Algoritmo Genético Combinatório para o Problema do Caixeiro Viajante

Vítor Rezende Silva

Agosto 2024

1 Introdução

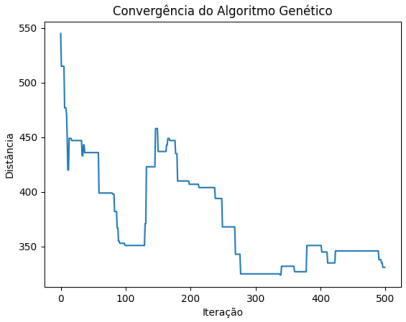
Para que o algoritmo seja executado ele deve receber as seguintes variáveis como entrada:

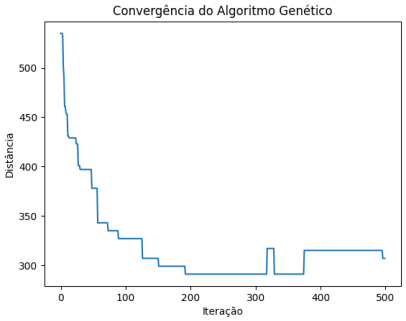
- num_iteracoes: o número de iterações até o fim da execução
- num_individuos: o número de indivíduos por população
- tc: taxa de cruzamento
- tm: taxa de mutação
- filename: nome do arquivo contendo a matriz de distâncias entre as cidades

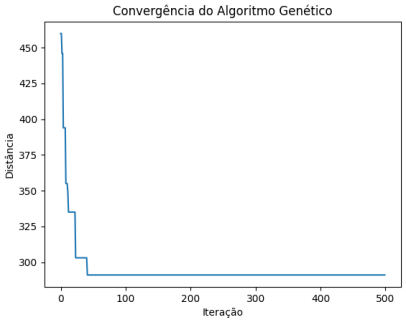
O algoritmo foi implementado usando elitismo, ou seja, o melhor indivíduo de cada geração será passado adiante para a próxima, esse tópico será alvo de discussão mais adiante, e o método de seleção de pais utilizado foi o torneio. Ao fim de cada execução é gerado um gráfico com a variação do valor de fitness com o tempo e um arquivo chamado gen.txt que registra esses mesmo valores.

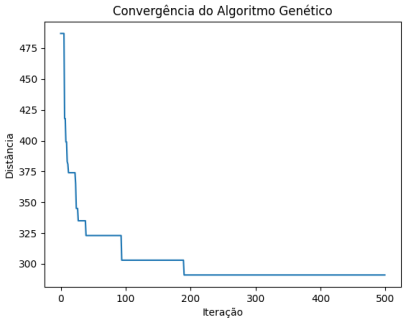
2 Testes

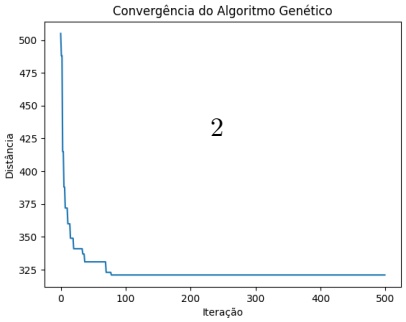
Como foi dito anteriormente, o algoritmo foi implementado com elitismo, ou seja, o melhor indivíduo de cada geração deverá ter o seu valor de fitness sempre menor ou igual ao da última. Contudo, por motivos que falharam a minha compreensão durante a implementação desse algoritmo, em algumas gerações, o individuo de melhor fitness encontrado pelo algoritmo teria um valor maior do que o anterior, o que em teoria seria impossível. Observe os seguintes testes:


Convergência do Algoritmo Genético
Distância
Iteração
550
500
450
400
350
0
100
200
300
400
500


Convergência do Algoritmo Genético
Distância
Iteração
500
450
400
350
300
0
100
200
300
400
500


Convergência do Algoritmo Genético
Distância
Iteração
450
425
400
375
350
325
300
0
100
200
300
400
500


Convergência do Algoritmo Genético
Distância
Iteração
475
450
425
400
375
350
325
300
0
100
200
300
400
500


Convergência do Algoritmo Genético
Distância
Iteração
500
475
450
425
400
375
350
325
0
100
200
300
400
500

Todos eles foram feitos com os mesmos valores de `num_iteracoes`, `tc` e `tm`, sendo eles 500, 70 e 5 respectivamente, só variando o `num_individuos` nos valores 50, 100, 200, 300 e 500. Variando esse valor foi possível observar que esse problema estava acontecendo em populações pequenas, enquanto que com valores maiores ele praticamente não ocorria.

Eventualmente esse problema foi resolvido. Para isso, nas funções main e nova_pop foram trocadas 3 atribuições, o melhor individuo de cada geração era atribuido a variavel "melhor", contudo, por algum motivo em algumas execuções esse valor estava sendo arbitrariamente alterado, por isso, no lugar de passar a lista diretamente foi passada uma cópia da lista usando a biblioteca copy.

Código antigo:

```
def nova_pop(pop, tc, tm, num_individuos, melhor):
tam = len(pop[0])

pop_filhos = [melhor]


i = 0
while len(pop_filhos) < num_individuos:
    if i+1 >= tam:
        break

    f1, f2 = cruzamento(pop[i], pop[i+1], tc)
    pop_filhos.append(f1)
    if len(pop_filhos) < num_individuos:
        pop_filhos.append(f2)

for i in range(1, num_individuos):
    mutacao(pop_filhos[i], tm)

return pop_filhos

def main(num_iteracoes, num_individuos, tc, tm, filename):
file = open("AG_combinatorio/gen.txt", "w")
matriz, tam = get_cidades(filename)
cont = 0
iteracao = []
resultados = []

pop = cria_pop(num_individuos, tam)

indice, distancia = melhor_ind(pop, matriz)
melhor = pop[indice]

while cont < num_iteracoes:
```

```
        file.write(str(pop[indice]) + "\n")
        file.write(str(distancia) + " " + str(cont))
        file.write("\n------------------------------------------------------\n")

        resultados.append(distancia)
        iteracao.append(cont)

        melhores = torneio(pop, num_individuos, matriz)
        pop = nova_pop(melhores, tc, tm, num_individuos, melhor)

        indice, distancia = melhor_ind(pop, matriz)
        melhor = pop[indice]

        cont += 1
```

Código novo:

```
    def nova_pop(pop, tc, tm, num_individuos, melhor):
    tam = len(pop[0])

    pop_filhos = [copy.deepcopy(melhor)]


    i = 0
    while len(pop_filhos) < num_individuos:
        if i+1 >= tam:
            break

        f1, f2 = cruzamento(pop[i], pop[i+1], tc)
        pop_filhos.append(f1)
        if len(pop_filhos) < num_individuos:
            pop_filhos.append(f2)

    for i in range(1, num_individuos):
        mutacao(pop_filhos[i], tm)

    return pop_filhos

def main(num_iteracoes, num_individuos, tc, tm, filename):
    file = open("AG_combinatorio/gen.txt", "w")
    matriz, tam = get_cidades(filename)
    cont = 0
    iteracao = []
    resultados = []

    pop = cria_pop(num_individuos, tam)
```

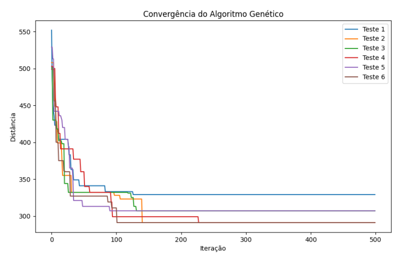
```
    indice, distancia = melhor_ind(pop, matriz)
    melhor = copy.deepcopy(pop[indice])

    while cont < num_iteracoes:

        file.write(str(pop[indice]) + "\n")
        file.write(str(distancia) + " " + str(cont))
        file.write("\n------------------------------------------------------\n")

        resultados.append(distancia)
        iteracao.append(cont)

        melhores = torneio(pop, num_individuos, matriz)
        pop = nova_pop(melhores, tc, tm, num_individuos, melhor)

        indice, distancia = melhor_ind(pop, matriz)
        melhor = copy.deepcopy(pop[indice])

        cont += 1
```

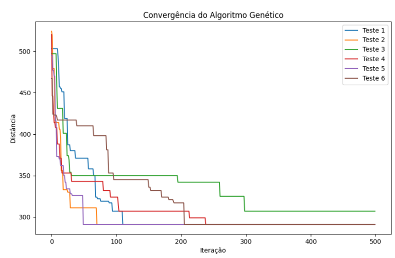
Depois desse problema ser resolvido foram realizados testes sobre o algoritmo usando o seguinte script em python

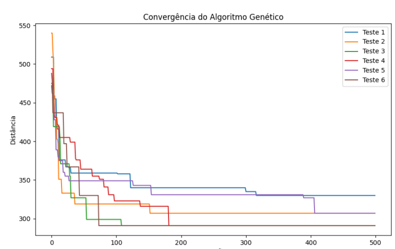
```
    from AG import *
import matplotlib.pyplot as plt

e = [[500, 100, 100, 1], [500, 100, 80, 1],
     [500, 200, 80, 2], [500, 200, 70, 2],
     [500, 300, 70, 5], [500, 300, 70, 5]]

distancias = []
resultados = []
iteracoes = []

for params in e:
    distancia, resultado, iteracao = main(params[0], params[1], params[2], params[3], "AG_co
    distancias.append(distancia)
    resultados.append(resultado)
    iteracoes.append(iteracao)

print(*distancias)

plt.figure(figsize=(10, 6))

for i in range(len(resultados)):
    plt.plot(iteracoes[i], resultados[i], label=f'Teste {i+1}')
```

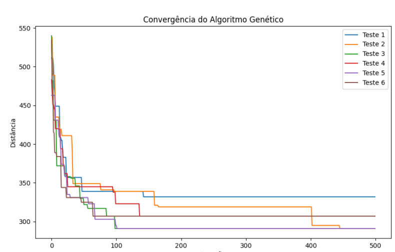
```
plt.xlabel('Iteração')
plt.ylabel('Distância')
plt.title('Convergência do Algoritmo Genético')
plt.legend()
plt.show()
```

O script foi executado 4 vezes e os resultados obtidos foram os seguintes:









No geral, os testes que obteram os melhores resultados foi o teste 6, onde os valores de entrada são, [500, 300, 70, 5], encontrando o valor ótimo, 291, com o menor número de iterações