas aleatórios respeitando limites estabelecidos. O framework pode ser acessado em `https://github.com/matheusvxf/GeneticAlgorithm`.

4.1 Problema da Mochila

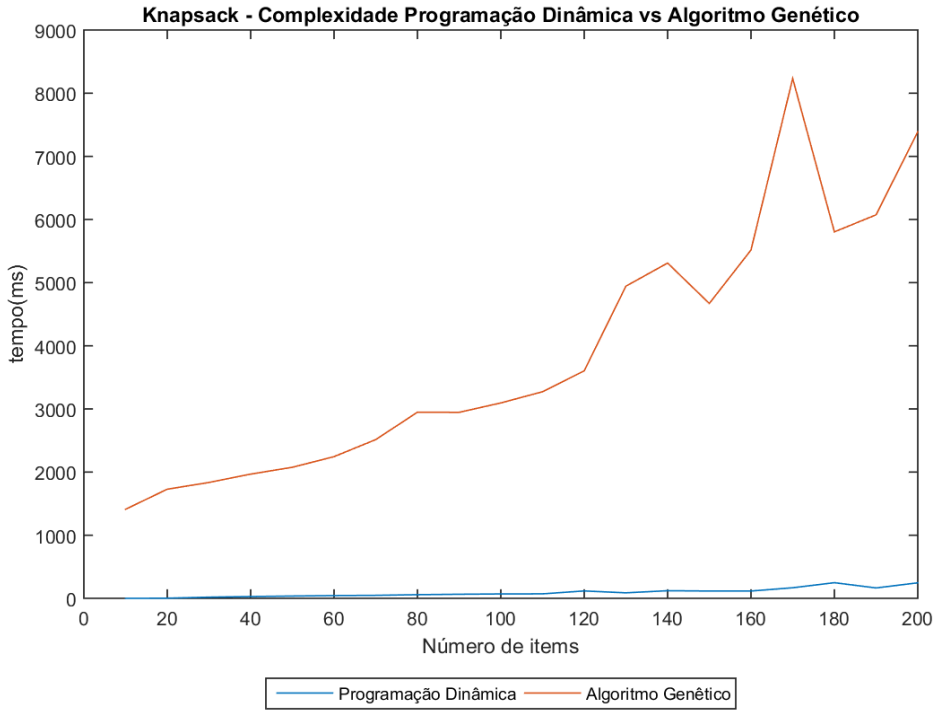
Inicialmente foi analisada a complexidade do algoritmo genético e da programação dinâmica. Primeiramente, foram executados testes para o peso limite da mochila de 1000 com o número de itens variando de 10 a 200. Na Figura 6 vemos que o algoritmo de programação dinâmica se comportou muito melhor que o algoritmo genético uma vez que este último é exponencial em W e linear no número de itens. Em um segundo teste, visando demonstrar a característica pseudo-polinomial do algoritmo de programação dinâmica, foi testado para uma problemas com 100 itens e peso da mochila variando de 10000 até 200000, Figura 7. Neste experimento foi plotado o logaritmo do peso da mochila e vemos que de fato o tempo de execução da programação dinâmica cresce exponencialmente enquanto o algoritmo genético independe do peso limite da mochila.

Em seguida, foram analisados vários gráficos da evolução da Fitness do pior caso, melhor caso, média da Fitness da população, variância da Fitness da população ao longo das gerações e uma reta horizontalmente com a Fitness da solução ótima alcançada pelo algoritmo de programação dinâmica. Todos os gráficos para o número de itens variando de 10 a 200 se encontram no Apêndice.

Como podemos observar o algoritmo genético para 10 itens convergiu para a solução ótima com menos de 20 gerações. Os testes realizados para a mochila com 10, 20, 30 e 40 itens mostram resultados bastante satisfatórios onde obteve-se a melhor solução em quase todas as repetições do teste, Figura 25 ,26, 27 e 28.

Para os testes onde o universo de soluções é maior podemos perceber que ele se mostra menos preciso e acaba encontrando uma solução boa mas não próxima da desejada. No entanto, aumentando o tamanho da população e o número de gerações para instâncias maiores do problema, ou mesmo executando o algoritmo várias vezes permite-se aproximar a solução ótima.

Figura 6: Complexidade de acordo com o número de itens. Fonte: Autoria própria.



Como exemplo, foi executado o algoritmo genético 30 vezes para o problema da mochila com 100 itens e obteve-se o resultado da Figura 8 onde vemos que algumas instâncias de execução conseguiram alcançar a solução ótima. Neste exemplo, a média da Fitness das melhores soluções foram 917.3667 com desvio padrão de 29.53 sendo a solução ótima com Fitness de 975.

Figura 7: Complexidade do peso da mochila. Fonte: Autoria própria.

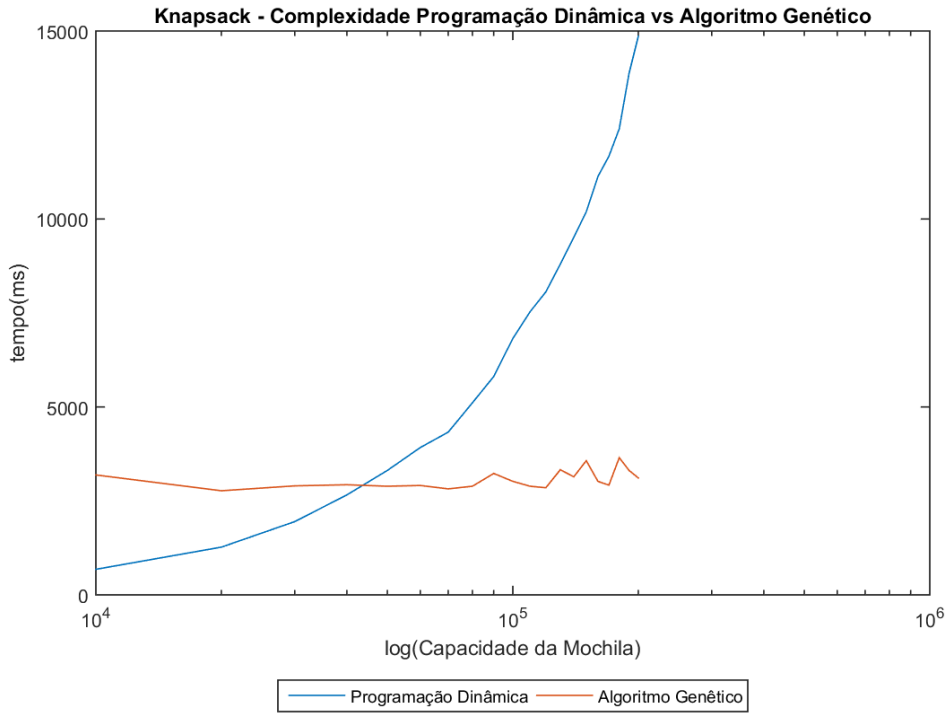
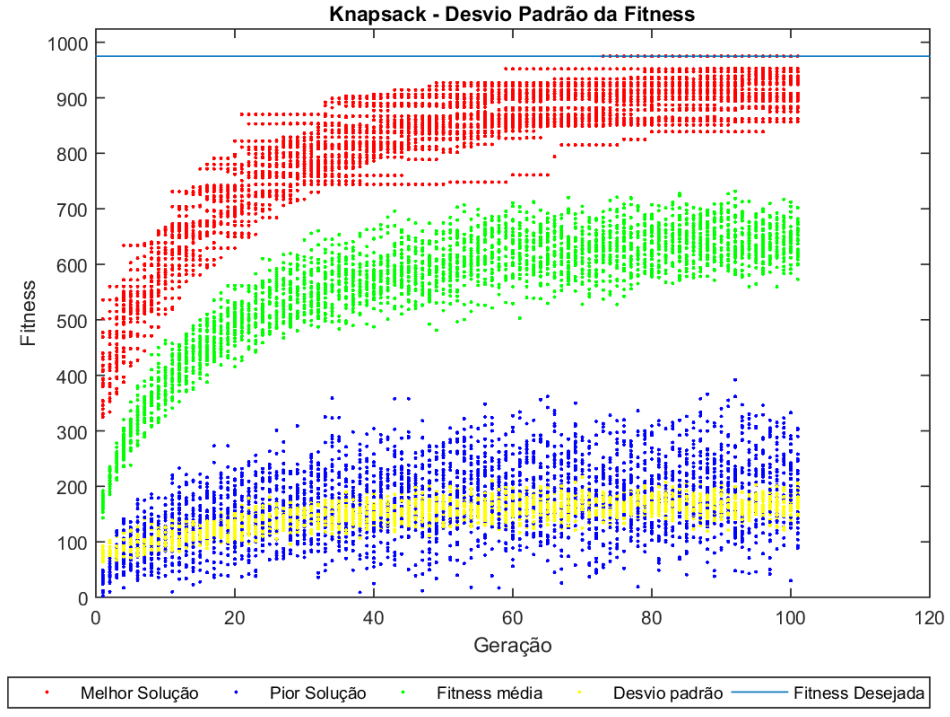


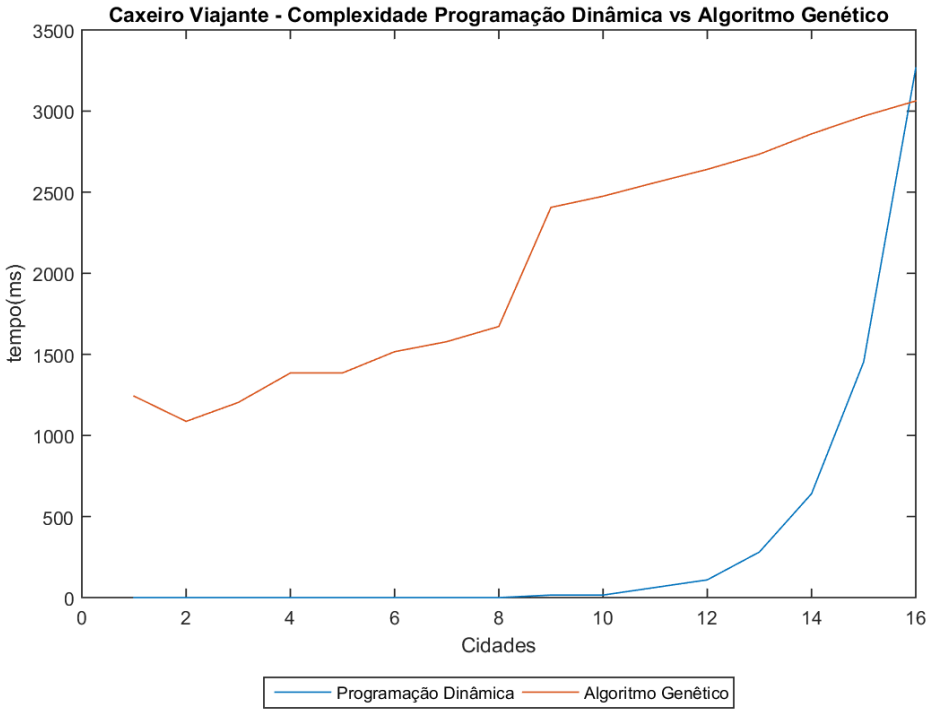
Figura 8: Knapsack - Comparação de 30 execuções para o problema da mochila com 100 itens. Fonte: Autoria própria.



4.2 Problema do Caixeiro Viajante

Primeiramente foi analisado a complexidade do algoritmo genético (AG) e da programação dinâmica. Na Figura 9, vou plotado o gráfico do tempo gasto na execução para o número de cidades variando de 1 até 16. Vemos o comportamento linear do AG em comparação ao crescimento exponencial da PG.

Figura 9: Complexidade Programação Dinâmica vs Algoritmo Genético. Fonte: Autoria própria.



Na análise da evolução da Fitness frequentemente o AG foi capaz de encontrar a melhor solução para até 11 cidades, Figura 10.

Para analisar a confiabilidade do algoritmo, este foi executado para um problema com 16 cidades 30 vezes. Como podemos ver na Figura 11 nenhuma instância alcançou a melhor solução com Fitness de 3536; por outro lado, a melhor solução encontrada foi de Fitness 3614 - no geral uma média de 4237 e desvio padrão de 268,72.

Figura 10: Gráfico do caixeiro viajante para 11 cidades. Fonte: Autoria própria.

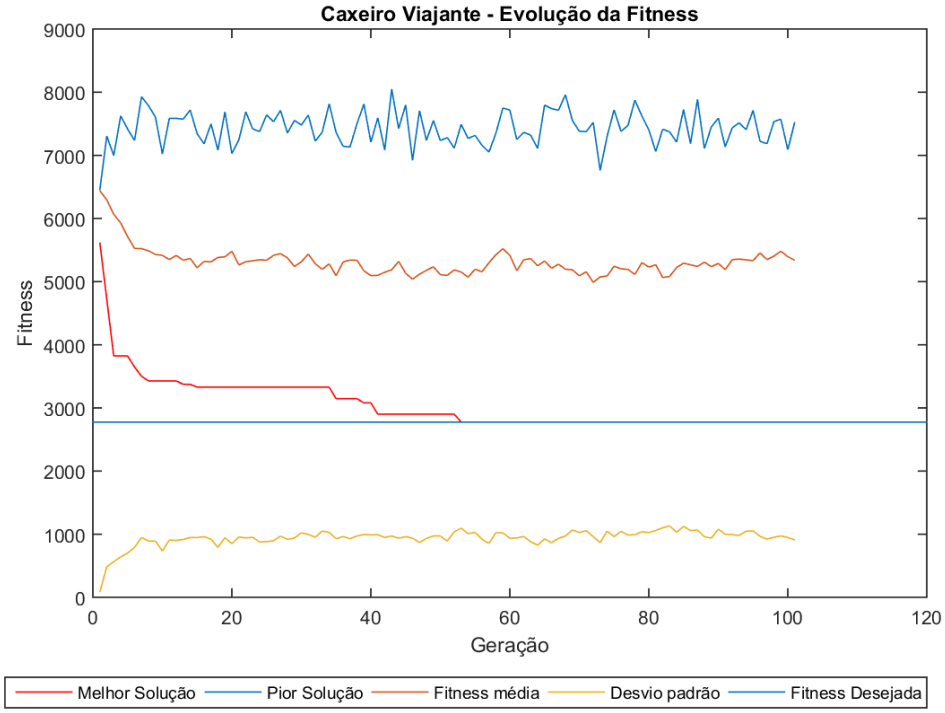
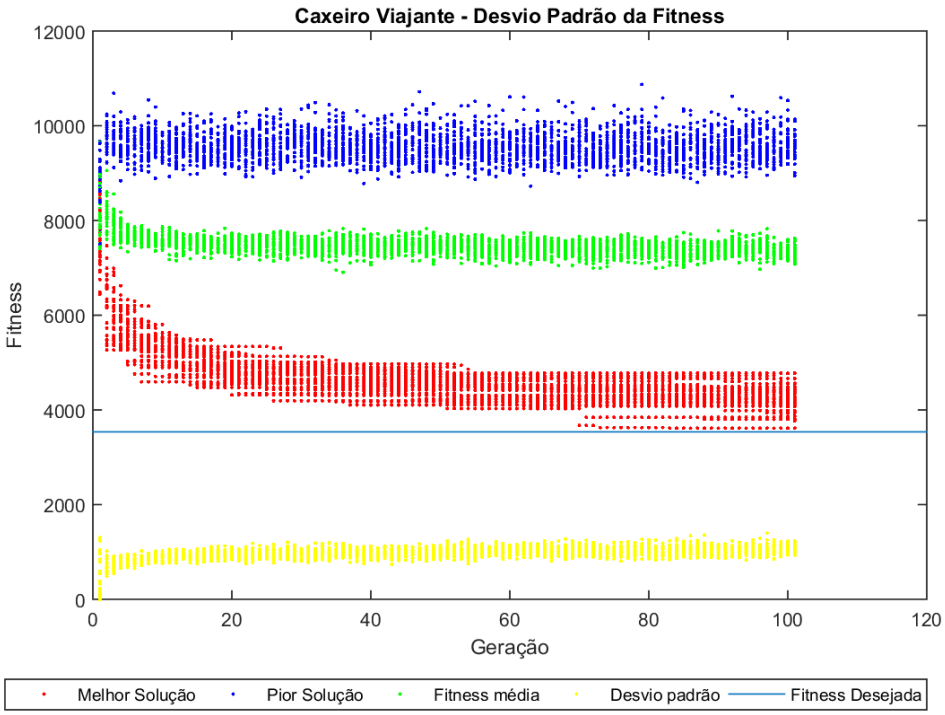


Figura 11: Comparação de 30 execuções para o problema do caixeiro viajante para 16 cidades. Fonte: Autoria própria.



5 Conclusão

Pode-se concluir que usando AG é possível resolver problemas clássicos de computação. A aplicação de um algoritmo genético possibilitou a obtenção de soluções aproximadas boas e muitas vezes soluções ótimas. Pode-se notar que este é uma ferramenta poderosa quando tamanho do problema cresce como pôde-se observar para o caixeiro viajante para 16 cidades e o problema da mochila para 100 cidades. Conclui-se ainda que implementar elitismo foi uma boa opção para manter o melhor indivíduo da população. Nos testes realizados obteve-se algoritmos com baixo desvio padrão como o desejado comprovando que as execuções do algoritmo convergem para boas soluções no geral.Os ajustes de nosso AG como a taxa de mutação, cruzamento, população e número de gerações também se mostraram importantíssimos para a implementação de um AG de qualidade. Ainda, esse trabalho proporcionou o desafio de lidar com problemas da NP-completos, um campo com vários problemas importantes da atualidade. Em trabalhos futuros seria interessante a análise de outras técnicas de seleção, mutação e cruzamento em vista de melhorar os resultados obtidos; melhorar a performance computacional do framework desenvolvido; e explorar técnicas para ajustes dos parâmetros genéticos.

6 Apêndice

Nessa seção foram plotados todos os gráficos requisitados, sendo dividido em duas subseções: uma para o caixeiro viajante e outra para o problema da mochila.

6.1 Gráficos do Caixeiro Viajante

Figura 12: Gráfico do caixeiro viajante para 5 cidades. Fonte: Autoria própria.

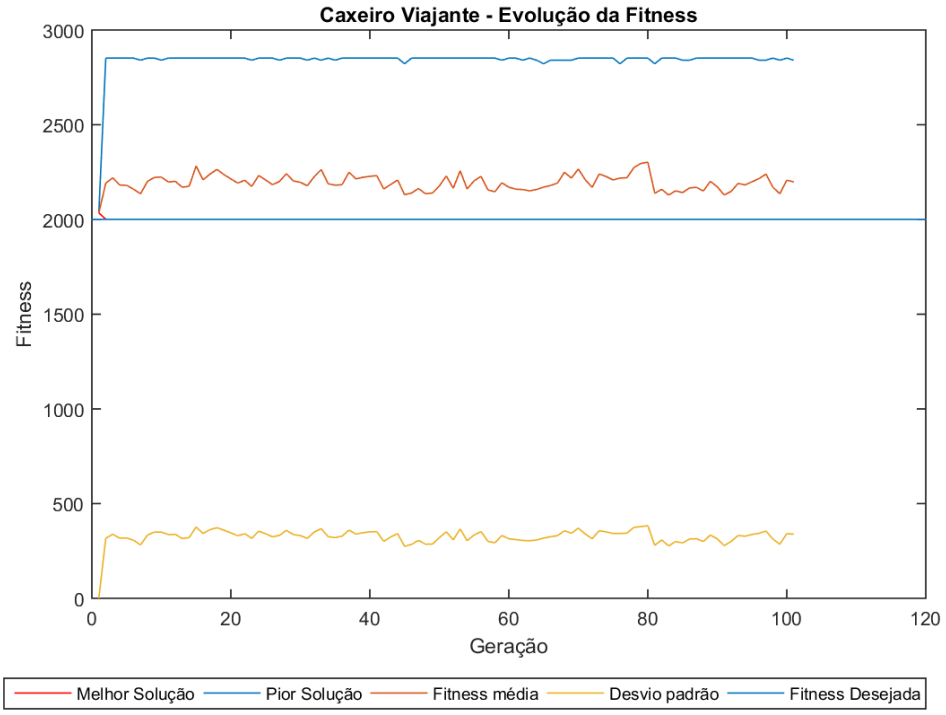


Figura 13: Gráfico do caixeiro viajante para 6 cidades. Fonte: Autoria própria.

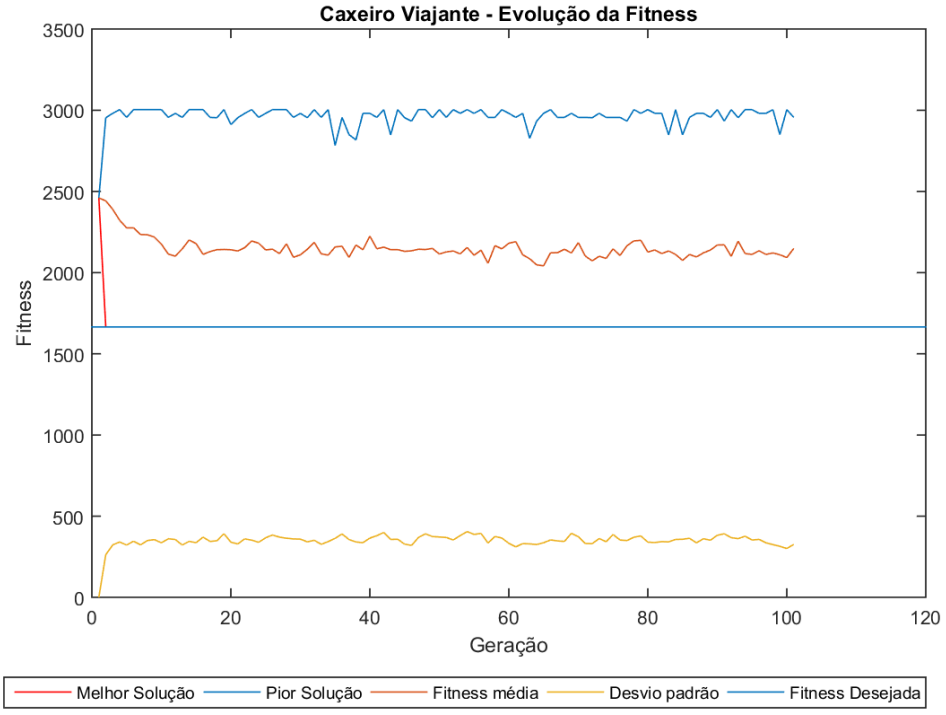


Figura 14: Gráfico do caixeiro viajante para 7 cidades. Fonte: Autoria própria.

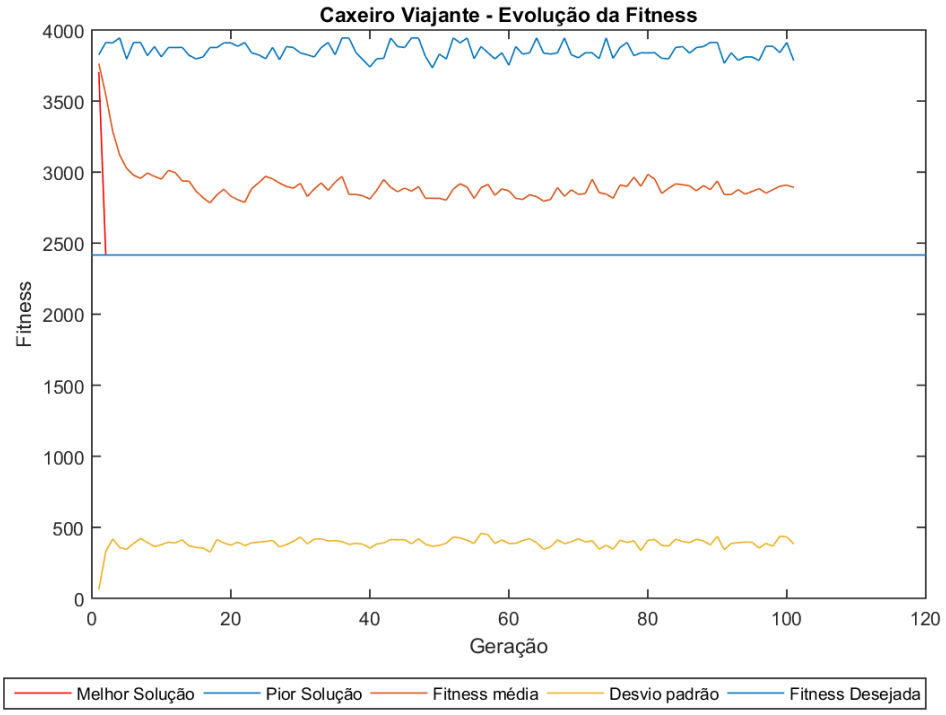


Figura 15: Gráfico do caixeiro viajante para 8 cidades. Fonte: Autoria própria.

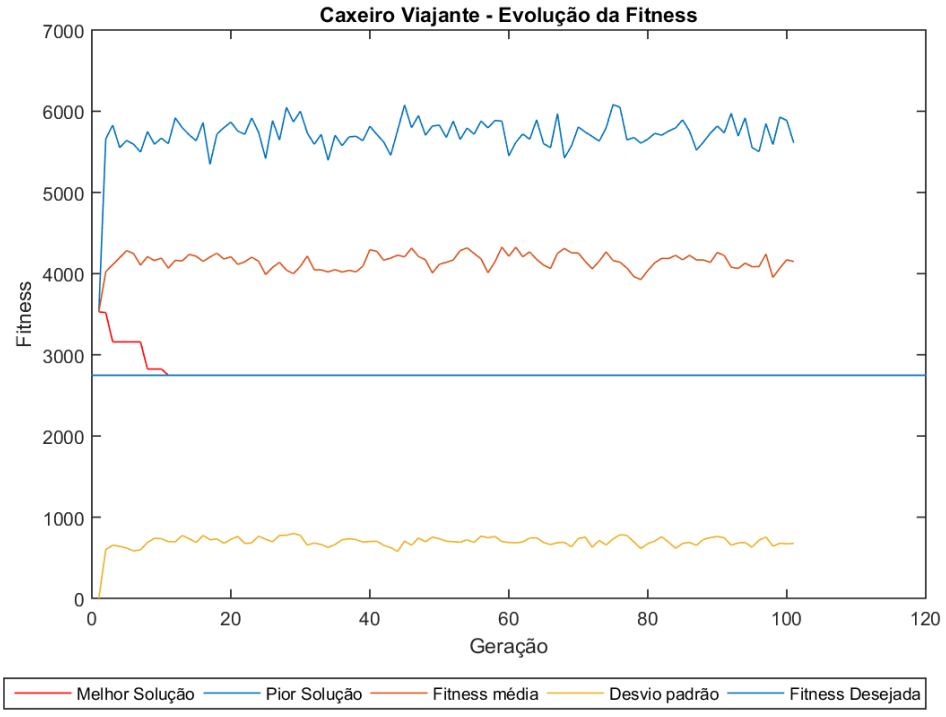


Figura 16: Gráfico do caixeiro viajante para 9 cidades. Fonte: Autoria própria.

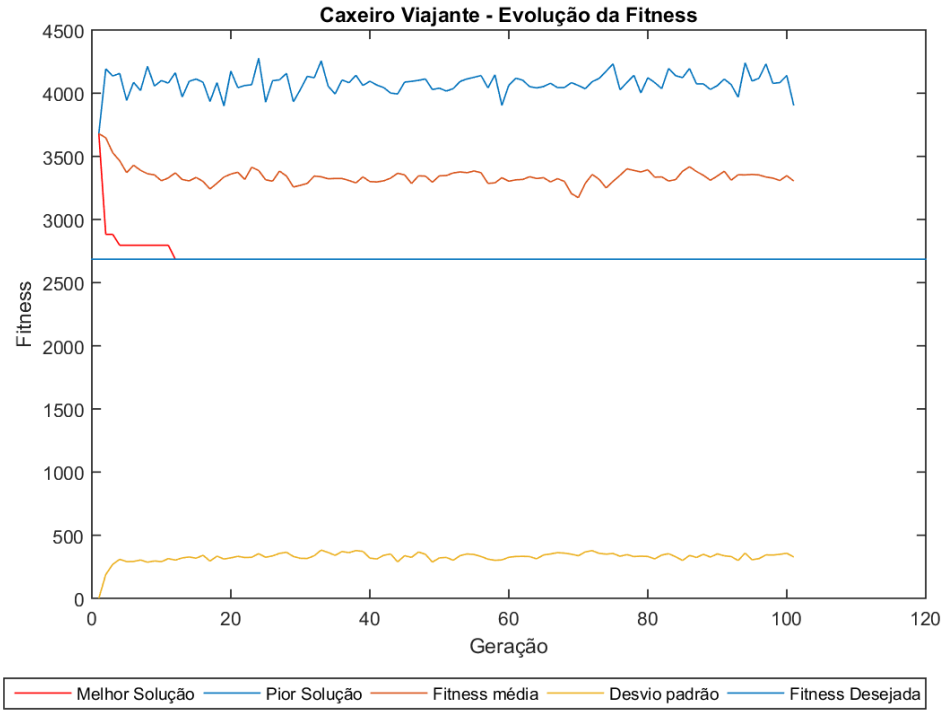


Figura 17: Gráfico do caixeiro viajante para 10 cidades. Fonte: Autoria própria.

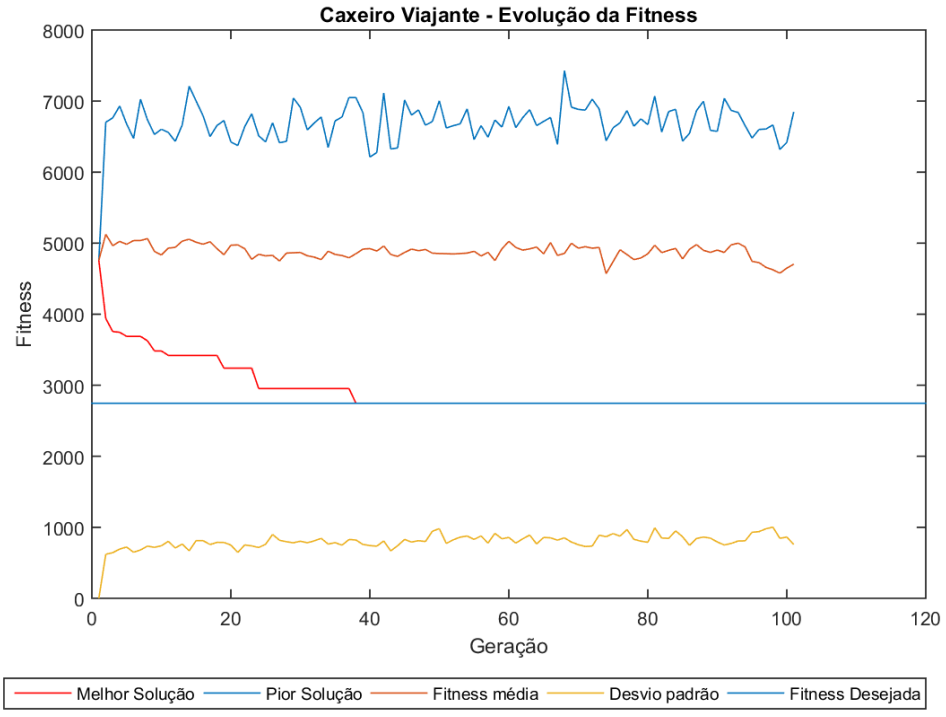


Figura 18: Gráfico do caixeiro viajante para 11 cidades. Fonte: Autoria própria.

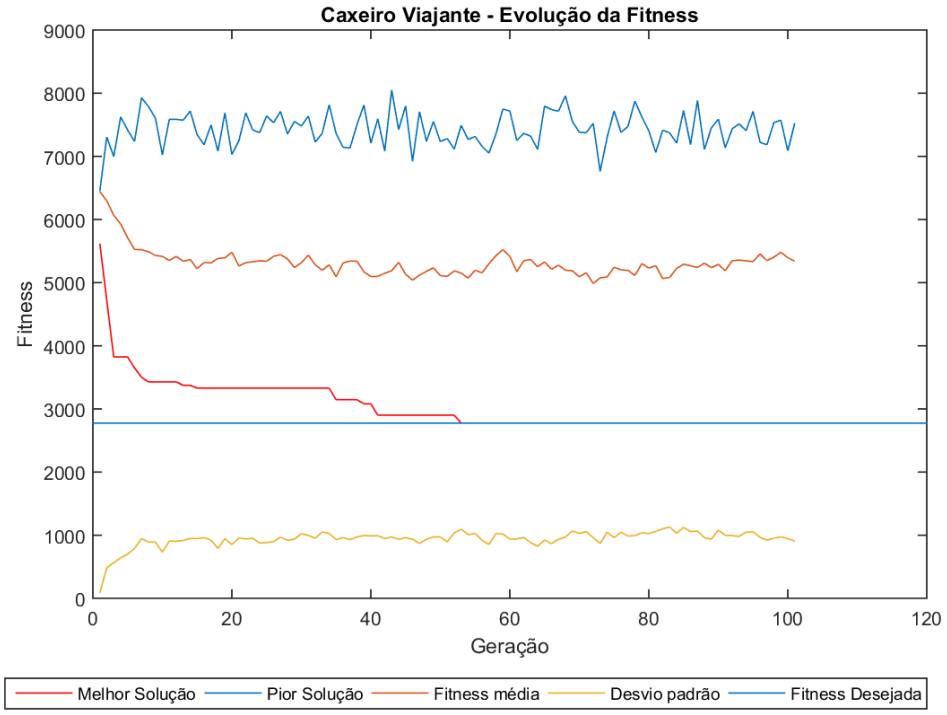


Figura 19: Gráfico do caixeiro viajante para 12 cidades. Fonte: Autoria própria.

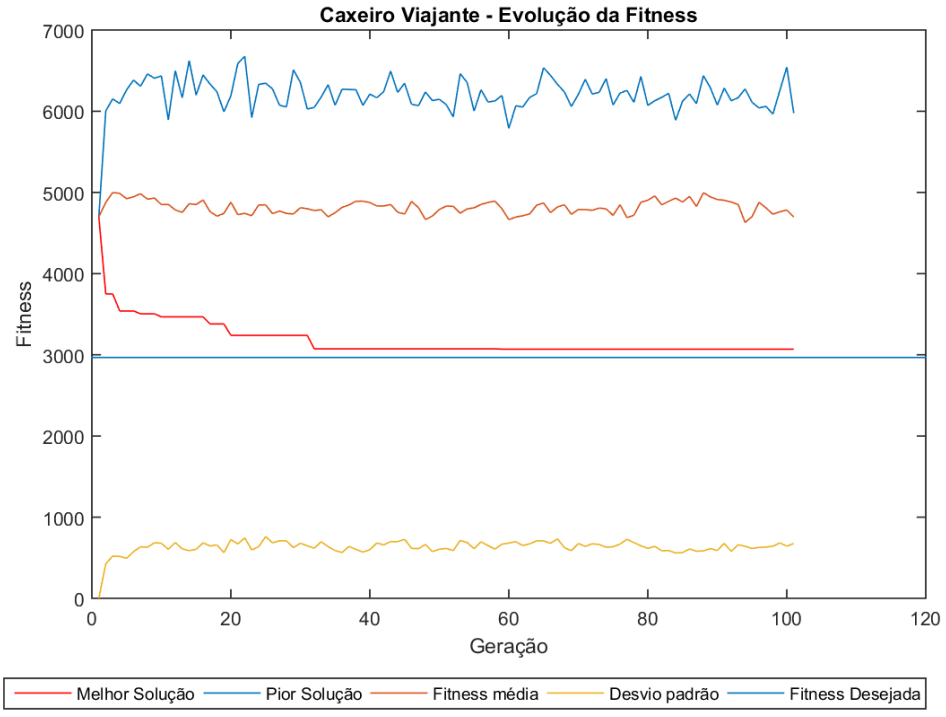


Figura 20: Gráfico do caixeiro viajante para 13 cidades. Fonte: Autoria própria.

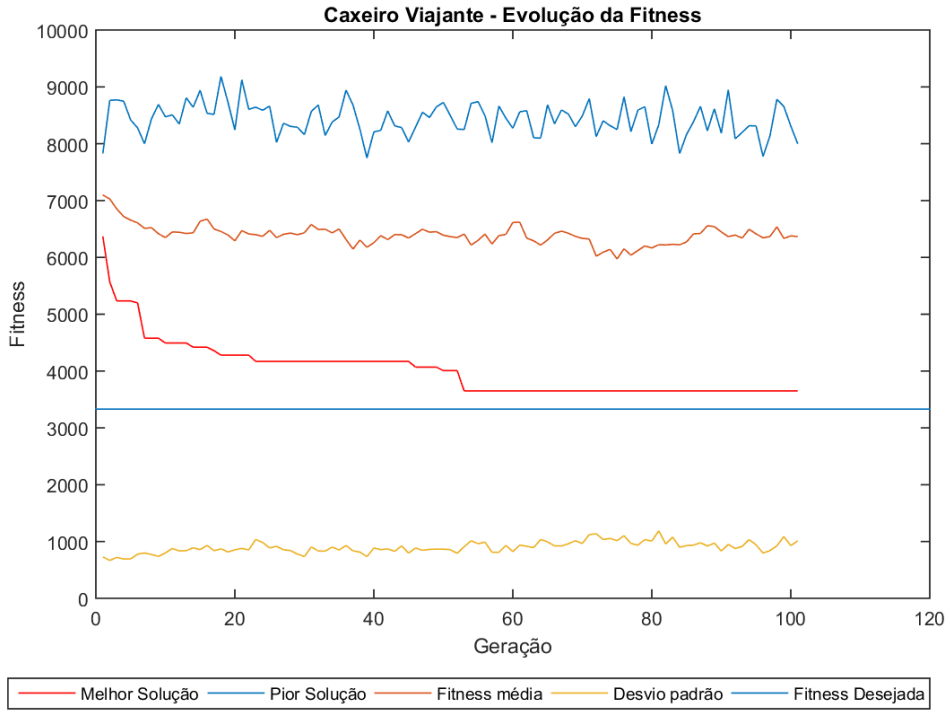


Figura 21: Gráfico do caixeiro viajante para 14 cidades. Fonte: Autoria própria.

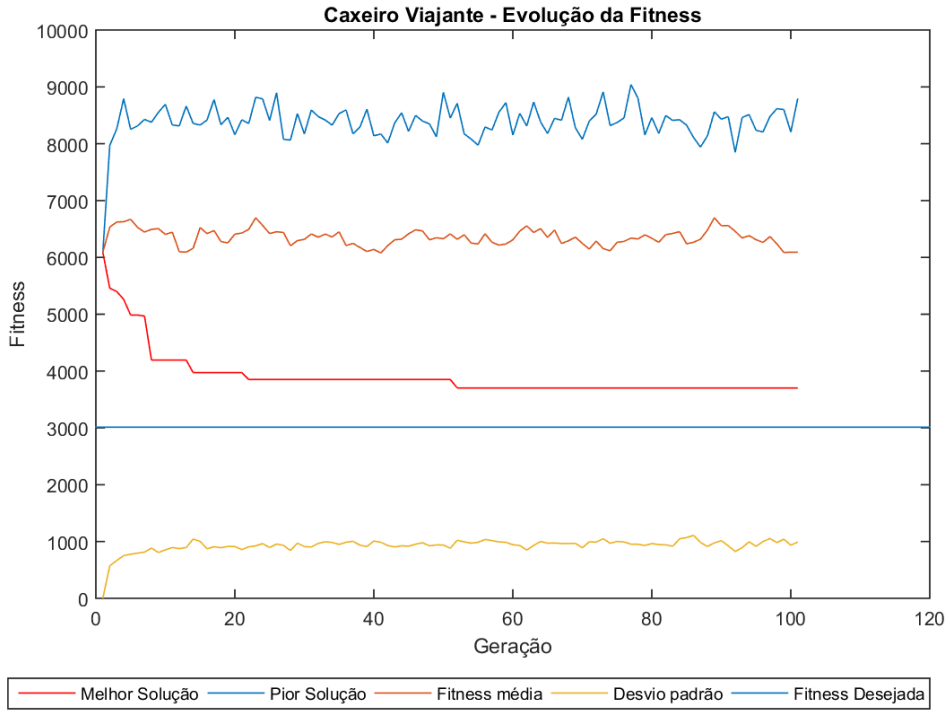


Figura 22: Gráfico do caixeiro viajante para 15 cidades. Fonte: Autoria própria.

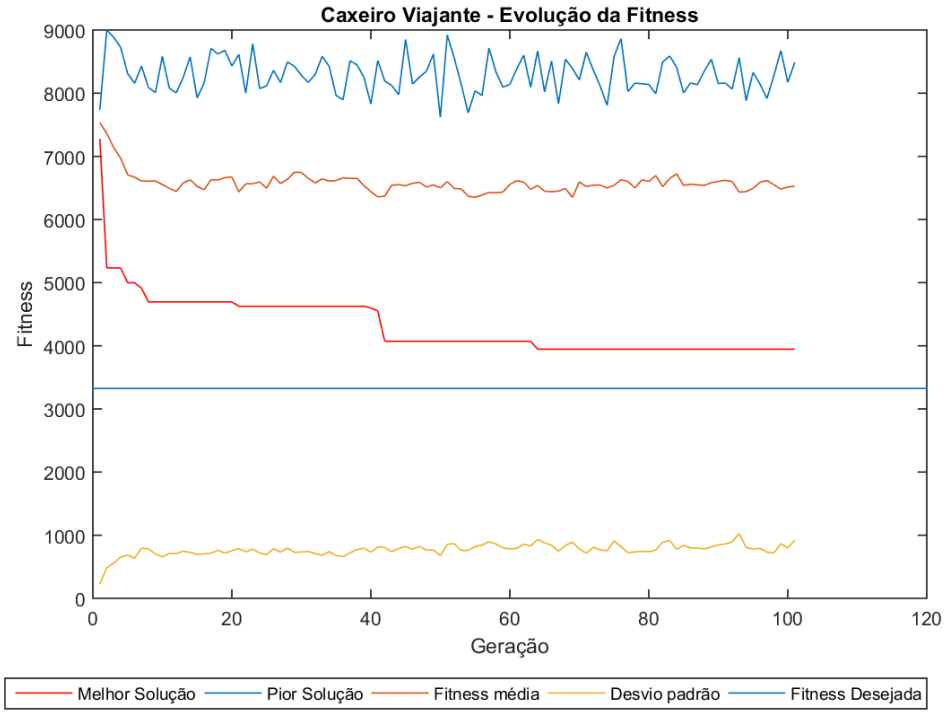


Figura 23: Gráfico do caixeiro viajante para 16 cidades. Fonte: Autoria própria.

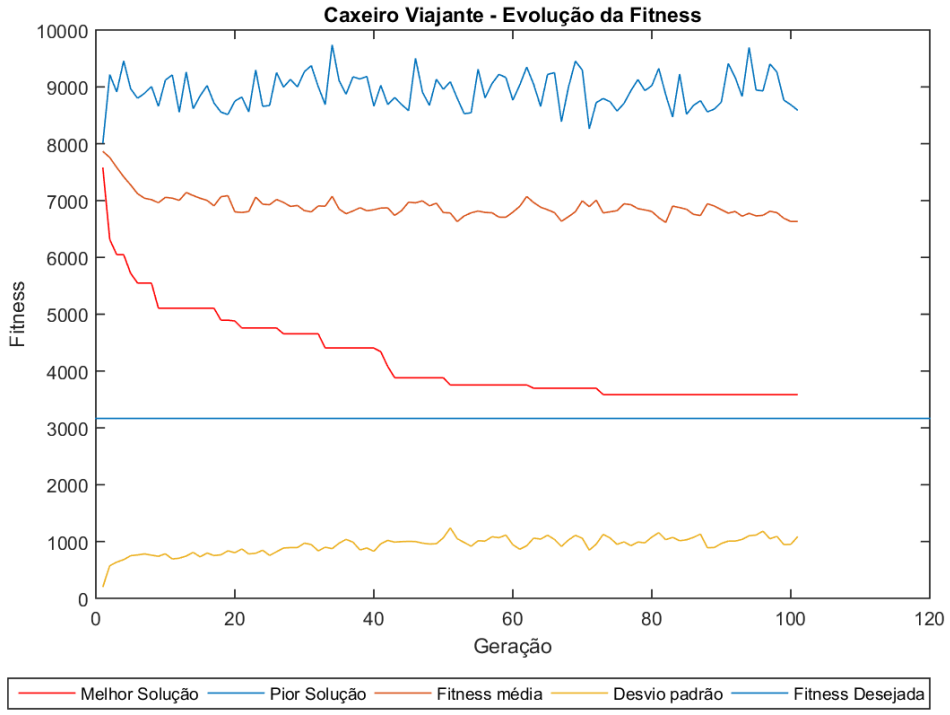
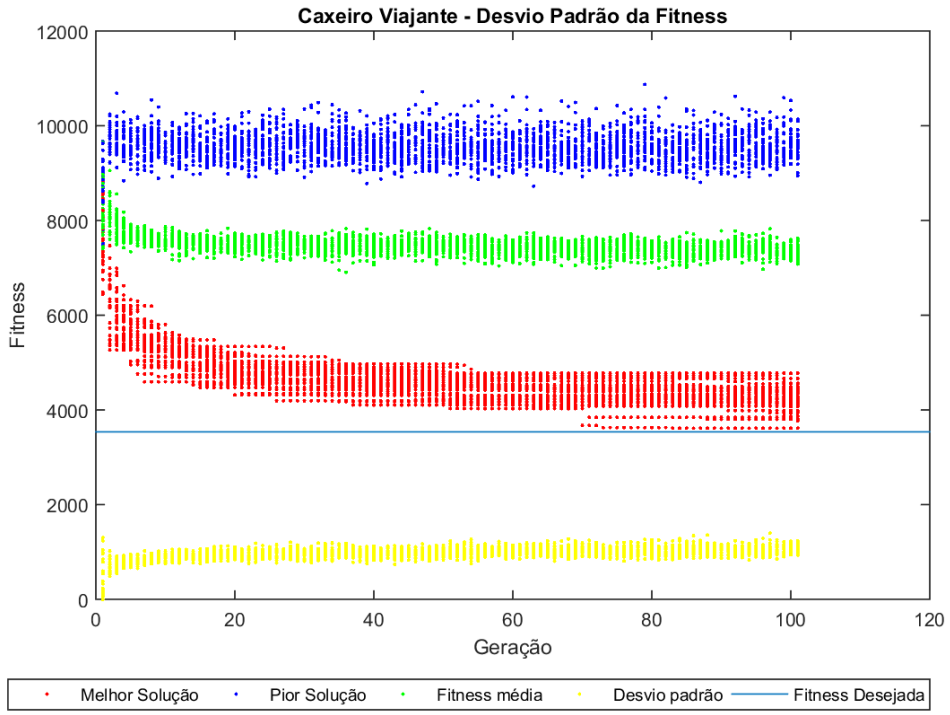


Figura 24: Os valores do desvio padrão para o caixeiro viajante



6.2 Gráficos do Problema da Mochila

Figura 25: Gráfico da Mochila para 10 itens. Fonte: Autoria própria.

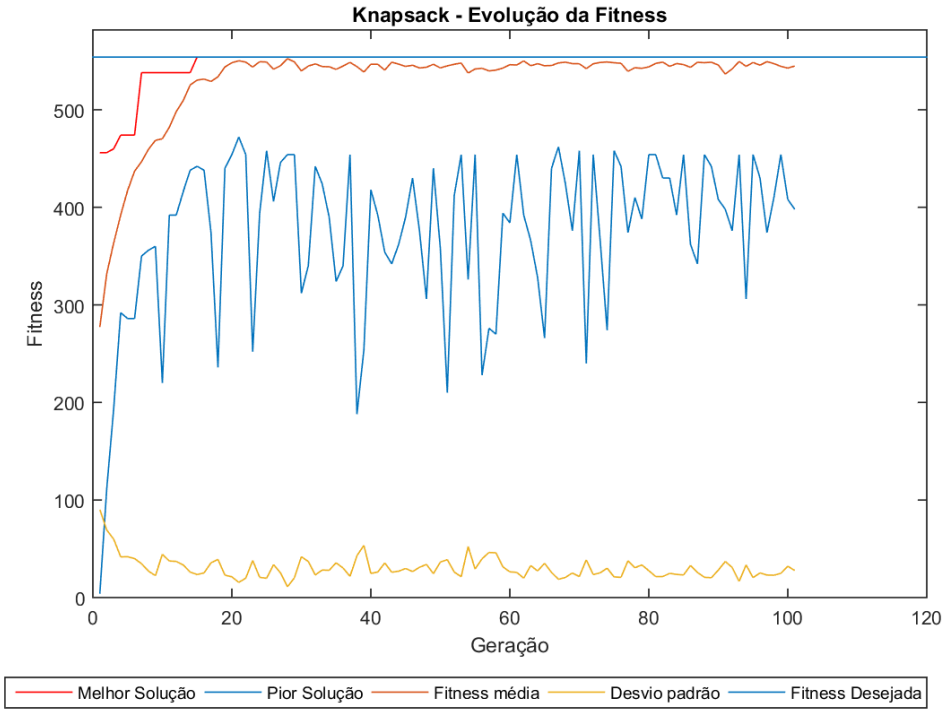


Figura 26: Grafico da Mochila para 20 itens. Fonte: Autoria própria.

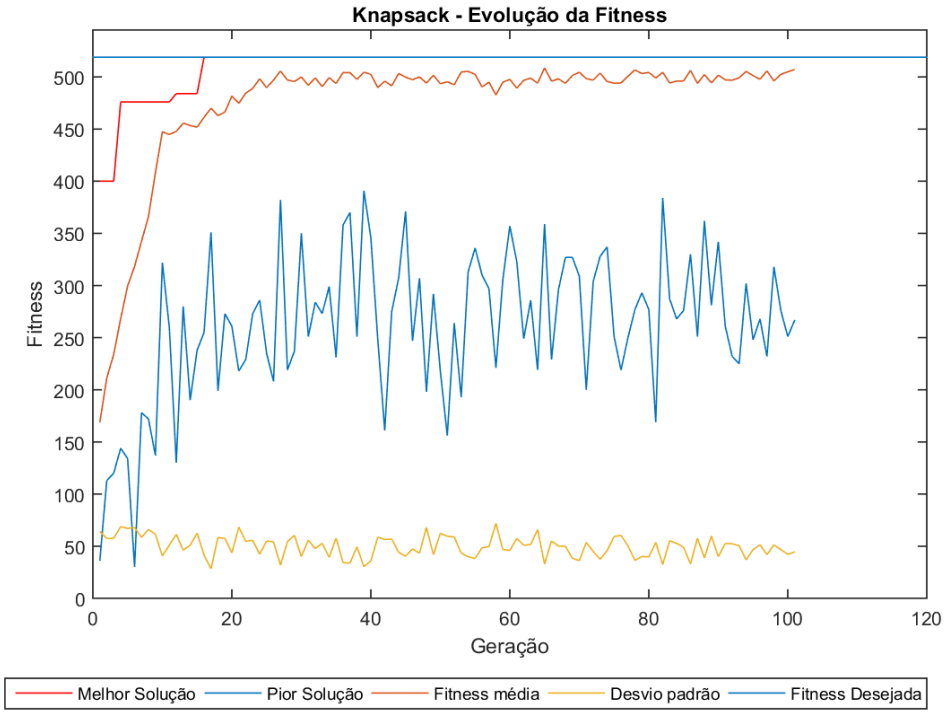


Figura 27: Grafico da Mochila para 30 itens. Fonte: Autoria própria.

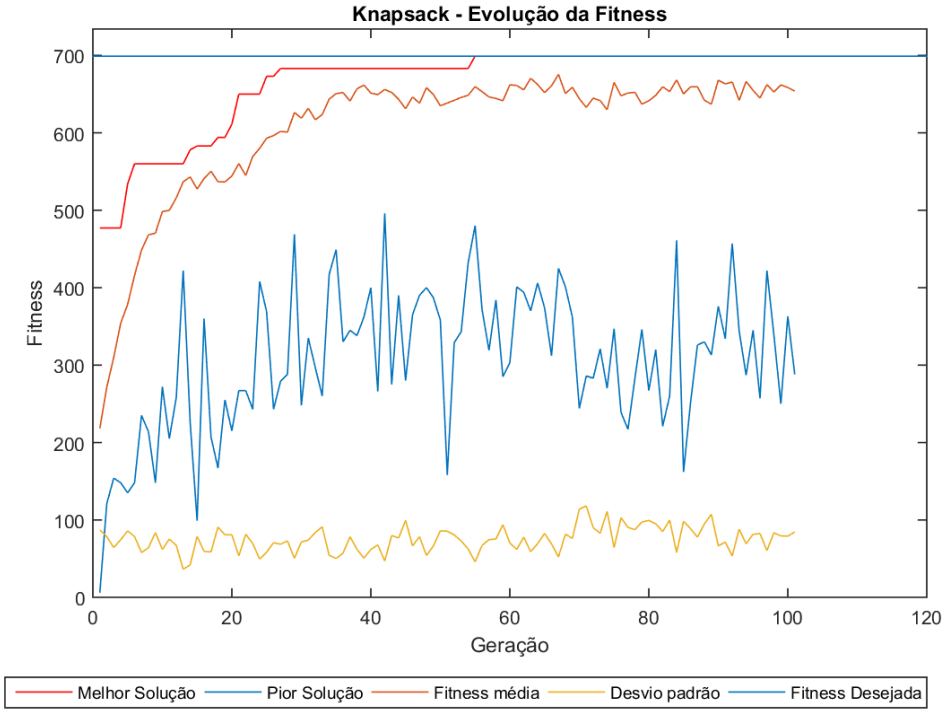


Figura 28: Gráfico da Mochila para 40 itens. Fonte: Autoria própria.

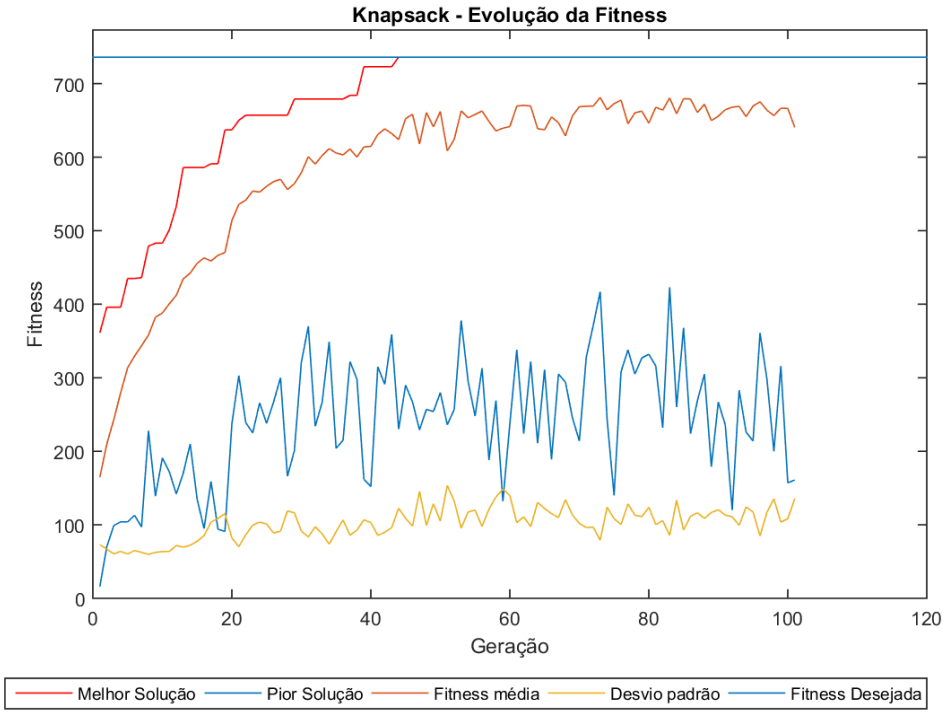


Figura 29: Gráfico da Mochila para 50 itens. Fonte: Autoria própria.

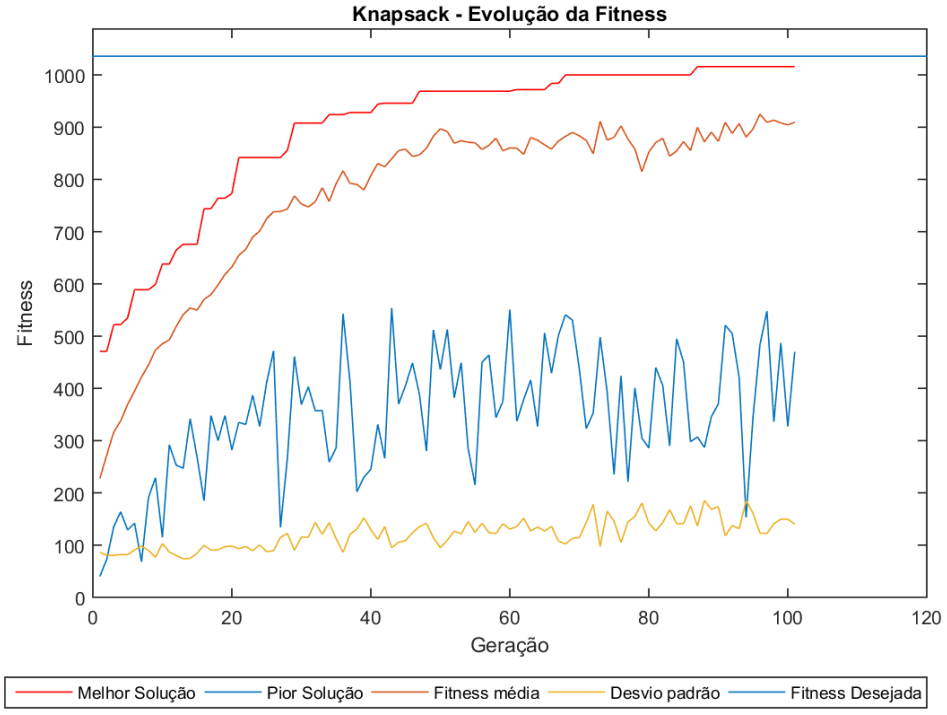


Figura 30: Gráfico da Mochila para 60 itens. Fonte: Autoria própria.

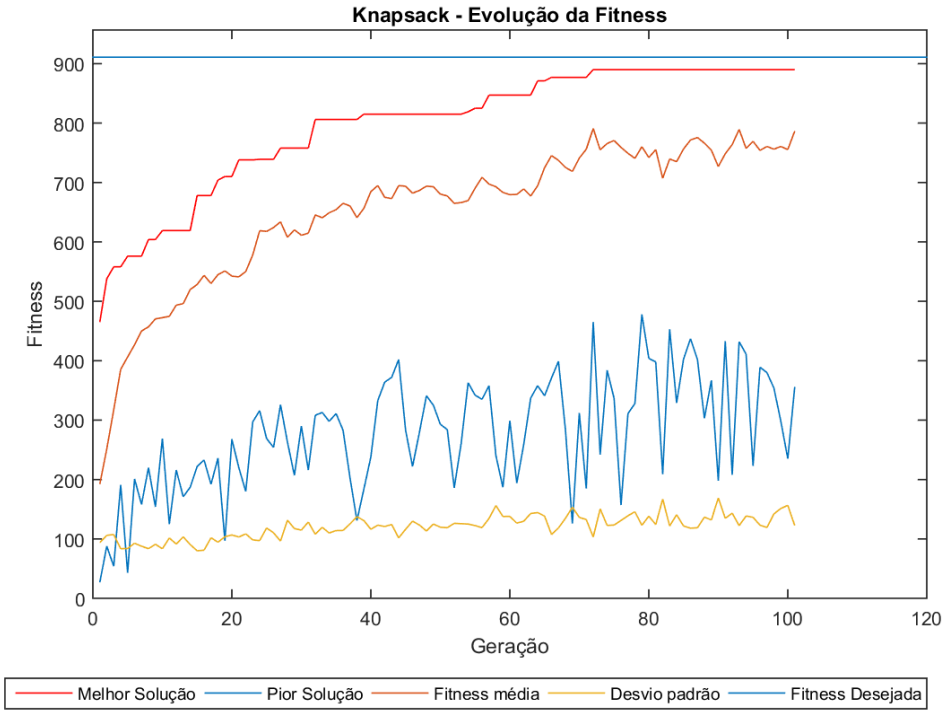


Figura 31: Gráfico da Mochila para 70 itens. Fonte: Autoria própria.

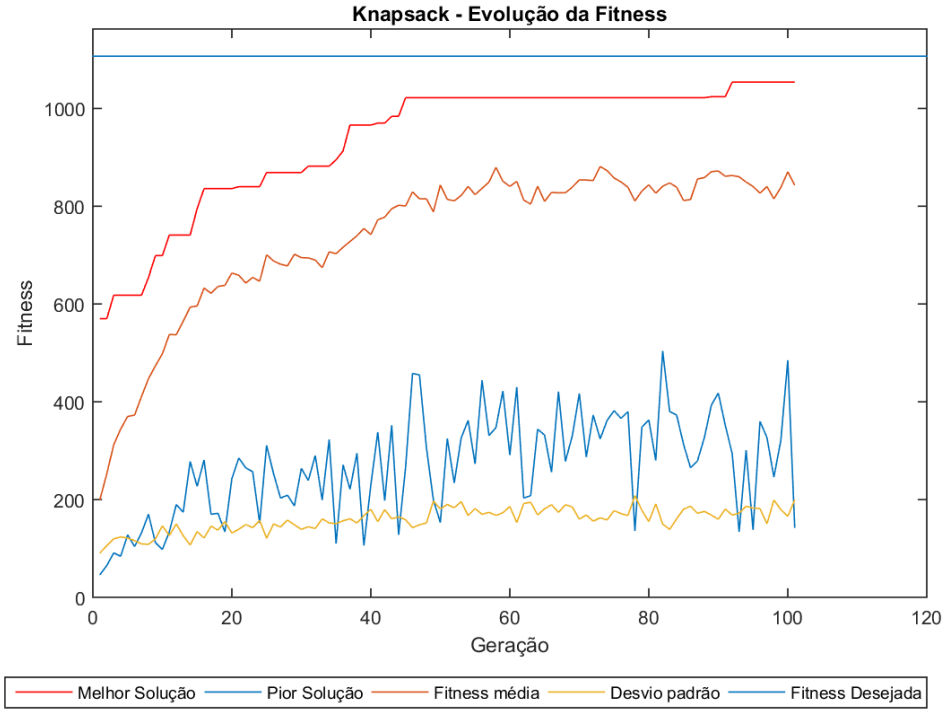


Figura 32: Gráfico da Mochila para 80 itens. Fonte: Autoria própria.

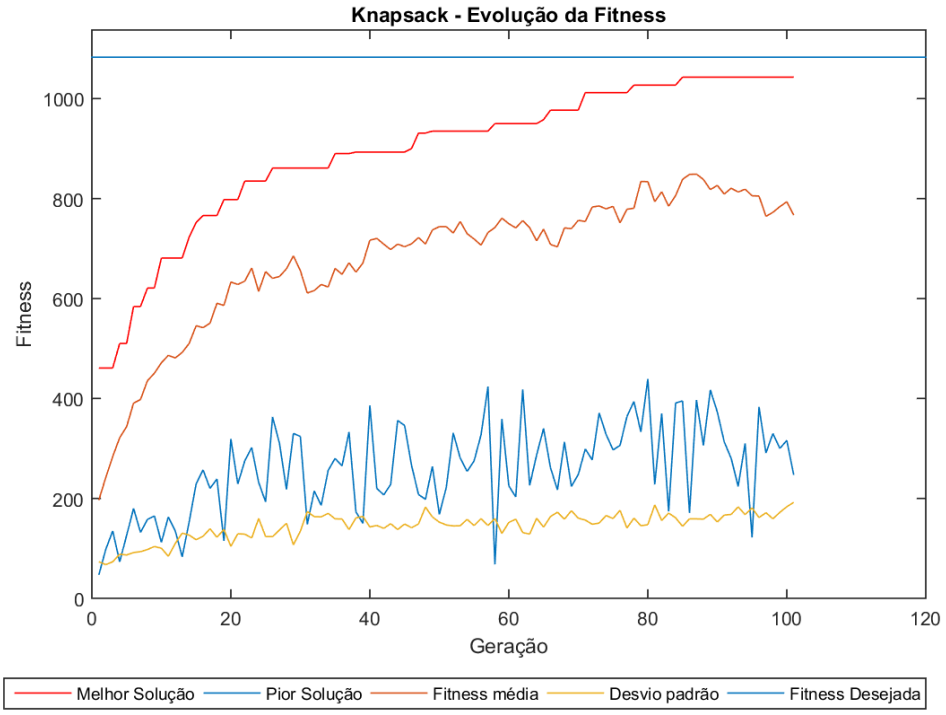


Figura 33: Gráfico da Mochila para 90 itens. Fonte: Autoria própria.

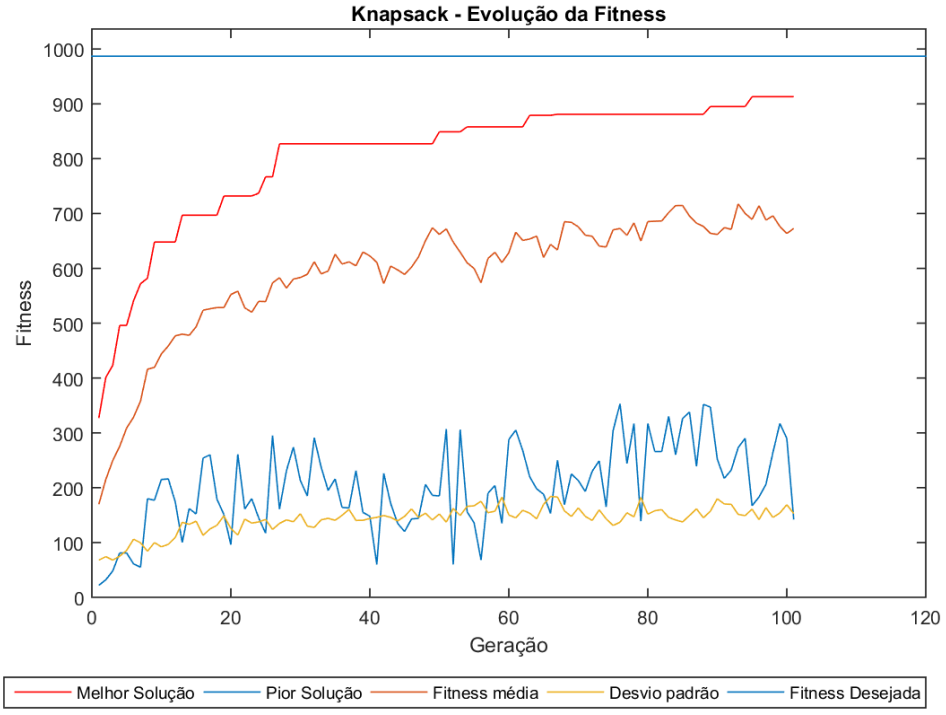


Figura 34: Gráfico da Mochila para 100 itens. Fonte: Autoria própria.

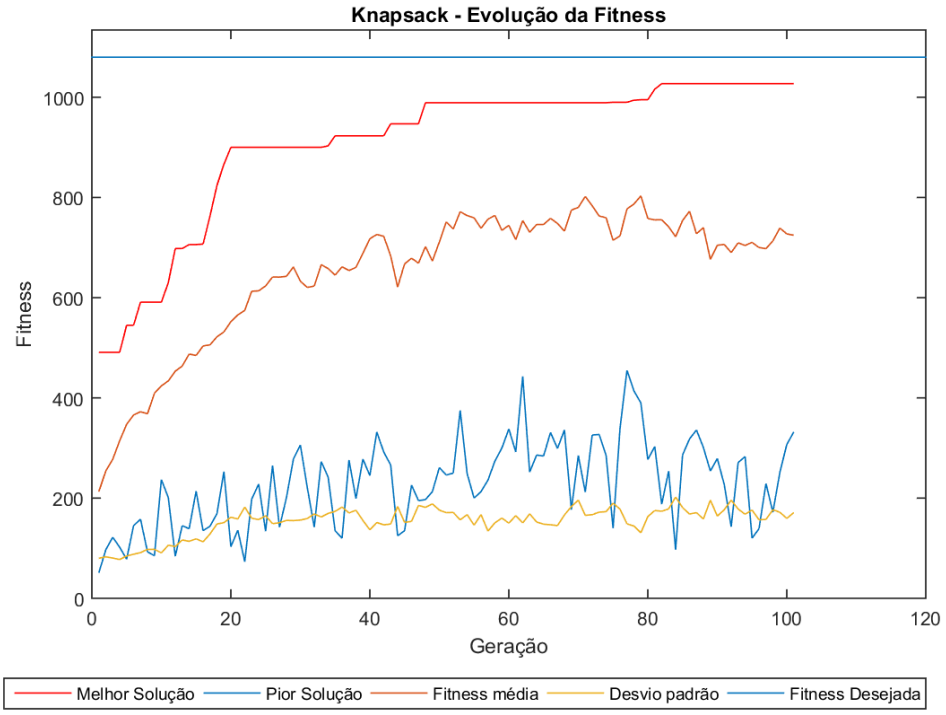


Figura 35: Gráfico da Mochila para 110 itens. Fonte: Autoria própria.

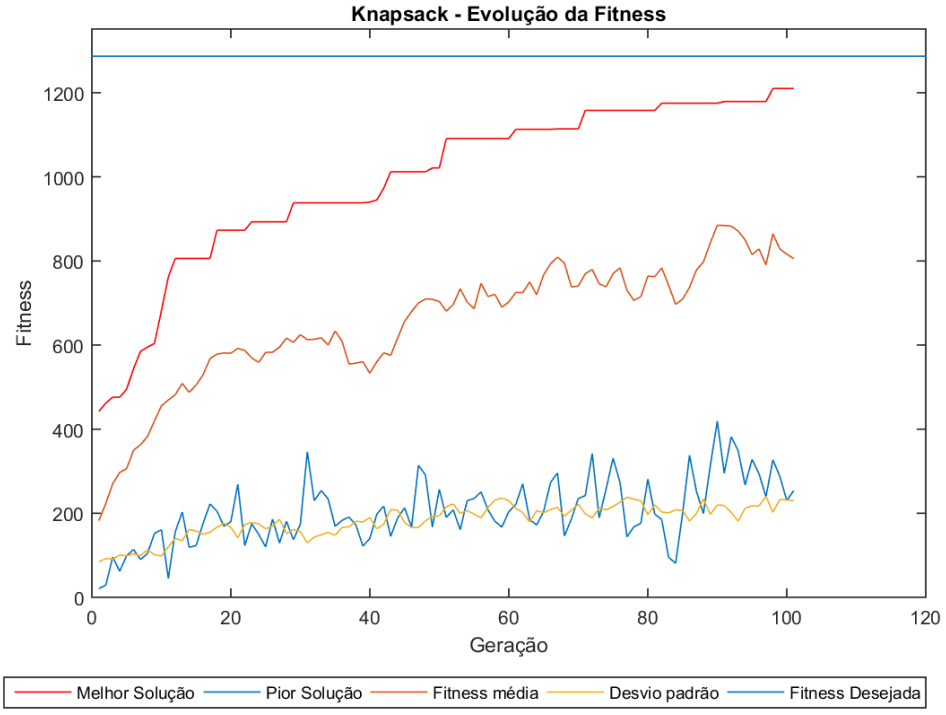


Figura 36: Gráfico da Mochila para 120 itens. Fonte: Autoria própria.

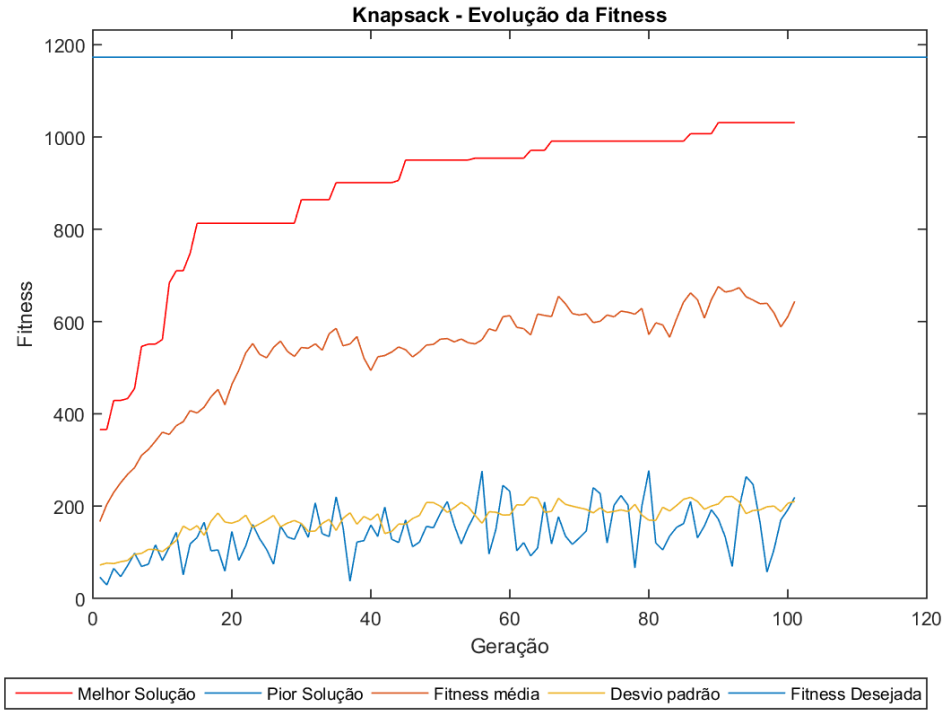


Figura 37: Gráfico da Mochila para 130 itens. Fonte: Autoria própria.

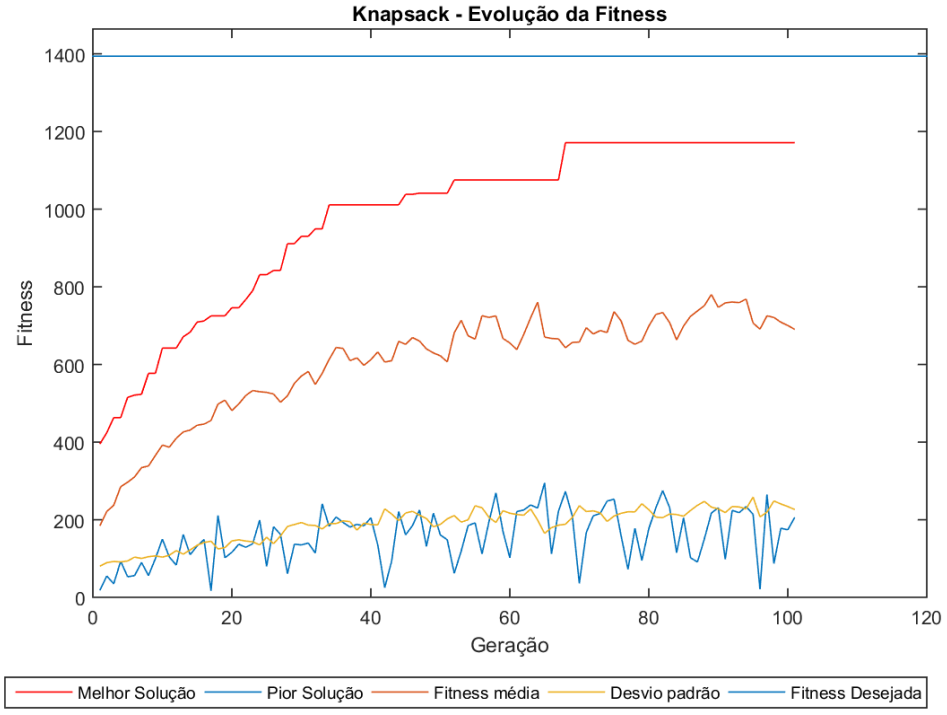


Figura 38: Gráfico da Mochila para 140 itens. Fonte: Autoria própria.

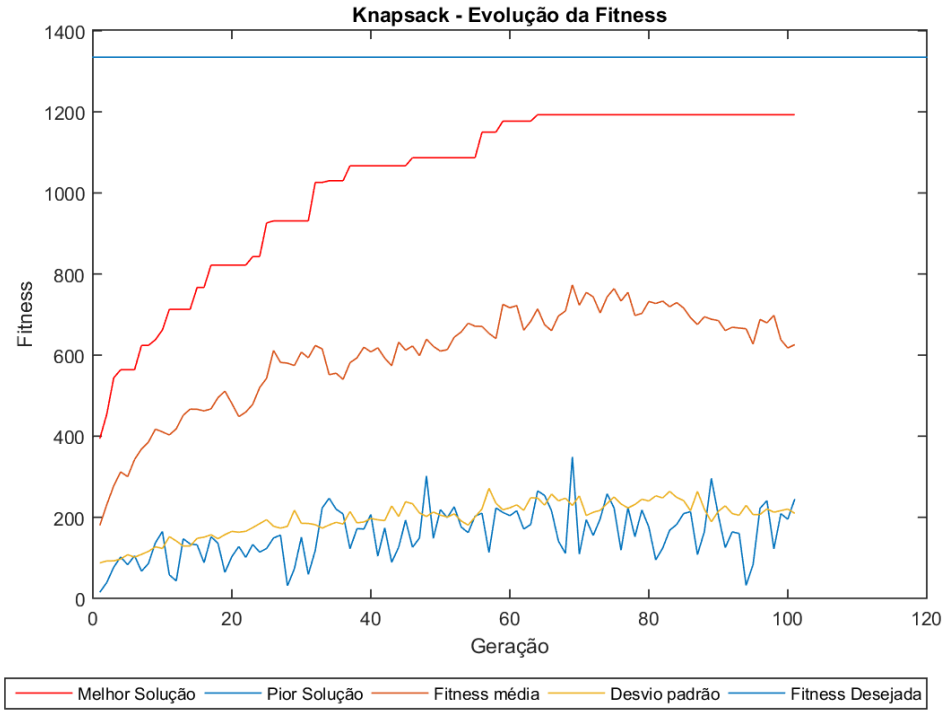


Figura 39: Gráfico da Mochila para 150 itens. Fonte: Autoria própria.

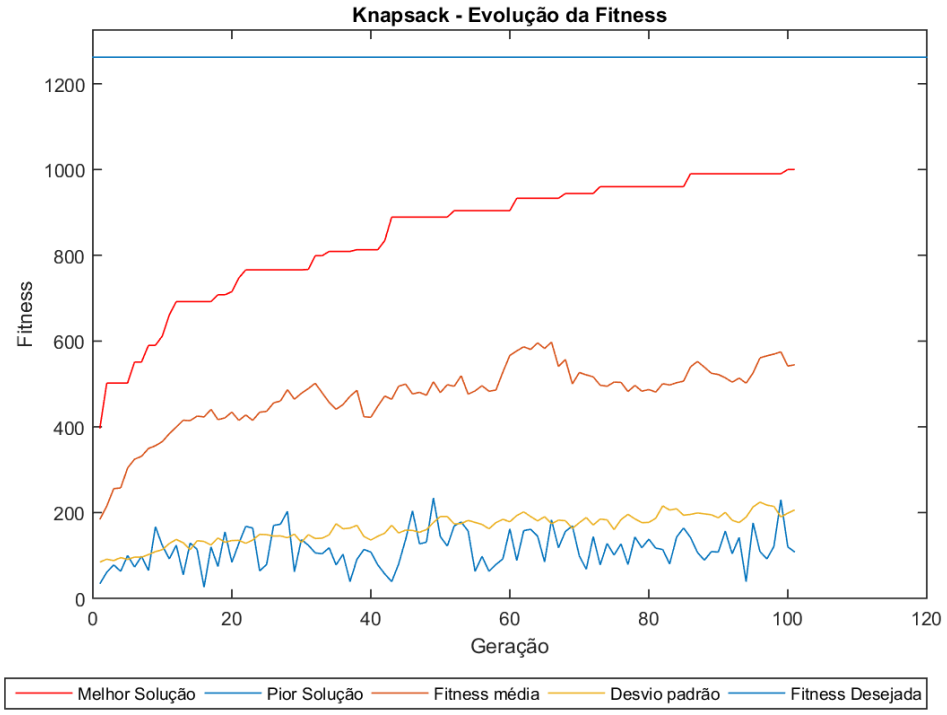


Figura 40: Gráfico da Mochila para 160 itens. Fonte: Autoria própria.

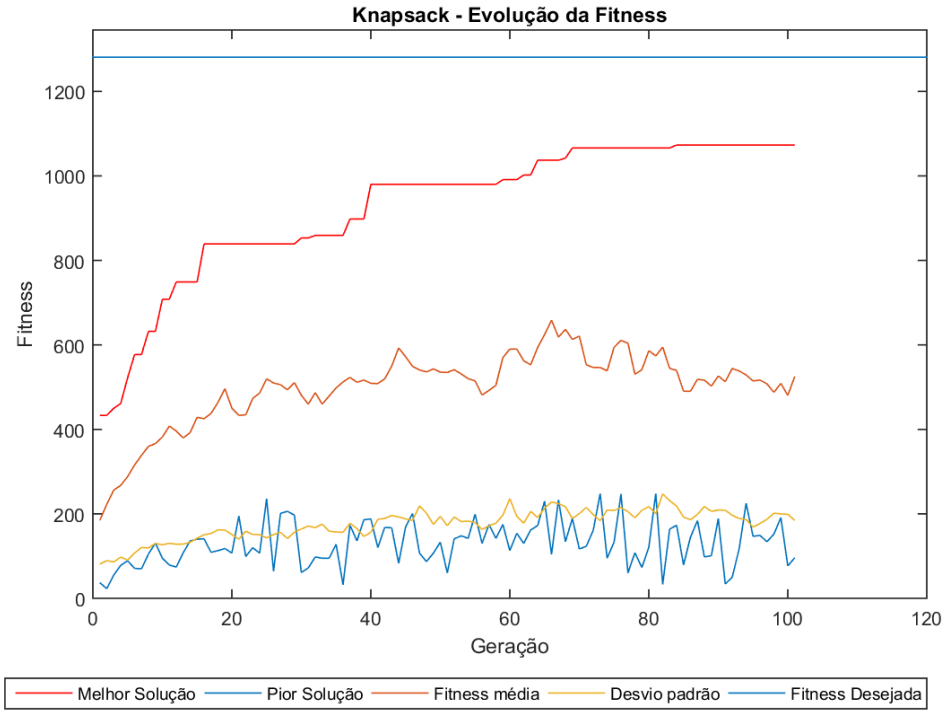


Figura 41: Gráfico da Mochila para 170 itens. Fonte: Autoria própria.

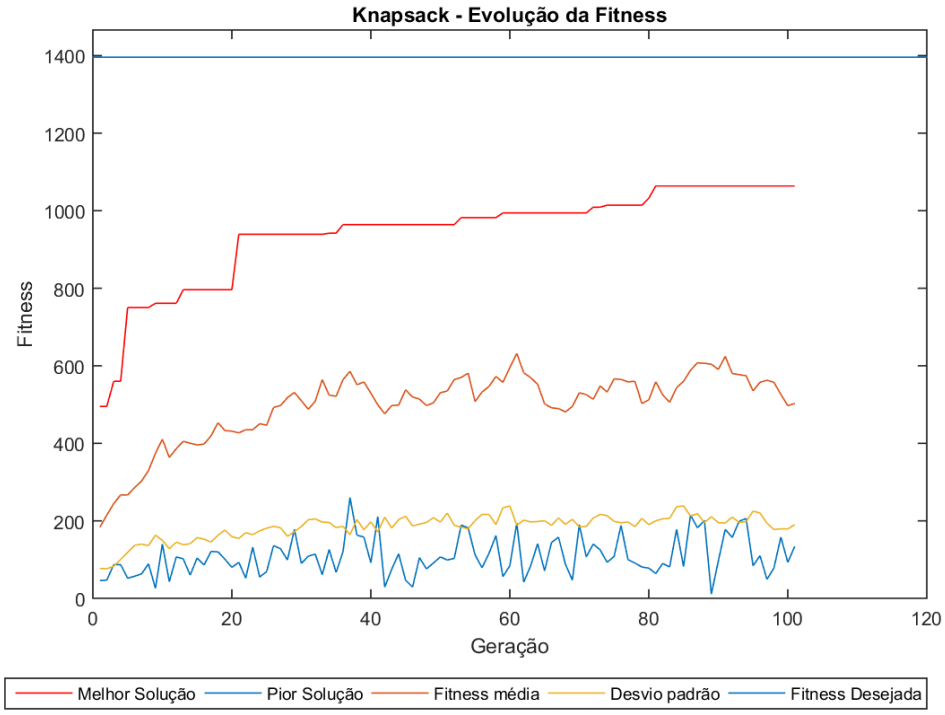


Figura 42: Gráfico da Mochila para 180 itens. Fonte: Autoria própria.

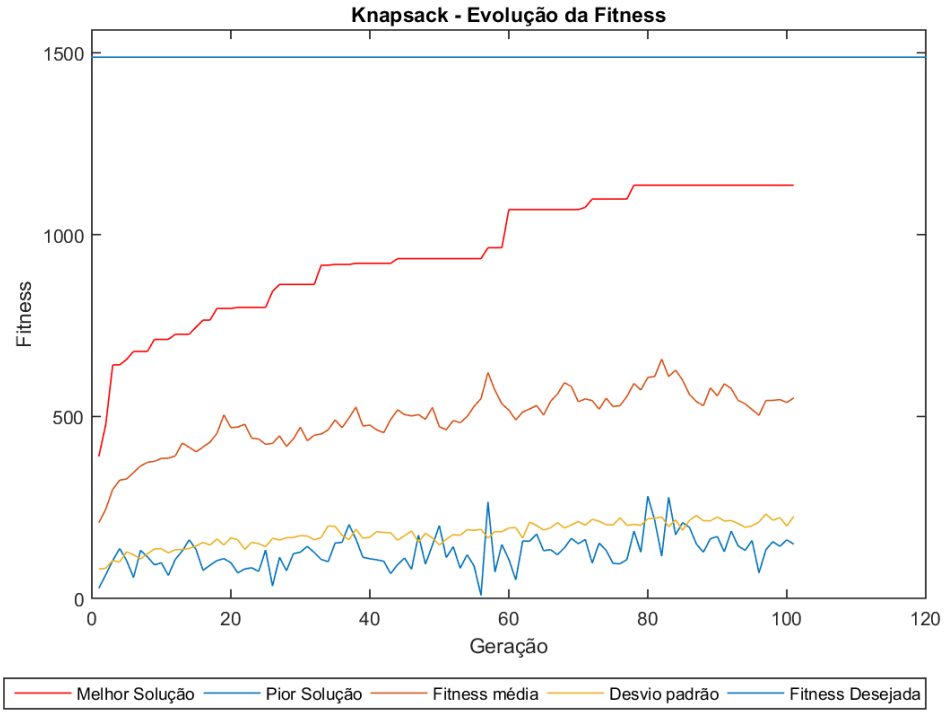


Figura 43: Gráfico da Mochila para 190 itens. Fonte: Autoria própria.

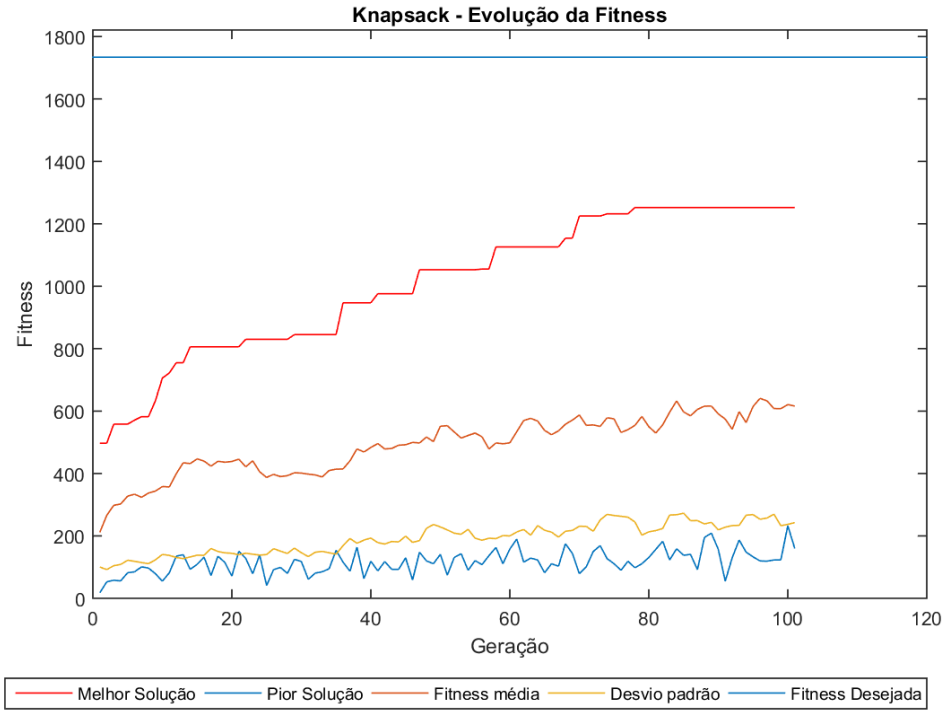
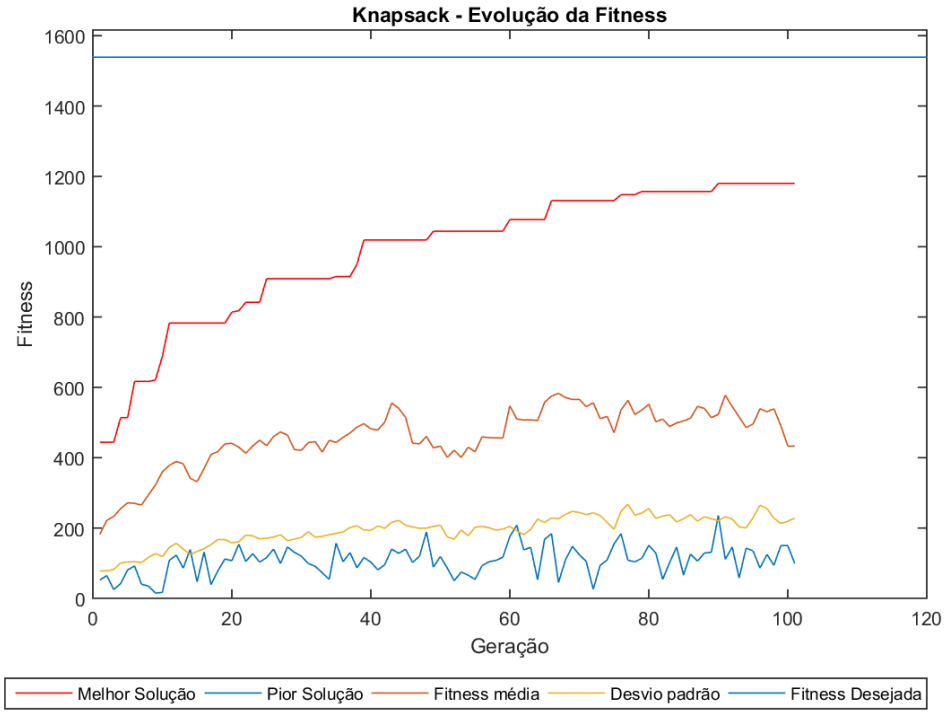


Figura 44: Gráfico da Mochila para 200 itens. Fonte: Autoria própria.



7 Referências Bibliográficas

Referências

[1] LEE, JACOBSON Disponível em: **http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5**

[2] MARQUES CUNHA, ADILSON Disponível em: **http://www.ele.ita.br/ flavios/listex2.htm**

[3] KUMAR, PANKAJ, Disponível em: **http://www.quora.com/How-do-I-solve-the-traveling-salesman-problem-using-dynamic-programming**

[4] Genetic Algorithms: Precursor to Genetic Programs.Disponível em:**http://www.stumptown.com/diss/chapter2.html**