ESCOLA SUPERIOR DE PLANEJAMENTO E GESTÃO - ESPG

YGOR WESLEY SOARES MOREIRA LIMA

APLICAÇÃO DE ALGORITMOS GENÉTICOS PARA A SOLUÇÃO DO PROBLEMA DE CARREGAMENTO DE CAMINHÕES

Brasília/DF

2022

ESCOLA SUPERIOR DE PLANEJAMENTO E GESTÃO - ESPG

YGOR WESLEY SOARES MOREIRA LIMA

Mat.: 5020218452

APLICAÇÃO DE ALGORITMOS GENÉTICOS PARA A SOLUÇÃO DO PROBLEMA DE CARREGAMENTO DE CAMINHÕES

Trabalho de Conclusão de Curso apresentado à Escola Superior de Planejamento e Gestão - ESPG como requisito para aprovação no curso de Pós-Graduação em Engenharia de Software.

Brasília/DF

2022

Nome: Ygor Wesley Soares Moreira Lima
Mat.: 5020218452

Aplicação de Algoritmos Genéticos para a Solução do Problema de Carregamento de Caminhões

Artigo apresentado à Escola Superior de Planejamento e Gestão como requisito para aprovação no curso de Pós-graduação em Engenharia de Software.

Nota: _____ Data: __/__/_____

_____

Prof. _____
Prof. _____
Prof. _____

APLICAÇÃO DE ALGORITMOS GENÉTICOS PARA A SOLUÇÃO DO PROBLEMA DE CARREGAMENTO DE CAMINHÕES

Ygor Wesley Soares Moreira Lima, ESPG, contato@ygorml.org

RESUMO

Este artigo propõe uma solução para o problema de otimização de carregamento de caminhões utilizando uma abordagem de algoritmos genéticos. A partir de uma pesquisa exploratória, aplicada e bibliográfica foi possível o desenvolvimento de um algoritmo capaz de otimizar o carregamento, por meio da modelagem de unidades de carga. Obteve-se uma solução ótima com 400 gerações.

Palavras-chave: Algoritmos Genéticos, Logística, Inteligência Artificial

ABSTRACT

This paper proposes a solution to the truck loading optimization problem using a genetic algorithm approach. From an exploratory, applied and bibliographic research, it was possible to develop an algorithm capable of optimizing the load, through the modeling of load units. An optimal solution was obtained with 400 generations.

Keywords: Genetic Algorithms, Logistics, Artificial Intelligence

1 INTRODUÇÃO

O presente artigo propõe uma solução para o problema de otimização de carregamento de caminhões utilizando uma abordagem de algoritmos genéticos. Este artigo classifica-se como uma pesquisa exploratória, aplicada e bibliográfica. Como objetivos específicos do trabalho, tem-se a necessidade de modelar o carga de um caminhão, além de desenvolver o algoritmo. Como objetivo geral, otimizar o carregamento de um caminhão, considerando o limite de carga e itens.

Na seção "Algoritmos Genéticos" será exposto um breve histórico sobre a Teoria da Evolução e o surgimento dos Algoritmos Genéticos como campo de estudo da Ciência da Computação, além de aspectos relevantes para a compreensão do assunto.

Na seção "Definição do Problema de Logística", será apresentada a pergunta-problema que guiou o desenvolvimento do trabalho.

Na seção "Desenvolvimento do Algoritmo", serão expostos os princípios norteadores do desenvolvimento do Algoritmo Genético voltado à solução do problema proposto.

Na seção "Resultados", serão expostos os resultados encontrados através da execução do algoritmo genético proposto.

Por fim, na seção "Considerações Finais", será apresentada uma breve reflexão sobre o trabalho desenvolvido.

2 ALGORITMOS GENÉTICOS

Através da observação da natureza, Charles Darwin desenvolveu sua teoria sobre a origem e evolução das espécies na Terra. A publicação do livro "A origem das Espécies" em 1859 foi o início da consolidação da Teoria Evolutiva, não só no meio acadêmico, mas também na sociedade. (SIVANANDAM, 2008).

Extrapolando os conceitos da Teoria Evolutiva de Charles Darwin para o mundo da Ciência da Computação, surgiu o campo denominado de Computação Evolucionária, que se utiliza de diversos axiomas e ideias de Darwin (SIVANANDAM, 2008).

Ainda sob o contexto da Computação Evolucionária, os Algoritmos Genéticos são métodos de otimização e busca inspirados nos mecanismos de evolução de populações de seres vivos (LACERDA, 1999).

Entende-se por otimização como a busca da melhor solução para um dado problema consistindo em tentar várias soluções e utilizar a informação obtida neste processo de forma a encontrar soluções cada vez melhores. (LACERDA, 1999).

Quando se soluciona um problema com Algoritmos Genéticos, ao invés de se buscar por uma solução específica, definimos um conjunto de características para o conjunto possível de soluções. (SHEPPARD, 2016)

Sabe-se que a malha rodoviária federal do Brasil possui atualmente extensão total de 75,8 mil km, dos quais 65,4 mil km correspondem a rodovias pavimentadas e 10,4 mil km correspondem a rodovias não pavimentadas. (BRASIL, 2019)

Um dos problemas enfrentados por empresas responsáveis por fretes rodoviários é a otimização do carregamento de caminhões, muitas vezes buscando obter um maior valor de carga transportada, com o menor espaço possível. Trata-se, portanto, de um problema de otimização.

Conforme Pereira (2020), dentro do contexto de algoritmos genéticos, os indivíduos mais adaptados ao seu ambiente terão maior probabilidade de procriar, sendo este perfil de seleção chamado de elitista.

Para Pereira (2020), são consideradas como etapas dos Algoritmos Genéticos: Criação de uma população inicial, determinação dos indivíduos mais adaptados, cruzamento e procriação e por último a mutação.

Lacerda (1999) nos ensina que o primeiro passo de um Algoritmo Genético típico é a geração de uma população inicial de cromossomos, que é formada por um conjunto aleatório de cromossomos que representam possíveis soluções do problema a ser resolvido. Durante o processo evolutivo, esta população é avaliada e cada cromossomo recebe uma nota (denominada de aptidão no jargão da literatura de AGs), refletindo a qualidade da solução que ele representa.

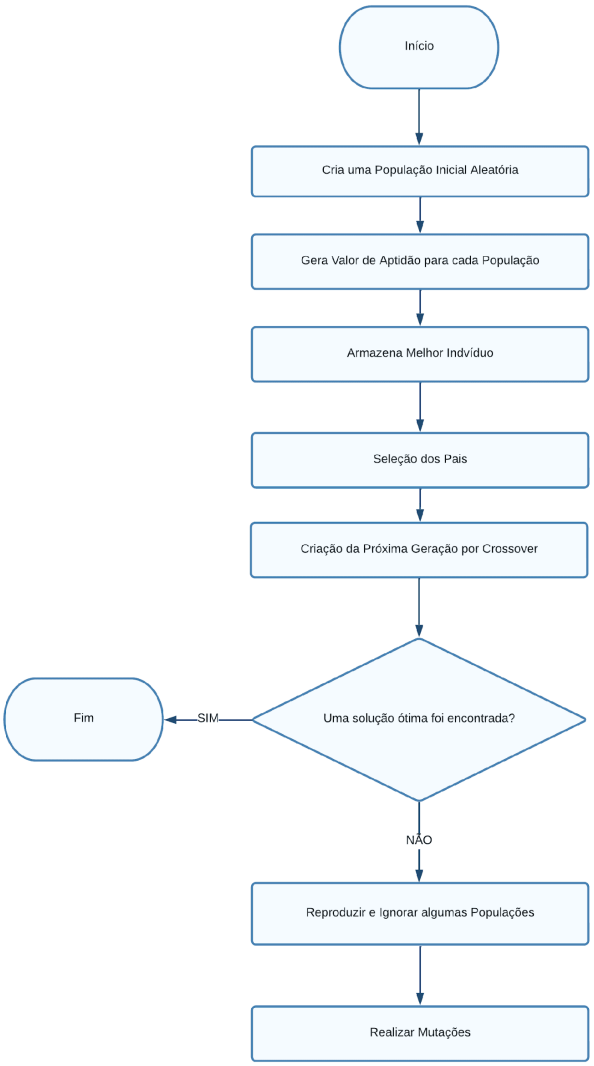
Em geral, os cromossomos mais aptos são selecionados e os menos aptos são descartados. Os membros selecionados podem sofrer modificações em suas características fundamentais através dos operadores de crossover e mutação, gerando descendentes para a próxima geração. Este processo é repetido até que uma solução satisfatória seja encontrada.

Para Lacerda (1999), um AG processa populações de cromossomos, sendo que este é uma estrutura de dados, geralmente vetor ou cadeia de bits. Este último, por sua vez, é a estrutura mais tradicional. Em geral, um cromossomo representa um conjunto de parâmetros da função objetivo cuja resposta será maximizada ou minimizada.

Conforme Lacerda (1999), um AG começa com uma população inicial de N cromossomos. Os cromossomos deverão ser gerados aleatoriamente no caso de inexistência de conhecimento prévio sobre a região do espaço de busca onde se encontra a solução do problema.

A figura 1 ilustra o fluxograma do funcionamento de um Algoritmo Genético.

Figura 1 – Fluxograma de um Algoritmo Genético



Fonte: Adaptado de SIVANANDAM; DEEPA (2008, p. 32)

3 DEFINIÇÃO DO PROBLEMA DE LOGÍSTICA

Para a correta modelagem do problema, definiu-se como pergunta-problema: Como maximizar o valor da carga de um caminhão, utilizando todo o espaço disponível, sabendo que é possível carregar apenas 20 produtos?

Tendo-se a pergunta-problema em mente, há ainda a necessidade da modelagem de cada unidade de carga. Para isso, os objetos foram considerados através de seu nome, espaço utilizado (em metros cúbicos) e valor de carga, conforme tabela 1.

Tabela 1 – Espaço Utilizado e Valor de Produtos

Nome/Descrição	**Espaço Utilizado**	**Valor (R$)**
Geladeira Dako	0,751 m^3	999,90
Notebook Dell	0,0035 m^3	2.499,90
Microondas Panasonic	0,0319 m^3	299,29
Notebook Asus	0,527 m^3	3.999,00
iPhone X	0,0000899 m^3	2.199,12
iPhone 13 Pro	0,0000899 m^3	10.911,12
Ventilador Panasonic	0,496 m^3	199,90
Geladeira Brastemp	0,635 m^3	849,00
TV 55”	0,400 m^3	4.346,99
TV 50”	0,290 m^3	3.999,90
Microondas Electrolux	0,0424 m^3	308,66
Geladeira Consul	0,870 m^3	1.199,89
TV 42”	0,200 m^3	2.999,90
Microondas LG	0,0544 m^3	429,90
Notebook Lenovo	0,498 m^3	1.999,90

Fonte: Própria (2022).

Além disso, a capacidade máxima de carregamento do caminhão será considerada como 3m^3. Outro dado importante a ser observado é o somatório dos volumes dos produtos da tabela, que é 4.79m^3.

4 DESENVOLVIMENTO DO ALGORITMO

Nesta etapa do trabalho, optou-se por seguir uma abordagem orientada à objetos para o desenvolvimento de código, de forma a maximizar a reutilização de código em trabalhos futuros, além de atender às boas práticas de Engenharia de Software.

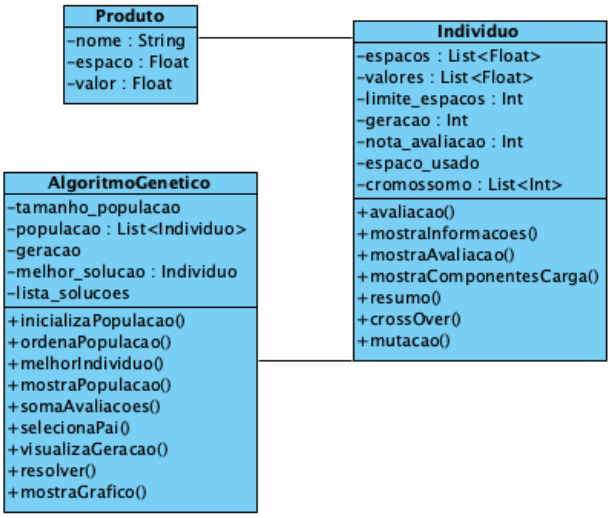
Cada produto a ser carregado no caminhão será tratado como um objeto da classe Produto, que possuirá como atributos "nome", "espaço" e "valor".

Da mesma forma, cada possível solução será tratada como um objeto da classe Individuo, que possui como atributos "espaços", "valores", "limite de espaços", "geração", "nota de avaliação", "espaço utilizado" e "cromossomo".

Por fim, o Algoritmo Genético *per si* também será tratado como um objeto, possuindo como atributos "tamanho da população", "população", "geração", "melhor solução" e, por fim, "lista de soluções".

Pode-se observar o Diagrama de Classes na Figura 2.

Figura 2 – Diagrama de Classes



Fonte: Própria (2022).

Para a consecução do objetivo primário do trabalho, faz-se necessária a implementação de diversos métodos responsáveis pelo comportamento do Algoritmo Genético, tais como: "inicializar população", "ordenar população", "selecionar melhor indivíduo", "selecionar pai" e, principalmente, "resolver".

O método resolver() será responsável, dentro de um contexto de programação estruturada, por inicializar a população, ordenar os indivíduos por ordem crescente de aptidão (maior valor carregado, sem ultrapassar os limites determinados), selecionar a melhor solução (indivíduo mais apto) e realizar o cross-over (reprodução).

Para implementação[1] do AG, foi utilizada a linguagem de programação Python, o que trouxe um ganho elevado de produtividade. O pacote *Matplotlib* foi utilizado para a

[1] Disponível em https://github.com/ygordev/Algoritmos-Geneticos-ESPG. Acesso em 13 jun. 2022.

visualização gráfica do desempenho do Algoritmo Genético. A implementação final encontra-se no apêndice "A" deste trabalho.

5 RESULTADOS

Algoritmos Genéticos são utilizados para a obtenção de soluções não-determinísticas, mas que podem ser aproximadas por um valor considerado ótimo. Desta forma, gerou-se 400 gerações, com uma taxa de mutação de 1%, que, segundo a literatura, é um valor razoável.

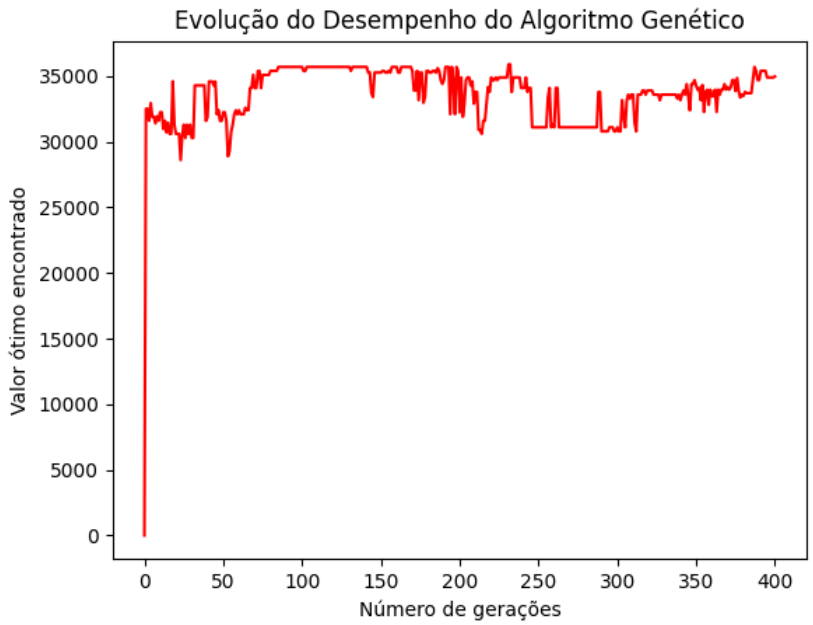
A obtenção da melhor solução foi obtida na 66ª geração, conforme pode-se observar na figura 3, com um total de espaço utilizado 2,9173 m^3 e valor de carga total R$35904.67, sendo esta descrita na tabela 2:

Tabela 2 – Composição da Carga – Solução Ótima

Carga	Valor (R$)
iPhone X	2.199,12
iPhone 13 Pro	10.911,12
TV 55"	4.346,99
TV 50"	3.999,90
TV 42"	2.999,90
Notebook Dell	2.499,90
Microondas Electrolux	308,66
Microondas LG	429,90
Microondas Panasonic	299,29
Geladeira Consul	1.199,89
Notebook Lenovo	1.999,90
Notebook Asus	3.999,00

Fonte: Própria (2022).

Figura 3 – Desempenho do AG para a otimização da carga



Fonte: Própria (2022).

6 CONSIDERAÇÕES FINAIS

Através da modelagem do problema inicial, definindo espaço utilizado e valor do produto, foi possível utilizar Algoritmos Genéticos para a obtenção de uma solução ótima de carregamento de caminhão. Tal abordagem foi possível por se tratar de um problema de otimização de uma função custo, sendo, neste caso, maximização.

REFERÊNCIAS

BRASIL. Ministério da Infraestrutura. **Rodovias Federais**. [S. l.], 29 abr. 2019. Disponível em: http://antigo.infraestrutura.gov.br/rodovias-brasileiras.html. Acesso em: 1 fev. 2022.

LACERDA, E. G. M; CARVALHO, A. C. P. L. F. **Introdução aos algoritmos genéticos**, Porto Alegre: Ed. Universidade/UFRGS : Associação Brasileira de Recursos Hídricos, 1999, v. , p. 99-150

PEREIRA, Eduardo. **Computação Evolucionária**: Aplique os algoritmos genéticos com Python e Numpy. [S. l.]: Casa do Código, 2020. 166 p. v. 1. ISBN 978-65-86110-34-0. E-book.

SIVANANDAM, S. N.; DEEPA, S. N. **Introduction to Genetic Algorithms**. 1. ed. New York: Springer, 2008. ISBN 978-3-540-73189-4.

APÊNDICE A - Algoritmo Genético

```
from random import random
import matplotlib.pyplot as plt

class Produto():
    def __init__(self, nome, espaco, valor):
        self.nome = nome
        self.espaco = espaco
        self.valor = valor

class Individuo():
    def __init__(self, espacos, valores, limite_espacos, geracao=0):
        self.espacos = espacos
        self.valores = valores
        self.limite_espacos = limite_espacos
        self.geracao = geracao
        self.nota_avaliacao = 0
        self.espaco_usado = 0
        self.cromossomo = []

        for i in range(len(espacos)):
            if random() < 0.5:
                self.cromossomo.append("0")
            else:
                self.cromossomo.append("1")

    def avaliacao(self):
        nota = 0
        soma_espacos = 0

        for i in range(len(self.cromossomo)):
            if self.cromossomo[i] == '1':
                nota += self.valores[i]
                soma_espacos += self.espacos[i]

        if soma_espacos > self.limite_espacos:
            nota = 1
        self.nota_avaliacao = nota
        self.espaco_usado = soma_espacos

    def mostraInformacoes(self):
        print("Espaco = %s" % str(self.espacos))
        print("Valores = %s" % str(self.valores))
        print("Cromossomo = %s" % str(self.cromossomo))
```

```
    def mostraAvaliacao(self):
        self.avaliacao()
        print("Nota do individuo: %s" % self.nota_avaliacao)
        print("Espaco utilizado pelo individuo: %s" % self.espaco_usado)

    def mostraComponentesCarga(self):
        print("\nComponentes da carga:")
        for i in range(len(lista_produtos)):
            if self.cromossomo[i] == '1':
                print("Nome: %s \t\t\tValor: %s" % (nomes[i],
valores[i]))

    def resumo(self):
        self.mostraInformacoes()
        self.mostraAvaliacao()
        self.mostraComponentesCarga()

    def crossOver(self, outro_individuo):
        corte = round(random() * len(self.cromossomo))

        filho1 = outro_individuo.cromossomo[0:corte] +
self.cromossomo[corte::]
        filho2 = self.cromossomo[0:corte] +
outro_individuo.cromossomo[corte::]

        filhos = [Individuo(self.espacos, self.valores,
self.limite_espacos, self.geracao+1),
                  Individuo(self.espacos, self.valores,
self.limite_espacos, self.geracao+1)]

        filhos[0].cromossomo = filho1
        filhos[1].cromossomo = filho2

        return filhos

    def mutacao(self, taxa_mutacao):
        #print("\nAntes: %s" % self.cromossomo)

        for i in range(len(self.cromossomo)):
            if random() < taxa_mutacao:
                if self.cromossomo[i] == '1':
                    self.cromossomo[i] = '0'
                else:
                    self.cromossomo[i] = '1'

        #print("Depois: %s" % self.cromossomo)
```

```
        return self

class AlgoritmoGenetico():
    def __init__(self, tamanho_populacao):
        self.tamanho_populacao = tamanho_populacao
        self.populacao = []
        self.geracao = 0
        self.melhor_solucao = 0
        self.lista_solucoes = []

    def inicializaPopulacao(self, espacos, valores, limite_espacos):
        for i in range(self.tamanho_populacao):
            self.populacao.append(Individuo(espacos, valores,
limite_espacos))
        self.melhor_solucao = self.populacao[0]

    def ordenaPopulacao(self):
        self.populacao = sorted(self.populacao, key = lambda populacao:
populacao.nota_avaliacao, reverse=True)

    def melhorIndividuo(self, individuo):
        if individuo.nota_avaliacao >
self.melhor_solucao.nota_avaliacao:
            self.melhor_solucao = individuo

    def mostraPopulacao(self):
        for i in range(self.tamanho_populacao):
            print("\n=======Individuo %s====" % str(i+1))
            self.populacao[i].resumo()

    def somaAvaliacoes(self):
        soma = 0
        for individuo in self.populacao:
            soma += individuo.nota_avaliacao
        return soma

    def selecionaPai(self, soma_avaliacao):
        pai = -1
        valor_sorteado = random() * soma_avaliacao
        soma = 0
        i = 0
        while i < len(self.populacao) and soma < valor_sorteado:
            soma += self.populacao[i].nota_avaliacao
            pai += 1
            i += 1
        return pai
```

```
    def visualizaGeracao(self):
        melhor = self.populacao[0]
        print("Geracao: %s -> Valor: R$%s \tEspaço: %s\tCromossomo: %s"
% (self.populacao[0].geracao,

melhor.nota_avaliacao,

melhor.espaco_usado,

melhor.cromossomo))

    def resolver(self, taxa_mutacao, numero_geracoes, espacos, valores,
limite_espacos):
        self.inicializaPopulacao(espacos, valores, limite_espacos)

        for individuo in self.populacao:
            individuo.avaliacao()

        self.ordenaPopulacao()

        melhor_solucao = self.populacao[0]

        self.lista_solucoes.append(self.melhor_solucao.nota_avaliacao)

        self.visualizaGeracao()

        for geracao in range(numero_geracoes):
            soma_avaliacao = self.somaAvaliacoes()
            nova_populacao = []

            for individuos_gerados in range(0, self.tamanho_populacao,
2):
                pai1 = self.selecionaPai(soma_avaliacao)
                pai2 = self.selecionaPai(soma_avaliacao)

                filhos =
self.populacao[pai1].crossOver(self.populacao[pai2])

                nova_populacao.append(filhos[0].mutacao(taxa_mutacao))
                nova_populacao.append(filhos[1].mutacao(taxa_mutacao))

            self.populacao = list(nova_populacao)

            for individuo in self.populacao:
                individuo.avaliacao()
```

```
            self.ordenaPopulacao()

            self.visualizaGeracao()

            melhor = self.populacao[0]
            self.lista_solucoes.append(melhor.nota_avaliacao)
            self.melhorIndividuo(melhor)

        print("\nMelhor solução -> G: %s -> Valor: R$%s -> Espaço: %s ->
Cromossomo: %s" % (self.melhor_solucao.geracao,

self.melhor_solucao.nota_avaliacao,

self.melhor_solucao.espaco_usado,

self.melhor_solucao.cromossomo))
        return self.melhor_solucao

    def mostraGrafico(self):
        plt.plot(ag.lista_solucoes, color='red')
        plt.title('Evolução do Desempenho do Algoritmo Genético')
        plt.xlabel('Número de gerações')
        plt.ylabel('Valor ótimo encontrado')
        plt.show()

## ======= main() ======

if __name__ == '__main__':

    lista_produtos = []
    lista_produtos.append(Produto("Geladeira Dako", 0.751, 999.90))
    lista_produtos.append(Produto("iPhone X", 0.0000899, 2911.12))
    lista_produtos.append(Produto("iPhone 13 Pro", 0.0000899, 10911.12))
    lista_produtos.append(Produto("Tv 55' ", 0.400, 4346.99))
    lista_produtos.append(Produto("Tv 50' ", 0.290, 3999.90))
    lista_produtos.append(Produto("Tv 42' ", 0.200, 2999.00))
    lista_produtos.append(Produto("Notebook Dell", 0.00350, 2499.90))
    lista_produtos.append(Produto("Ventilador Panasonic", 0.496,
199.90))
    lista_produtos.append(Produto("Microondas Electrolux", 0.0424,
308.66))
    lista_produtos.append(Produto("Microondas LG", 0.0544, 429.90))
    lista_produtos.append(Produto("Microondas Panasonic", 0.0319,
299.29))
    lista_produtos.append(Produto("Geladeira Brastemp", 0.635, 849.00))
```

```
    lista_produtos.append(Produto("Geladeira Consul", 0.870, 1199.89))
    lista_produtos.append(Produto("Notebook Lenovo", 0.498, 1999.90))
    lista_produtos.append(Produto("Notebook Asus", 0.527, 3999.00))

    espacos = []
    valores = []
    nomes = []

    for produto in lista_produtos:
        espacos.append(produto.espaco)
        valores.append(produto.valor)
        nomes.append(produto.nome)

    limite = 3 # limite de 3 metros cúbicos no caminhão
    tamanho_populacao = 20
    taxa_mutacao = 0.01
    numero_geracoes = 400

    ag = AlgoritmoGenetico(tamanho_populacao)

    resultado = ag.resolver(taxa_mutacao, numero_geracoes, espacos,
valores, limite)

    for i in range(len(lista_produtos)):
        if resultado.cromossomo[i] == '1':
            print("Nome: %s R$ %s" % (lista_produtos[i].nome,
lista_produtos[i].valor))

    resultado.resumo()
    ag.mostraGrafico()
```