

**UNIVERSIDADE FEDERAL DE MINAS GERAIS**

Escola de Engenharia

Engenharia Elétrica

**COMPUTAÇÃO EVOLUCIONÁRIA**

**Relatório 2 – Problema da Mochila**

Gabriel Saraiva Espeschit - 2015065541

19 de outubro de 2019

# **Introdução**

O objetivo do trabalho é escrever um algoritmo genético capaz de resolver o problema da mochila. O trabalho foi desenvolvido em Python. O problema da mochila foi expresso da seguinte maneira:

“*Dados N itens, onde cada item possui um benefício (vj) e um peso associado (wj), o problema consiste em selecionar o subconjunto de itens que maximiza a soma dos benefícios sem ultrapassar a capacidade (cap) da Mochila.*”

# **Metodologia**

O primeiro passo para resolver o problema proposto foi definir como representar as diferentes possibilidades de conteúdo que estaria dentro da mochila. Dada uma matriz, representado na figura abaixo, que define os valors e pesos dos objetos mochila, usou-se da representação binária para essa função.

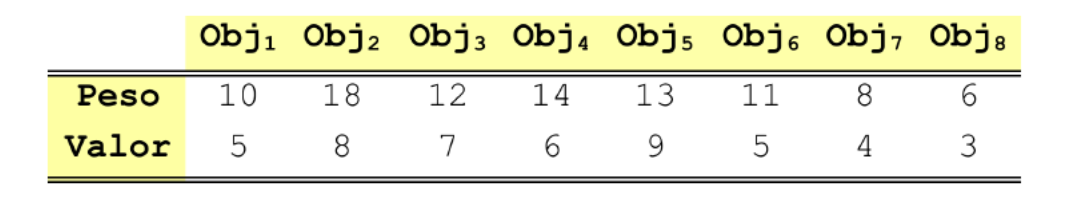


Figura 1 - Pesos e valores de cada objeto do problema

Como exemplo, tomamos o vetor para representação binária abaixo:

O valor dos objetos seria de 16 (9 + 4 + 3) e o seu peso de 27 (13 + 8 + 6).

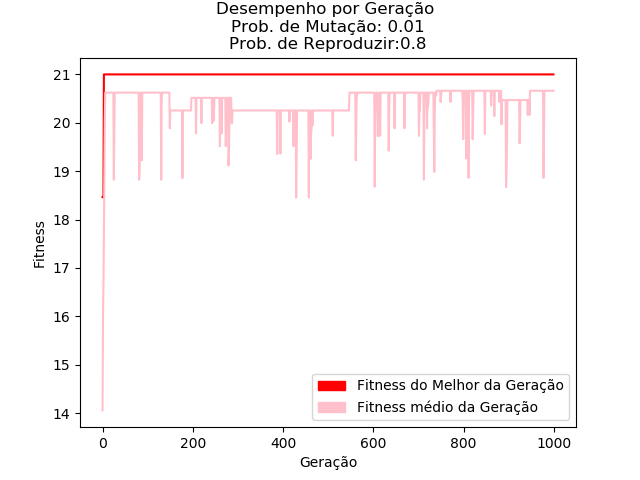
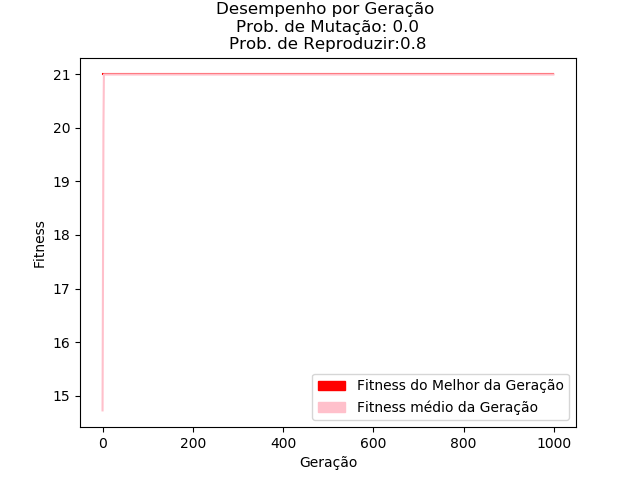
A partir disso, criou-se uma classe para representar os valores dos objetos, os pesos dos objetos, o número de objetos e a capacidade máxima suportada pela mochila. O algoritmo tem seu funcionamento conforme as seguintes etapas:

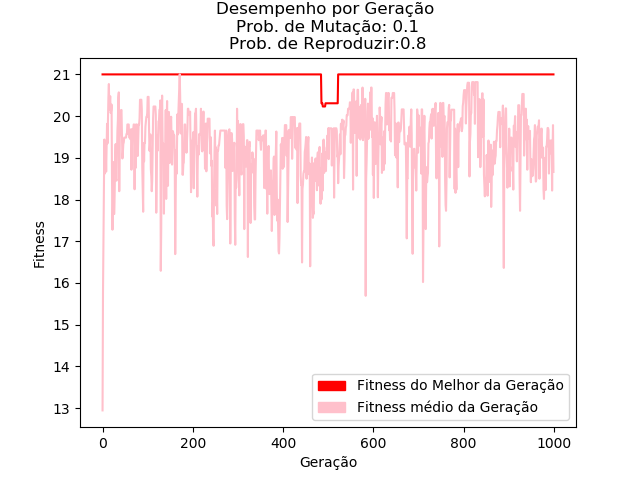
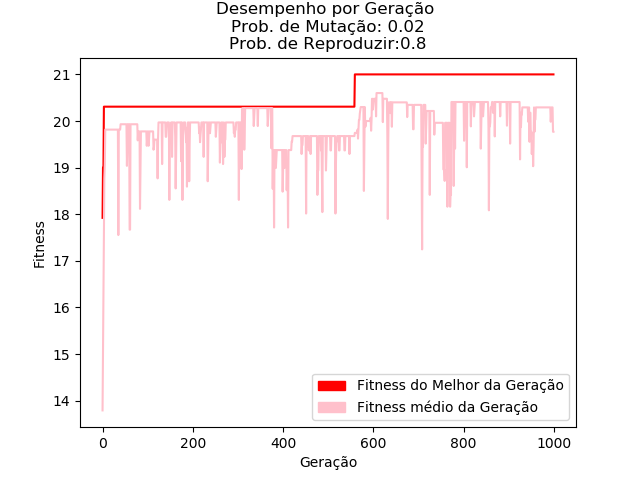
1. Inicialização: se cria uma matriz com representações binárias randômicas de dimensão N x *num\_p,* em que N é o número de objetos a serem selecionados (no caso do problema é 8) e *num\_p* é um número de indivíduos que pode ser definido na chamada da função (o arbitrário é de 10).
2. Avaliação do fitness: o fitness da população é avaliado com base em uma função que foi repassada para o aluno. Essa função valoriza os membros com valores mais altos, penalizando-os caso excedam o limite de carga da mochila.
3. Iteração: no processo de iteração se faz algumas ações valem destaque:
   1. Seleciona-se 2 pais com base em seus fitness utilizando o método da roleta. Em seguida, cria-se uma população de *num\_p* filhos. O cruzamento pode ser realizado com base em uma probabilidade *prob*\_*def* (o número padrão é de 0.8) dos 2 pais selecionados, e é um crossover simples. Caso não haja cruzamento, os filhos são cópias dos pais.
   2. Os filhos sofrem mutações com base em uma probabilidade *prob\_mut* (o padrão é de 0.02)
   3. Os filhos gerados na matriz são comparados com os indivíduos da população atual. Caso o fitness do enésimo filho seja melhor que o fitness do enésimo individuo atual, o filho toma seu lugar. Esse método de seleção pode ser considerado extremamente elitista, mas provou-se muito bom para garantir a rápida convergência do algoritmo, como vamos verificar abaixo.
   4. Por fim é feita uma avaliação da nova população que irá compor os pais da próxima geração. O fitness do melhor membro e o fitness médio é preservado.
4. É plotado um gráfico contendo o fitness médio e o melhor fitness de cada geração, o melhor individuo encontrado, após rodar o algoritmo diversas vezes, foi o:

Com um valor de 21 e um peso de 35.

1. **Resultados**

Experimentou-se com diferentes valores de *prob\_def* (probabilidade de ocorrer um cruzamento) e *prob\_mut* (probabilidade de ocorrer uma mutação). Os resultados seguem abaixo:





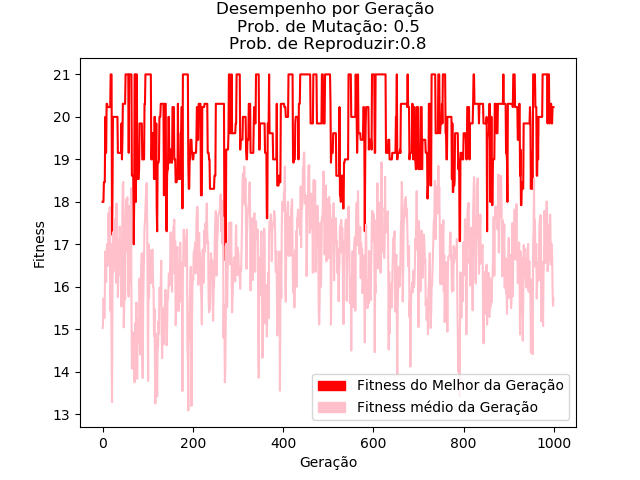
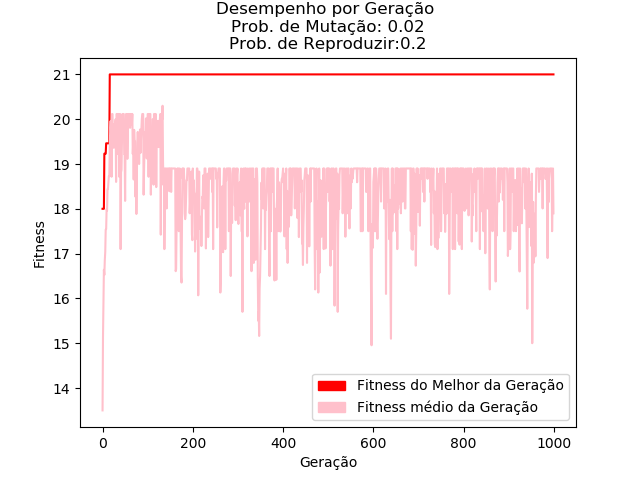
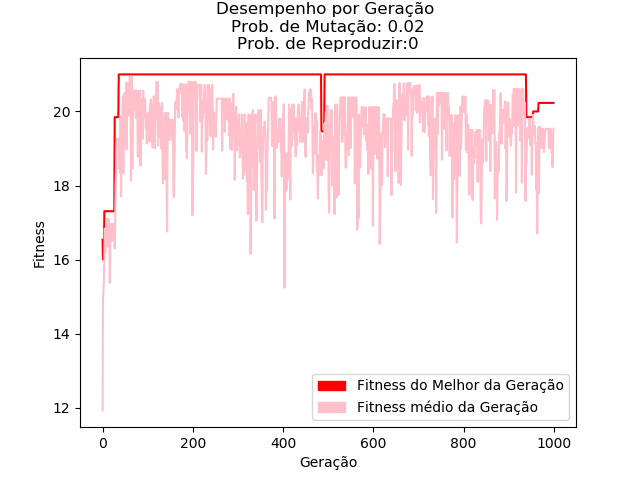
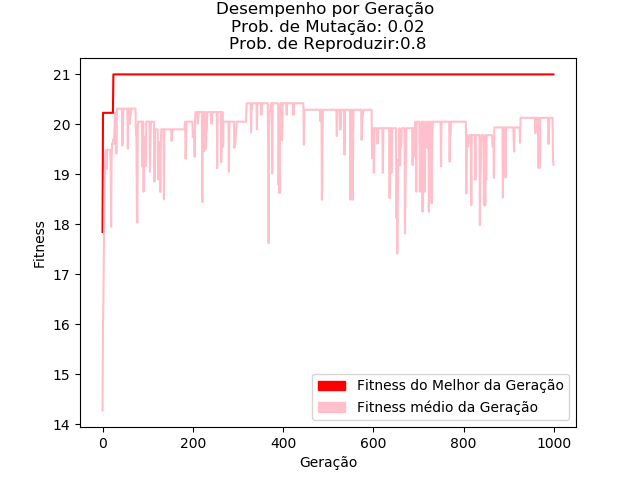
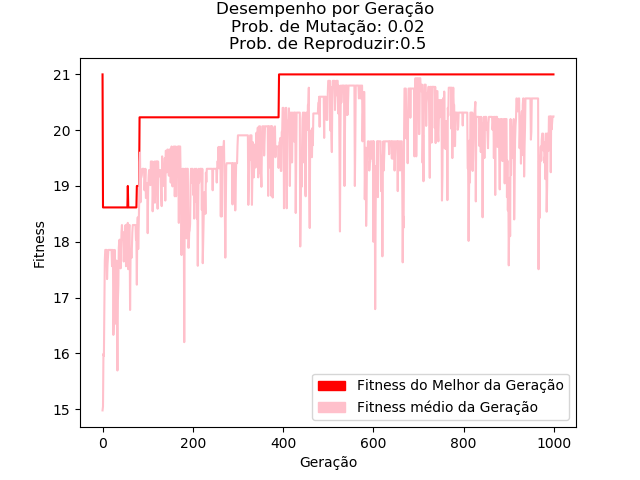


Figura 2 - Manteve-se a probabilidade de reprodução constante em 0.8 e alterou-se a probabilidade de mutação. Podemos ver que em: a) prob\_mut = 0. Atinge o resultado ótimo e tanto a média quanto o melhor individuo ficam constantes, não há oscilações; b) prob\_mut = 0.01. As oscilações como um todo são controladas e a média tende á solução ótima; c) prob\_mut =0.02. As mutações fazem com que a média oscile mais, no entanto, esse gráfico exemplifica a importância de usar mutações em um algoritmo evolucionário, pois, caso não houvesse mutações, é muito provável que cairíamos em um máximo local e não chegaríamos ao resultado ideal, como pode ser visto pela longa reta no melhor indivíduo que durou até a geração 600, aproximadamente; d) prob\_mut = 0.1. As oscilações na média estão mais descontroladas e chegam a piorar o melhor candidato da posição ideal, brevemente; e) prob\_mut = 0.5. As mutações causam com que o algoritmo entre um estado estocástico, oscilações são vistas tanto na média quanto na curva de melhor individuo e o algoritmo, nessa situação, não converge.





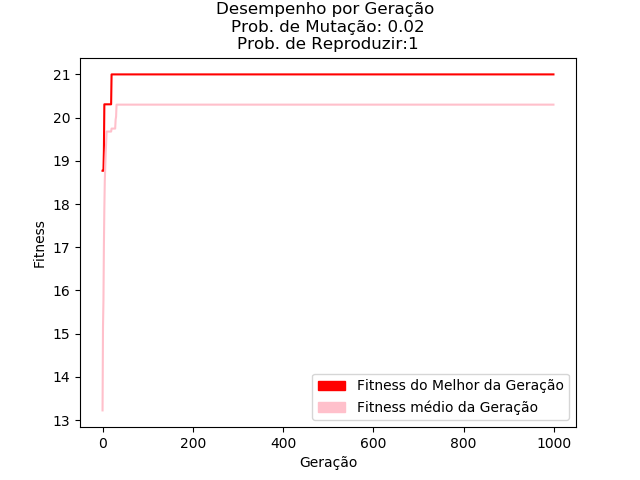


Figura 3 - Manteve-se a probabilidade de mutação constante em 0.02 e alterou-se a probabilidade de reprodução. Podemos ver que em: a) prob\_def = 0. Apesar de não ocorrer reprodução e todo filho ser uma cópia do pai, as mutações causam com que cheguemos à solução ideal. No entanto, como o processo é estocástico e não há uma forma de preservar essa melhor solução ao longo das gerações de forma efetiva, o melhor individuo logo é deposto por um que tem um fitness pior que o dele; b,c e d) prob\_def = 0.2, 0.5 e 0.8. Nessas situações o algoritmo apresentou um comportamento semelhante, a média chegava em um pico, antes de alguns indivíduos sofrerem mutações que o tornassem menos aptos abaixando o valor. Esses indivíduos eram rapidamente repostos por indivíduos melhores que fazia com que a média voltasse ao seu pico anterior. Nesses casos, os algoritmos sempre convergiam; e) podemos reparar uma situação peculiar aqui: a média para de oscilar. Para o leitor atento, isso é um resultado de nossa função de substituição de filhos. Uma vez que toda a população está infestada de indivíduos cujos filhos são tão aptos quantos os pais, os pais não serão substituídos, e nossa média ficará constante.

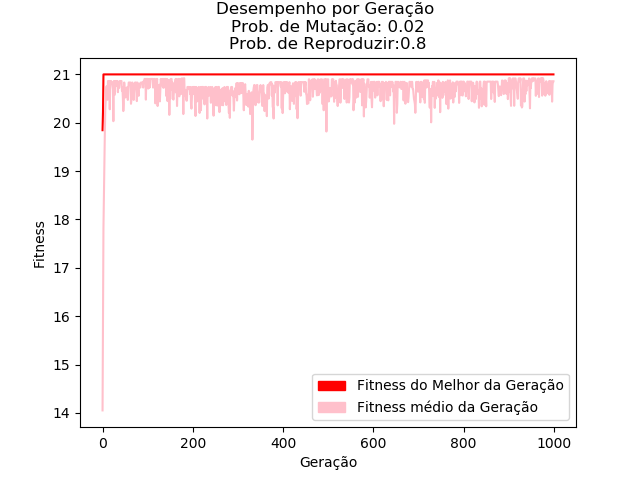


Figura 4 - Aqui usamos os valores padrões para probabilidade de mutação e reprodução, porém aumentamos o tamanho da população de 10 para 50 indivíduos. Podemos ver que a média fica muito mais compacta e próxima ao indivíduo ótimo, o que mostra que nosso algoritmo converge corretamente, conforme esperado

1. **Código**

O código desenvolvido em Python pode ser visto abaixo:

*import* numpy *as* np  
*import* matplotlib.pyplot *as* plt  
*import* matplotlib.patches *as* mpatches  
  
*# Classe Mochila para definir os parâmetros de inicialização e ficar fácil de mudar posteriormente  
class* Mochila:  
 N = 8 *# Número de Objetos* cap = 35 *# Capacidade* wj = [10, 18, 12, 14, 13, 11, 8, 6] *# Peso* vj = [5, 8, 7, 6, 9, 5, 4, 3] *# Valor  
  
# Função para calcular o fitness de cada linha da matriz população  
def* fitness(solution, values, weights, cap):  
 rho = *max*(np.divide(values, weights))  
 total\_benefit = np.dot(solution, values)  
 total\_weight = np.dot(solution, weights)  
 *if* total\_weight > cap:  
 f = total\_benefit - rho \* (total\_weight - cap)  
 *else*:  
 f = total\_benefit  
 *return* f  
  
*# Função para selecionar os pais usando o método da roleta  
def* sel\_pais(prob, num\_pais=2):  
 current = 1  
 a = []  
 parent\_index = []  
  
 *for* i *in range*(*len*(prob)):  
 *if* i == 0:  
 a.append(prob[i])  
 *else*:  
 a.append(prob[i]+a[i-1])  
  
 *while* current <= num\_pais:  
 r = np.random.rand(1)  
 i = 0  
 *while* a[i] < r:  
 i += 1  
 parent\_index.append(i)  
 current += 1  
 *return* parent\_index  
  
*# Função para fazer um cruzamento simples entre 2 indivíduos da população  
def* cruzamento(p1, p2):  
 n = *len*(p1)  
 pos = np.random.randint(1, n)  
 f1 = np.concatenate([p1[0:pos], p2[pos:]])  
 f2 = np.concatenate([p2[0:pos], p1[pos:]])  
  
 *return* f1, f2  
  
*# Função principal de execução do programa  
def* mochila(it=1000, num\_p=10, prob\_def=0.8, prob\_mut=0.02):  
 *# Inicialização do algoritmo* iteracao = 0  
 melhor = []  
 media = []  
 m1 = Mochila  
 *# Inicializando a população* solution = np.random.randint(2, size=(num\_p, m1.N))  
 *# Processo de iteração  
 while* iteracao < it:  
 *# Calculando o fitness* fit = []  
 *for* i *in range*(num\_p):  
 fit.append(fitness(solution[i], m1.vj, m1.wj, m1.cap))  
 fit\_prob = [(fit[i]/*sum*(fit)) *for* i *in range*(*len*(fit))]  
 filhos = np.zeros((num\_p, m1.N))  
 aux\_cruz = 0  
 *while* aux\_cruz < num\_p-1:  
 *# Selecionando dois pais usando o metodo da roleta* pais = sel\_pais(fit\_prob)  
 *# Fazendo o cruzamento* prob\_cross = np.random.rand(1)  
 *if* prob\_cross < prob\_def:  
 f1, f2 = cruzamento(solution[pais[0]], solution[pais[1]])  
 *else*:  
 f1, f2 = solution[pais[0]], solution[pais[1]]  
 *# Fazendo mutações aos filhos  
 for* i *in range*(m1.N):  
 prob\_m = np.random.rand(1)  
 *if* prob\_m < prob\_mut:  
 f1[i] = *int*(*not*(f1[i]))  
 f2[i] = *int*(*not*(f2[i]))  
 filhos[aux\_cruz] = f1  
 filhos[aux\_cruz+1] = f2  
 aux\_cruz += 2  
 *# Acrescentando os filhos a população  
 # Caso o fitness do íesimo filho for maior que o do íesimo membro, substitua-o  
 for* n *in range*(num\_p-1):  
 *if* fitness(filhos[n], m1.vj, m1.wj, m1.cap) > fit[n]:  
 solution[n] = filhos[n]  
  
 *# Avaliando a população* melhor.append(*sorted*(fit)[-1])  
 media.append(*sum*(fit)/*len*(fit))  
 *print*(f'Melhor: {*sorted*(fit)[-1]}\n Média:{*sum*(fit)/*len*(fit)} Geração: {iteracao}')  
 iteracao += 1  
 *print*(solution[fit.index(*sorted*(fit)[-1])])  
 plt.plot(melhor, c='r')  
 plt.plot(media, c='pink')  
 green\_patch = mpatches.Patch(color='r', label='Fitness do Melhor da Geração')  
 blue\_patch = mpatches.Patch(color='pink', label='Fitness médio da Geração')  
 plt.legend(handles=[green\_patch, blue\_patch])  
 *# plt.plot(pior, c='r')* plt.title(f'Desempenho por Geração \nProb. de Mutação: {prob\_mut}\nProb. de Reproduzir:{prob\_def}')  
 plt.xlabel('Geração')  
 plt.ylabel('Fitness')  
 plt.show()  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 mochila(it=1000, num\_p=10, prob\_def=0.8, prob\_mut=0)