# Análise de performance de algoritmos

**Mateus Walz, Jhuan Gabriel de Souza, Kelvim Silva Saidel**

**Engenharia de Software Universidade da região de Joinville – Univille – Joinville, SC – Brazil { mateus.walz@gmail.com, jhuan120@gmail.com, kelvimsfs@gmail.com}**

***Resumo.*** *Este trabalho teve por finalidade a realização de um experimento que visava verificar qual algoritmo entre aleatório, guloso e semi-guloso apresentava a melhor performance. De acordo com os resultados gerados podemos constatar que o algoritmo guloso se mostrou o mais eficiente dentre eles.*

## 1. Algoritmo aleatório

Busca a resolução de um problema de forma randômica sem qualquer outro método de resolução.

**Algoritmo Aleatório**

```
from random import randrange

for c in range(30):

    distancia_final = []
    solucao = []
    cidades_copia = []

    for i in range(len(cidades)):
        cidades_copia.append(i)

    while len(cidades_copia) > 0:
        indice = randrange(len(cidades_copia))
        cidade_sorteada = cidades_copia[indice]
        del cidades_copia[indice]
        solucao.append(cidade_sorteada)

        distancia_final.append(funcao_objetivo(solucao))


#print("Solução final :",solucao, " Distancia = ",funcao_objetivo(solucao))
print(distancia_final)
```

(Exemplo1, 2021);

## 2. Algoritmo guloso

É um algoritmo que buscar encontrar um ótimo global resolvendo o problema em fases e escolhendo sempre a solução ótima para cada uma.

**Algoritmo Guloso**

```
from random import randrange

for c in range(40):

    distancia_final = []
    solucao = []
    cidades_copia = []

    for i in range(len(cidades)):
      cidades_copia.append(i)

    # Seleciona a cidade de partida
    indice = randrange(len(cidades_copia))
    cidade_sorteada = cidades_copia[indice]
    del cidades_copia[indice]
    solucao.append(cidade_sorteada)

    while len(cidades_copia) > 0:
        cidade_de_onde_esta_saindo = cidades[solucao[len(solucao)-1]]
        cidade_mais_proxima = cidades_copia[0]
        cidade_mais_proxima_indice = 0
        menor_distancia = matriz_distancias[cidade_de_onde_esta_saindo-1][cidade_mais_proxima]##[cidade_de_onde_esta_saindo][cidade_mais_proxima]
        #print("------")
        #print("Cidade de onde esta saindo:",cidade_de_onde_esta_saindo)
        i = 0
        for proxima_cidade_candidata  in cidades_copia:
            distancia = matriz_distancias[cidade_de_onde_esta_saindo-1][proxima_cidade_candidata]
            #print("Cidade",cidades[proxima_cidade_candidata]," distancia ",distancia)
            if distancia < menor_distancia:
                menor_distancia = distancia
                cidade_mais_proxima = proxima_cidade_candidata
                cidade_mais_proxima_indice = i
            i = i + 1
        #print("Cidade mais proxima:",cidade_mais_proxima,"Distancia",menor_distancia)

        solucao.append(cidades_copia[cidade_mais_proxima_indice])
        distancia_final.append(funcao_objetivo(solucao))
        del cidades_copia[cidade_mais_proxima_indice]


    #print("Solução final :",solucao, " Di =stancia ",funcao_objetivo(solucao))
print(distancia_final)
```

(Exemplo2, 2021);

## 3. Algoritmo semi-guloso

Seleciona elementos de forma aleatória e depois classifica de acordo com o método guloso.

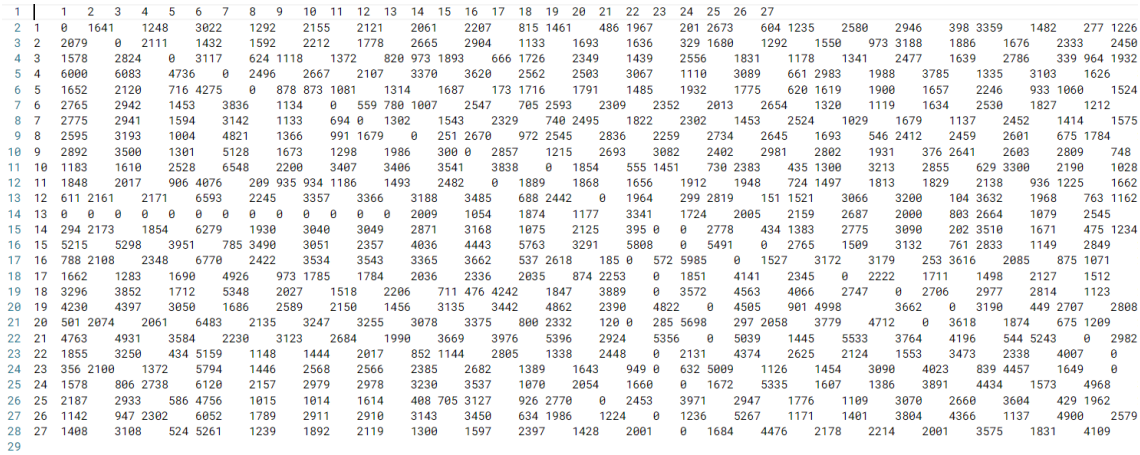
**Algoritmo Semi-Guloso**

```
from random import randrange


for c in range(40):

    distancia_final = []
    solucao = []
    cidades_copia = []
    delta = 70

    for i in range(len(cidades)):
        cidades_copia.append(i)

    # Seleciona a cidade de partida
    indice = randrange(len(cidades_copia))
    cidade_sorteada = cidades_copia[indice]
    del cidades_copia[indice]
    solucao.append(cidade_sorteada)

    while len(cidades_copia) > 0:
        # Sorteia a probabilidade de selecionar o método guloso ou aleatorio.
        valor_aleatorio = randrange(100)

        if delta > valor_aleatorio:
            #-------
            #. Metodo Guloso
            #-------
            #print("Metodo Guloso")
            cidade_de_onde_esta_saindo = cidades[solucao[len(solucao)-1]]
            cidade_mais_proxima = cidades_copia[0]
            cidade_mais_proxima_indice = 0
            menor_distancia = matriz_distancias[cidade_de_onde_esta_saindo-1][cidade_mais_proxima]
            #print("------")
            #print("Cidade de onde esta saindo:",cidade_de_onde_esta_saindo)
            i = 0
            for proxima_cidade_candidata  in cidades_copia:
                distancia = matriz_distancias[cidade_de_onde_esta_saindo-1][proxima_cidade_candidata]
                #print("Cidade",cidades[proxima_cidade_candidata]," distancia ",distancia)
                if distancia < menor_distancia:
                    menor_distancia = distancia
                    cidade_mais_proxima = proxima_cidade_candidata
                    cidade_mais_proxima_indice = i
                i = i + 1
            #print("Cidade mais proxima:",cidade_mais_proxima,"Distancia",menor_distancia)

            solucao.append(cidades_copia[cidade_mais_proxima_indice])
            distancia_final.append(funcao_objetivo(solucao))
            del cidades_copia[cidade_mais_proxima_indice]

        else:
            #-------
            #. Metodo Aleatório
            #-------
           # print("Metodo Aleatório")
            indice = randrange(len(cidades_copia))
            cidade_sorteada = cidades_copia[indice]
            del cidades_copia[indice]
            solucao.append(cidade_sorteada)

            distancia_final.append(funcao_objetivo(solucao))

#print("Solução final :",solucao, " Distancia = ",funcao_objetivo(solucao))
print(distancia_final)
```

(Exemplo3, 2021)

## 4. Metodologia

Nesse trabalho foi utilizado os algoritmos padrões desenvolvidos em sala de aula com algumas modificações. O algoritmo foi alterado para receber um arquivo de texto de acordo com a (figura1, 2021), em seguida separa a sua primeira linha a qual forma um vetor denominado *cidade*, pois nele possui 27 números que correspondem as 27 capitais brasileiras.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1641 | 1248 | 3022 | 1292 | 2155 | 2121 | 2061 | 2207 | 815 | 1461 | 486 | 1967 | 201 | 2673 | 604 | 1235 | 2580 | 2946 | 398 | 3359 | 1482 | 277 | 1226 | | | |
| 2 | 2079 | 0 | 2111 | 1432 | 1592 | 2212 | 1778 | 2665 | 2904 | 1133 | 1693 | 1636 | 329 | 1680 | 1292 | 1550 | 973 | 3188 | 1886 | 1676 | 2333 | 2450 | | | | | |
| 3 | 1578 | 2824 | 0 | 3117 | 624 | 1118 | 1372 | 820 | 973 | 1893 | 666 | 1726 | 2349 | 1439 | 2556 | 1831 | 1178 | 1341 | 2477 | 1639 | 2786 | 339 | 964 | 1932 | | | |
| 4 | 6000 | 6083 | 4736 | 0 | 2496 | 2667 | 2107 | 3370 | 3620 | 2562 | 2503 | 3067 | 1110 | 3089 | 661 | 2983 | 1988 | 3785 | 1335 | 3103 | 1626 | | | | | | |
| 5 | 1652 | 2120 | 716 | 4275 | 0 | 878 | 873 | 1081 | 1314 | 1687 | 173 | 1716 | 1791 | 1485 | 1932 | 1775 | 620 | 1619 | 1900 | 1657 | 2246 | 933 | 1060 | 1524 | | | |
| 6 | 2765 | 2942 | 1453 | 3836 | 1134 | 0 | 559 | 780 | 1007 | 2547 | 705 | 2593 | 2309 | 2352 | 2013 | 2654 | 1320 | 1119 | 1634 | 2530 | 1827 | 1212 | | | | | |
| 7 | 2775 | 2941 | 1594 | 3142 | 1133 | 694 | 0 | 1302 | 1543 | 2329 | 740 | 2495 | 1822 | 2302 | 1453 | 2524 | 1029 | 1679 | 1137 | 2452 | 1414 | 1575 | | | | | |
| 8 | 2595 | 3193 | 1004 | 4821 | 1366 | 991 | 1679 | 0 | 251 | 2670 | 972 | 2545 | 2836 | 2259 | 2734 | 2645 | 1693 | 546 | 2412 | 2459 | 2601 | 675 | 1784 | | | | |
| 9 | 2892 | 3500 | 1301 | 5128 | 1673 | 1298 | 1986 | 300 | 0 | 2857 | 1215 | 2693 | 3082 | 2402 | 2981 | 2802 | 1931 | 376 | 2641 | 2603 | 2809 | 748 | | | | | |
| 10 | 1183 | 1610 | 2528 | 6548 | 2200 | 3407 | 3406 | 3541 | 3838 | 0 | 1854 | 555 | 1451 | 730 | 2383 | 435 | 1300 | 3213 | 2855 | 629 | 3300 | 2190 | 1028 | | | | |
| 11 | 1848 | 2017 | 906 | 4076 | 209 | 935 | 934 | 1186 | 1493 | 2482 | 0 | 1889 | 1868 | 1656 | 1912 | 1948 | 724 | 1497 | 1813 | 1829 | 2138 | 936 | 1225 | 1662 | | | |
| 12 | 611 | 2161 | 2171 | 6593 | 2245 | 3357 | 3366 | 3188 | 3485 | 688 | 2442 | 0 | 1964 | 299 | 2819 | 151 | 1521 | 3066 | 3200 | 104 | 3632 | 1968 | 763 | 1162 | | | |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2009 | 1054 | 1874 | 1177 | 3341 | 1724 | 2005 | 2159 | 2687 | 2000 | 803 | 2664 | 1079 | 2545 | |
| 14 | 294 | 2173 | 1854 | 6279 | 1930 | 3040 | 3049 | 2871 | 3168 | 1075 | 2125 | 395 | 0 | 0 | 2778 | 434 | 1383 | 2775 | 3090 | 202 | 3510 | 1671 | 475 | 1234 | | | |
| 15 | 5215 | 5298 | 3951 | 785 | 3490 | 3051 | 2357 | 4036 | 4443 | 5763 | 3291 | 5808 | 0 | 5491 | 0 | 2765 | 1509 | 3132 | 761 | 2833 | 1149 | 2849 | | | | | |
| 16 | 788 | 2108 | 2348 | 6770 | 2422 | 3534 | 3543 | 3365 | 3662 | 537 | 2618 | 185 | 0 | 572 | 5985 | 0 | 1527 | 3172 | 3179 | 253 | 3616 | 2085 | 875 | 1071 | | | |
| 17 | 1662 | 1283 | 1690 | 4926 | 973 | 1785 | 1784 | 2036 | 2336 | 2035 | 874 | 2253 | 0 | 1851 | 4141 | 2345 | 0 | 2222 | 1711 | 1498 | 2127 | 1512 | | | | | |
| 18 | 3296 | 3852 | 1712 | 5348 | 2027 | 1518 | 2206 | 711 | 476 | 4242 | 1847 | 3889 | 0 | 3572 | 4563 | 4066 | 2747 | 0 | 2706 | 2977 | 2814 | 1123 | | | | | |
| 19 | 4230 | 4397 | 3050 | 1686 | 2589 | 2150 | 1456 | 3135 | 3442 | 4862 | 2390 | 4822 | 0 | 4505 | 901 | 4998 | | 3662 | 0 | 3190 | 449 | 2707 | 2808 | | | | |
| 20 | 501 | 2074 | 2061 | 6483 | 2135 | 3247 | 3255 | 3078 | 3375 | 800 | 2332 | 120 | 0 | 285 | 5698 | 297 | 2058 | 3779 | 4712 | 0 | 3618 | 1874 | 675 | 1209 | | | |
| 21 | 4763 | 4931 | 3584 | 2230 | 3123 | 2684 | 1990 | 3669 | 3976 | 5396 | 2924 | 5356 | 0 | 5039 | 1445 | 5533 | 3764 | 4196 | 544 | 5243 | 0 | 2982 | | | | | |
| 22 | 1855 | 3250 | 434 | 5159 | 1148 | 1444 | 2017 | 852 | 1144 | 2805 | 1338 | 2448 | 0 | 2131 | 4374 | 2625 | 2124 | 1553 | 3473 | 2338 | 4007 | 0 | | | | | |
| 23 | 356 | 2100 | 1372 | 5794 | 1446 | 2568 | 2566 | 2385 | 2682 | 1389 | 1643 | 949 | 0 | 632 | 5089 | 1126 | 1454 | 3090 | 4023 | 839 | 4457 | 1649 | 0 | | | | |
| 24 | 1578 | 806 | 2738 | 6120 | 2157 | 2979 | 2978 | 3230 | 3537 | 1070 | 2054 | 1660 | 0 | 1672 | 5335 | 1607 | 1386 | 3891 | 4434 | 1573 | 4968 | | | | | | |
| 25 | 2187 | 2933 | 586 | 4756 | 1015 | 1014 | 1614 | 408 | 705 | 3127 | 926 | 2770 | 0 | 2453 | 3971 | 2947 | 1776 | 1109 | 3070 | 2660 | 3604 | 429 | 1962 | | | | |
| 26 | 1142 | 947 | 2302 | 6052 | 1789 | 2911 | 2910 | 3143 | 3450 | 634 | 1986 | 1224 | 0 | 1236 | 5267 | 1171 | 1401 | 3804 | 4366 | 1137 | 4900 | 2579 | | | | | |
| 27 | 1408 | 3108 | 524 | 5261 | 1239 | 1892 | 2119 | 1300 | 1597 | 2397 | 1428 | 2001 | 0 | 1684 | 4476 | 2178 | 2214 | 2001 | 3575 | 1831 | 4109 | | | | | | |

(figura1,2021).

Após isso se fez necessário converter os valores de string para int para que esses pudessem ser contabilizados pelos algoritmos. As demais linhas sofreram o mesmo processo, mas todas foram alocadas em forma de vetores tipo int dentro de uma matriz chamada de *matriz_distancia* que serviria como base de dados para os cálculos de distância, conforme (figura2, 2021) abaixo:

**Carregando arquivo txt + dados de entrada**

```
cidades = []
matriz_distancias = []

# Start writing code here...
file1 = open('matriz_distancia.txt', 'r')
Lines = file1.readlines()

count = 0
# Strips the newline character
for line in Lines:
    count += 1
    print("Line{}: {}".format(count, line.strip()))
    if count == 1:
      cidades = line.split()
      for i in range(len(cidades)):
          cidades[i] = int(cidades[i])
    else:
      matriz_distancias.append(line.split( ))

for l in range(0,27):
        for p in range(0,28):
            matriz_distancias[l][p-1]= int(matriz_distancias[l][p-1])


print("--cidades---")
print(len(cidades))
print("------------")
print("--- Matriz---")
for linha in matriz_distancias:
    print(linha)
```

(figura2, 2021)

Além disso, fora construído um vetor denominado *distancia_final* que possui o objetivo de alocar todas as distancias geradas em cada execução, uma vez que foi implementado um loop que executa 30 vezes os algoritmos, cada vez que estes são acionados, (figura3, 2021).

‘

**Algoritmo Aleatório**

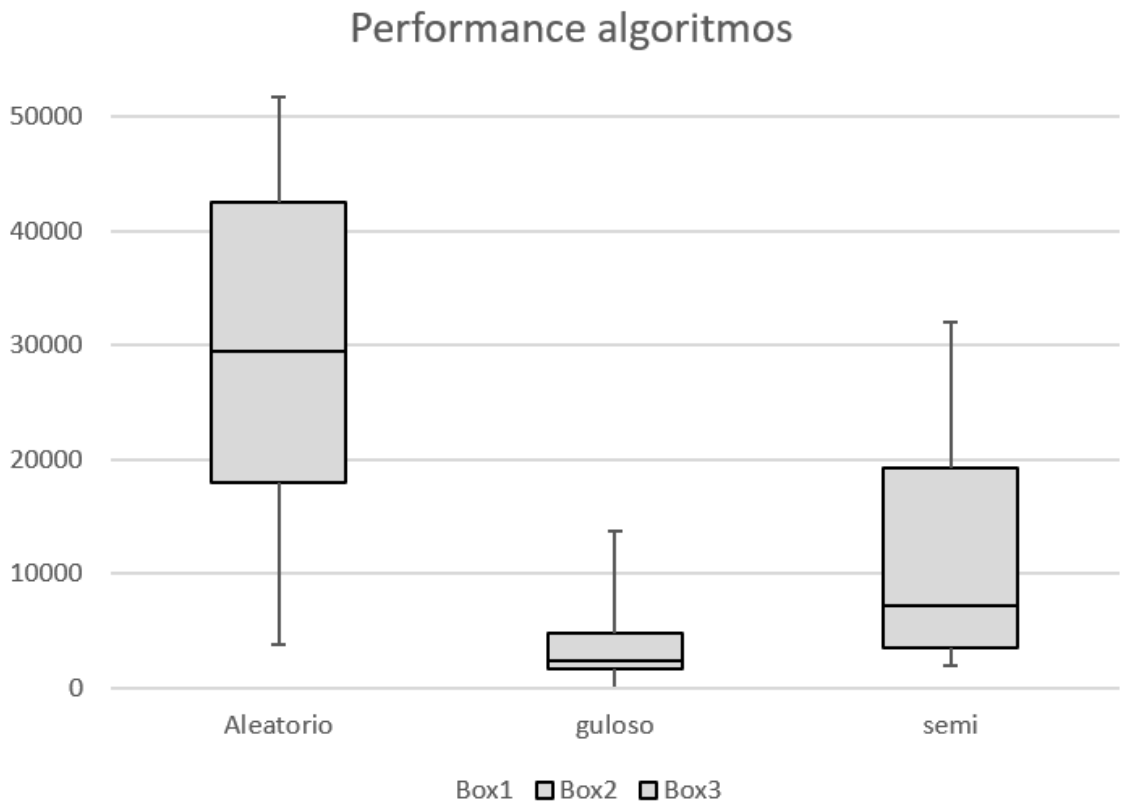
```
from random import randrange

for c in range(30):

    distancia_final = []
    solucao = []
    cidades_copia = []

    for i in range(len(cidades)):
        cidades_copia.append(i)

    while len(cidades_copia) > 0:
        indice = randrange(len(cidades_copia))
        cidade_sorteada = cidades_copia[indice]
        del cidades_copia[indice]
        solucao.append(cidade_sorteada)

        distancia_final.append(funcao_objetivo(solucao))


#print("Solução final :",solucao, " Distancia = ",funcao_objetivo(solucao))
print(distancia_final)
```

(figura3,2021).

Depois o resultado foi transferido para uma planilha em excel onde foi extraído o máximo, mínimo, primeiro, segundo e terceiro quartil, e os limites inferiores e superiores de cada algoritmo. Com isso, foi gerado os diagrama de caixas, além dessas coisas, também foi extraído dos dados a média e o desvio padrão.

## 5. Resultados

Por algum motivo desconhecido até então a execução dos algoritmos só carregou 26 distancias, uma hipótese que justificaria isso seria a quantidade de cidades e de distância entre elas. Mas indo além, os resultados obtidos foram:




(Grafico1, 2021).

| | Aleatório | guloso | semi |
|---|---|---|---|
| **Mínimo** | 3836 | 0 | 1949 |
| **Quartil 1** | 18046,5 | 1676 | 3567,5 |
| **Quartil 2** | 29443 | 2341 | 7179 |
| **Quartil 3** | 42547,25 | 4795,5 | 19234,75 |
| **Máximo** | 51798 | 13779 | 32037 |
| **Lim. Superior** | 9250,75 | 8983,5 | 12802,25 |
| **Lim. Inferior** | 14210,5 | 1676 | 1618,5 |

| | ALEATORIO | GULOSO | SEMI-GULOSO |
|---|---|---|---|
| **Média** | 29337,38462 | 3505,846154 | 12023,80769 |
| **Desvio padrão** | 15134,37834 | 3209,496704 | 10090,31399 |

Podemos observar através do gráficos e dos dados apresentados acima que o algoritmo aleatório como esperado teve a pior performance dentre todos eles, por outro lado, o algoritmo guloso surpreendentemente teve a melhor performance, e o semi-guloso que se esperava ter a melhor performance ficou entre os dois.