

TSP Art Python

Trabalho prático

Ana Torres, Larissa Oliveira e Tatiana Cantelle

GCC 218 - Algoritmos em Grafos
Mayron Moreira

UFLA - 2021/1

Sumário

01

Conhecendo o
TSP Art

02

Gerando as
imagens

03

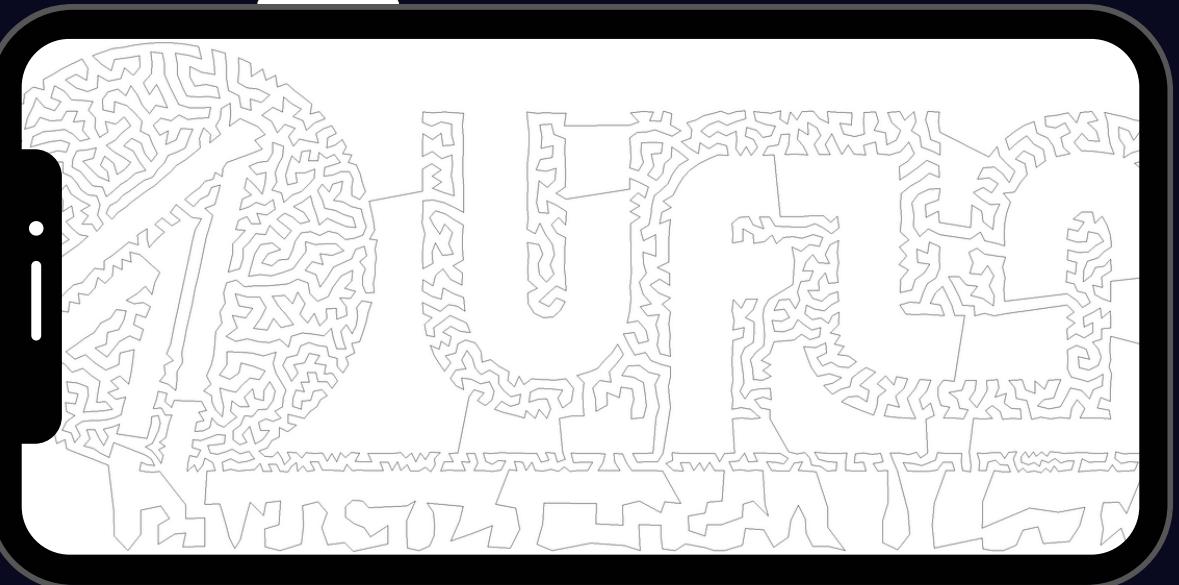
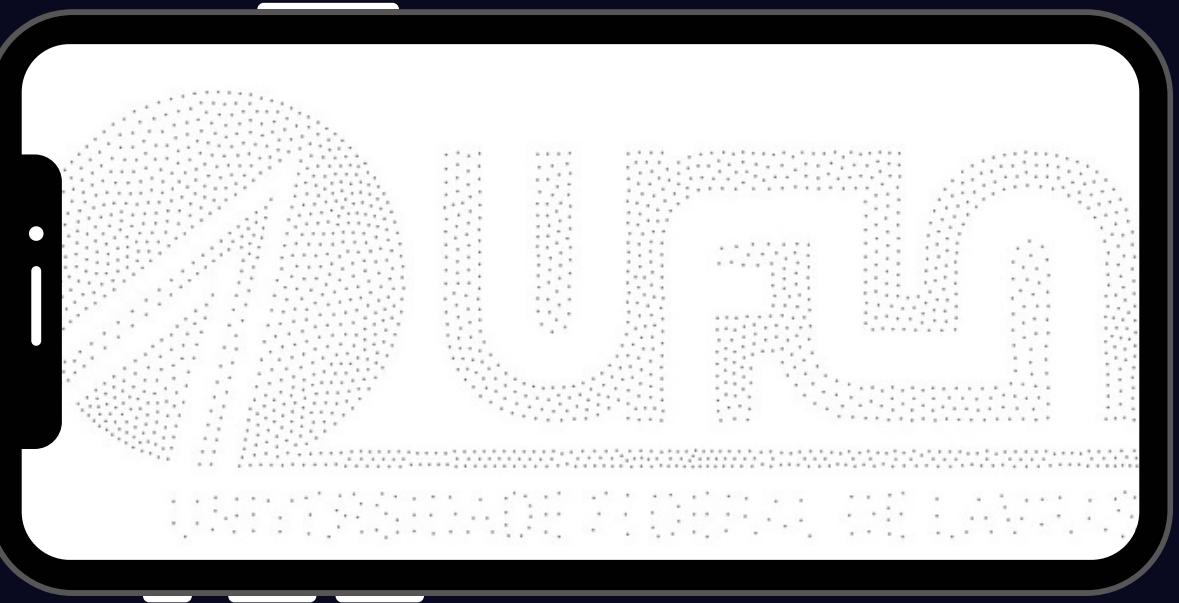
Implementações
e resultados

Conhecendo o **TSP Art**

Este método visa a conversão de uma imagem em desenho pontilhado (*stipple*), no qual cada ponto (vértice) é tratado como as cidades do Problema do Caxeiro Viajante (em inglês, TSP).

A partir de uma cidade, o caxeiro deve visitar todas as outras exatamente uma vez e então retornar para o ponto de origem. Assume-se que ele viajará em linha reta (arestas).

Como resultado, o trajeto é traçado e passível de ser mensurado pela distância entre os nós.



Gerando as imagens



Repositório: [*tsp-art-python*](#), do cientista Matthew Mack.

Definir o caminho da imagem original, bem como o nome e caminho da imagem final no arquivo *draw-tsp-path.py*.

```
C: > Users > dpcdti > Desktop > GCC218 > Trabalho > AnaLarrisaTatianaTspArt > draw-tsp-path.py > ORIGINAL_IMAGE = "images/ufla-logo.png"  
12  
13 # Change these file names to the relevant files.  
14 ORIGINAL_IMAGE = "images/ufla-logo.png"  
15 IMAGE_TSP = "images/ufla-logo-1024-stipple.tsp"  
16
```



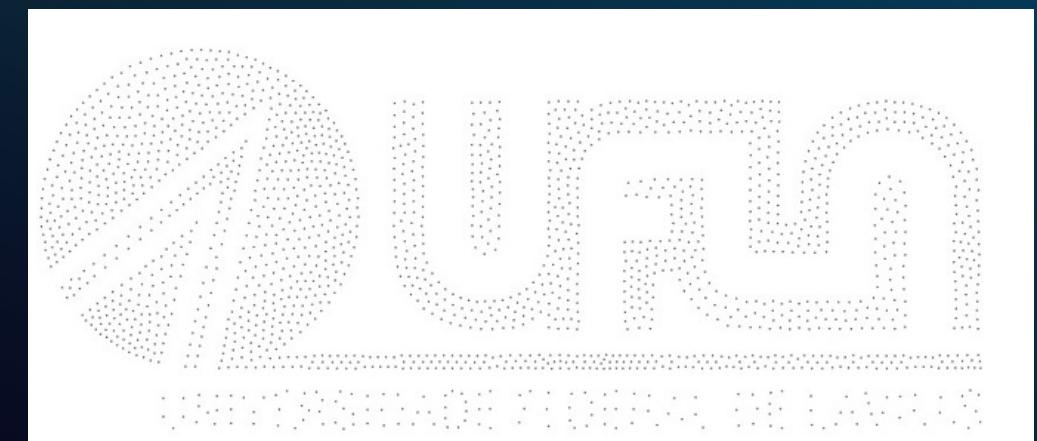
Gerando as imagens



Repositório: [*tsp-art-python*](#), do cientista Matthew Mack.

Definir o número de pontos a serem gerados no arquivo *stippling.py*. Neste trabalho, pode assumir **1024, 2048 ou 3072 pontos**.

```
C: > Users > dpcdti > Desktop > GCC218 > Trabalho > AnaLarrisaTatianaTspArt > stippling.py > ...
12
13 # Total number of points to stipple your image with
14 NUMBER_OF_POINTS = 1024
15
```



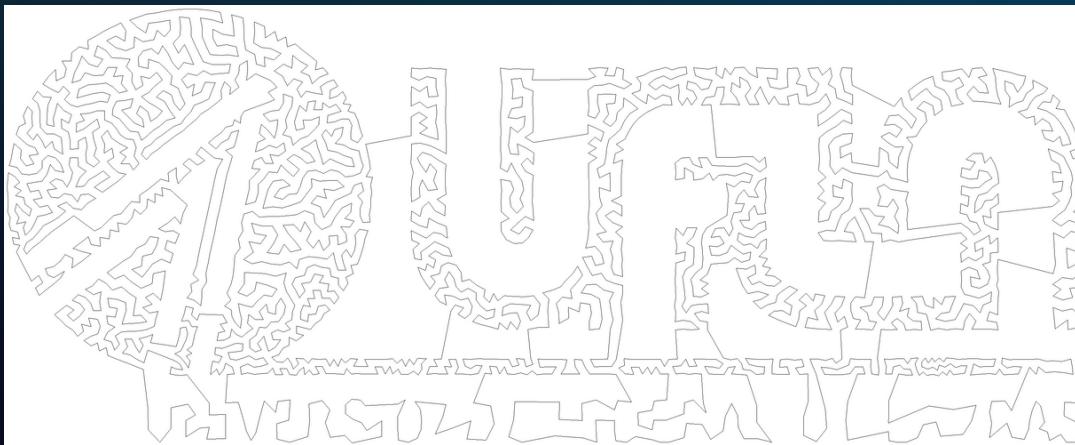
Gerando as imagens



Repositório: [*tsp-art-python*](#), do cientista Matthew Mack.

Definir a abordagem selecionada para geração da solução e implementar as heurísticas propostas no main do arquivo **draw-tsp-path.py**.

```
C: > Users > dpcdti > Desktop > GCC218 > Trabalho > AnaLarrisaTatianaTspArt > draw-tsp-path.py > ORIGINAL_IMAGE
141
142     def main():
143         """Entry point of the program."""
144         print('\n Qual algoritmo deseja executar? \n '
145             '1 - TSP Art (Original) \n '
146             '2 - Vizinho Mais Proximo \n '
147             '3 - Vizinho Mais Proximo + 2OPT \n '
148             '4 - Random + 2OPT \n ')
149         opc = int(input("Selecione: "))
150
```



Implementações e resultados



TSP Art Original

Está é a opção 1 do menu.

Não houve alteração do código proposto por Mack, estruturado em 5 passos, a saber:



Passo 1/5: Create data model



Passo 2/5: Computing distance matrix



Passo 3/5: Setting an initial solution



Passo 4/5: Solving



Passo 5/5: Drawing the solution

```
users > dpcdti > Desktop > GCC218 > Trabalho > tsp-art-python-master > draw-tsp-path.py

def main():
    """Entry point of the program."""
    # Instantiate the data problem.
    print("Step 1/5: Initialising variables")
    data = create_data_model()

    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)
    print("Step 2/5: Computing distance matrix")
    distance_matrix = compute_euclidean_distance_matrix(data['locations'])

    # Setting first solution heuristic.
    print("Step 3/5: Setting an initial solution")
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

    # Solve the problem.
    print("Step 4/5: Solving")
    solution = routing.SolveWithParameters(search_parameters)

    # Print solution on console.
    if solution:
        #print_solution(manager, routing, solution)
        print("Step 5/5: Drawing the solution")
        routes = get_routes(solution, routing, manager)
        draw_routes(data['locations'], routes)
    else:
        print("A solution couldn't be found :(")
```

Vizinho mais próximo

Está é a opção 2 do menu.

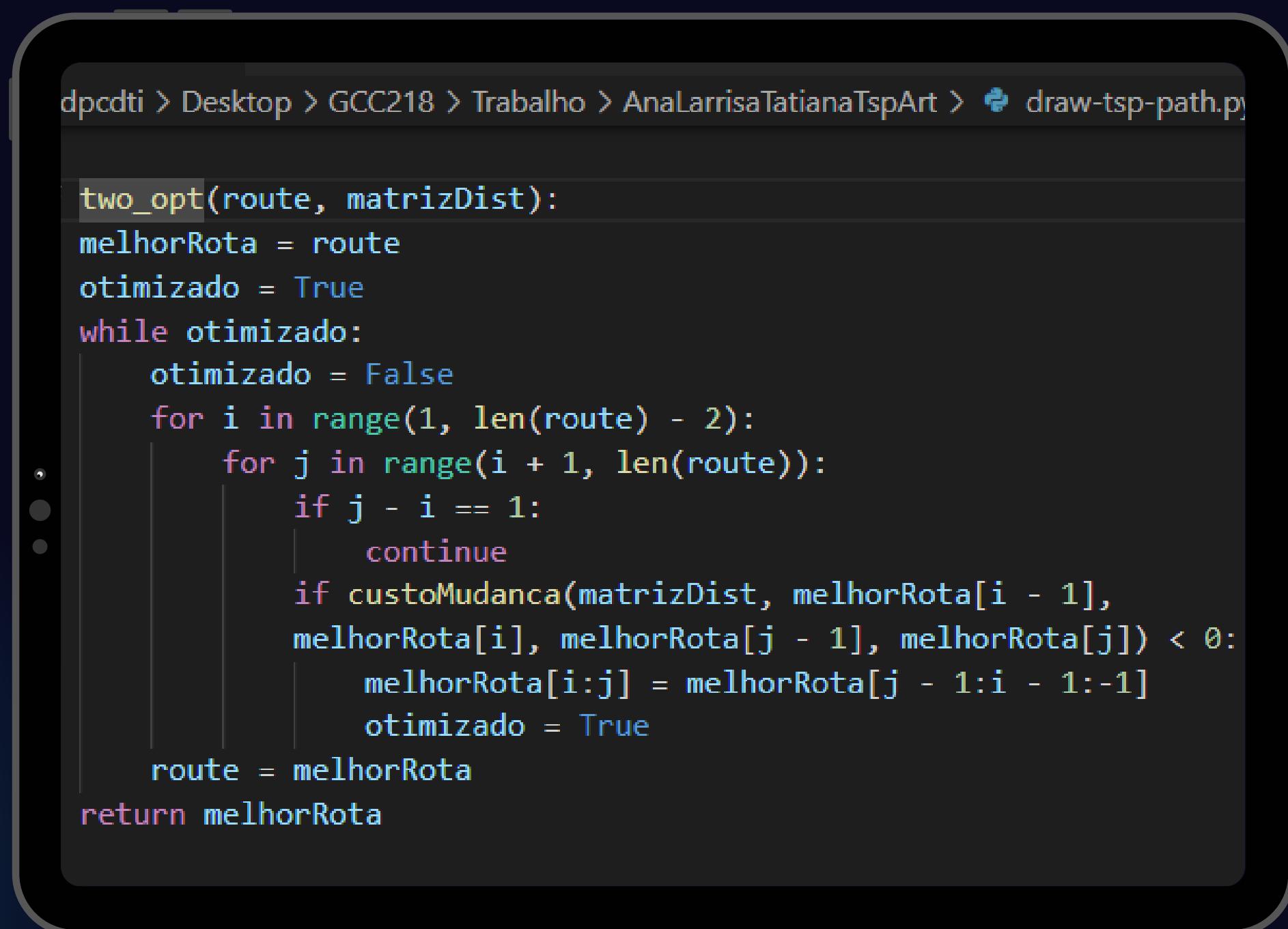
A heurística selecionada pelo grupo tem por objetivo criar uma rota, adicionando repetidamente arestas com menos peso que não conduzam a um nó já visitado anteriormente.

```
C: > Users > dpcdti > Desktop > GCC218 > Trabalho > AnaLarrisaTatianaTspArt >
97
98     def vizinhoMaisProximo(matrizDist, v):
99         visitado = [v]
100        rota = [v]
101
102        while len(visitado) != len(matrizDist[v]):
103            minDist = float('Inf')
104            for x in matrizDist[v]:
105                if x not in visitado:
106                    d = matrizDist[v][x]
107                    if d < minDist :
108                        vertice_visitado = x
109                        minDist = d
110
111            visitado.append(vertice_visitado)
112            rota.append(vertice_visitado)
113            v = vertice_visitado
114            rota.append(0)
115
116
```

Vizinho mais próximo + 2-opt

Está é a opção 3 do menu.

A abordagem baseada em busca local objetiva diminuir a distância total percorrida por meio da deleção de arestas não adjacentes e criação de duas novas. Repete-se este processo para todos os pares de aresta até encontrar a melhor troca.



```
dpcdti > Desktop > GCC218 > Trabalho > AnaLarrisaTatianaTspArt > draw-tsp-path.py

two_opt(route, matrizDist):
    melhorRota = route
    otimizado = True
    while otimizado:
        otimizado = False
        for i in range(1, len(route) - 2):
            for j in range(i + 1, len(route)):
                if j - i == 1:
                    continue
                if custoMudanca(matrizDist, melhorRota[i - 1],
                    melhorRota[i], melhorRota[j - 1], melhorRota[j]) < 0:
                    melhorRota[i:j] = melhorRota[j - 1:i - 1:-1]
                    otimizado = True
        route = melhorRota
    return melhorRota
```

Random + 2-opt

Está é a opção 4 do menu.

Enquanto a opção 3 empregou a heurística do vizinho mais próximo, esta utilizou uma rota criada randomicamente, a partir da lista contendo a quantidade de pontos gerados.

Então, a nova rota foi submetida à heurística de melhoria 2-opt, permitindo comparar o impacto da rota de entrada no resultado final.

```
C: > Users > dpcdti > Desktop > GCC218 > Trabalho
135
136     # import random
137
138     #Gera uma rota aleatório
139     def randomRoute(n):
140         routes = list(range(n))
141         random.shuffle(routes)
142
143         return routes
```

Resultados



Comparação das imagens geradas para cada number_point

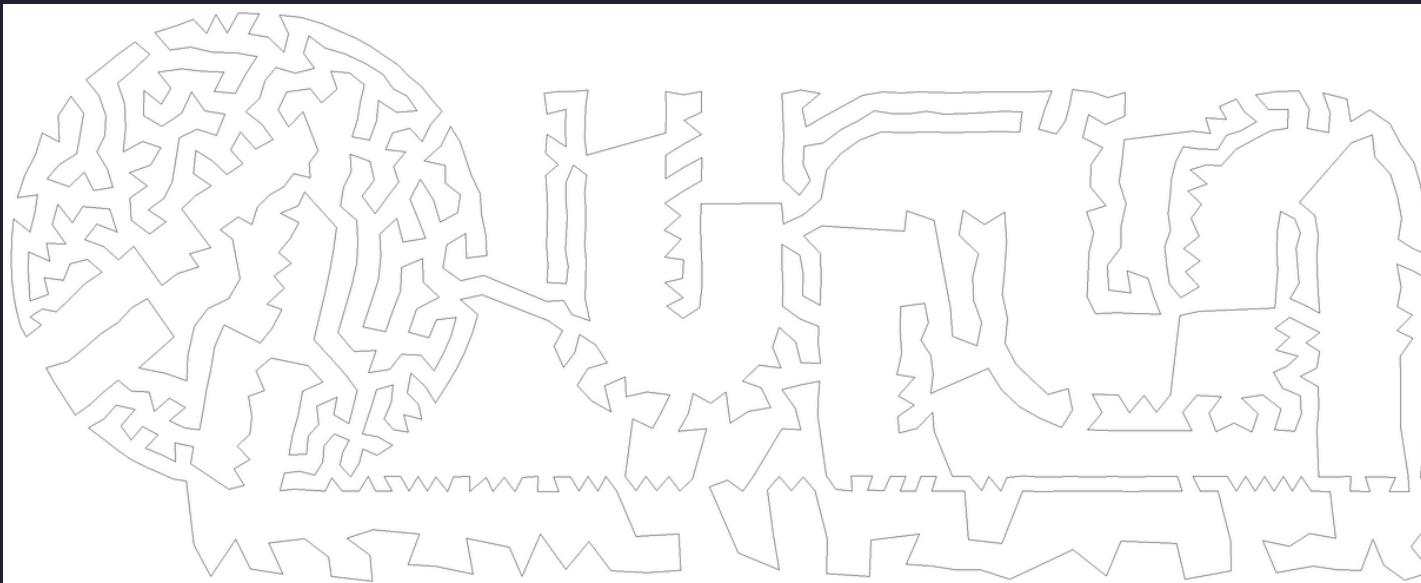


Comparação das distâncias percorridas para cada rota

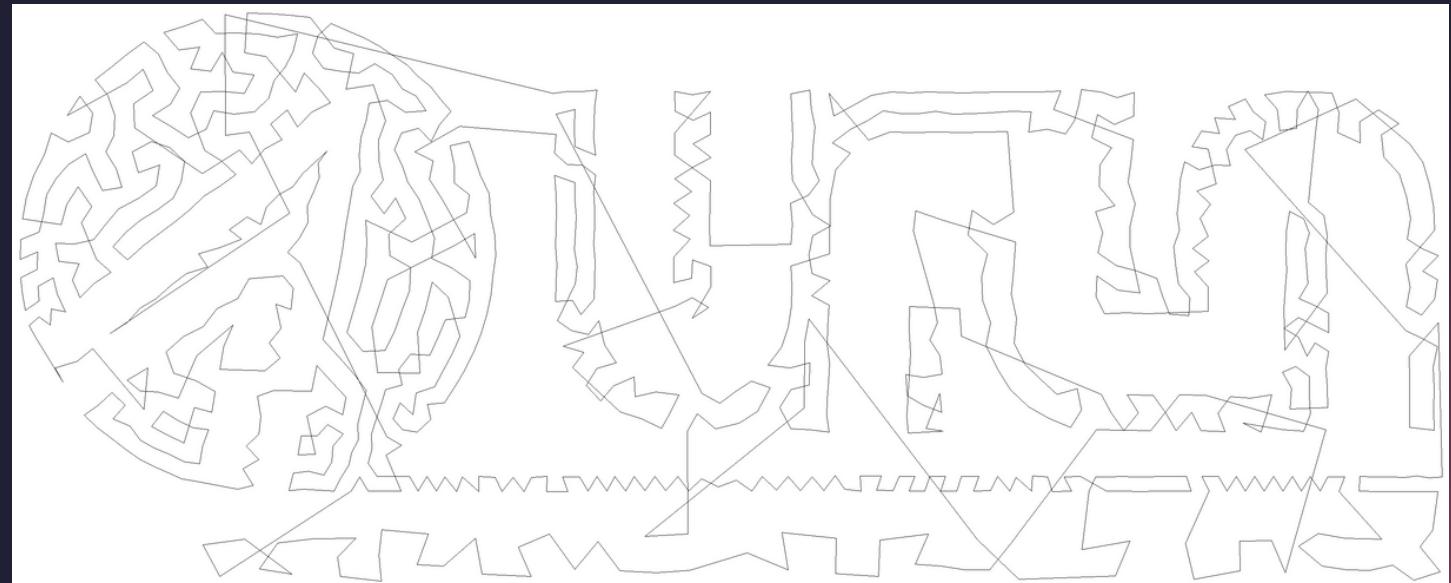


Comparação do desvio de cada abordagem implementada frente ao TSP Art

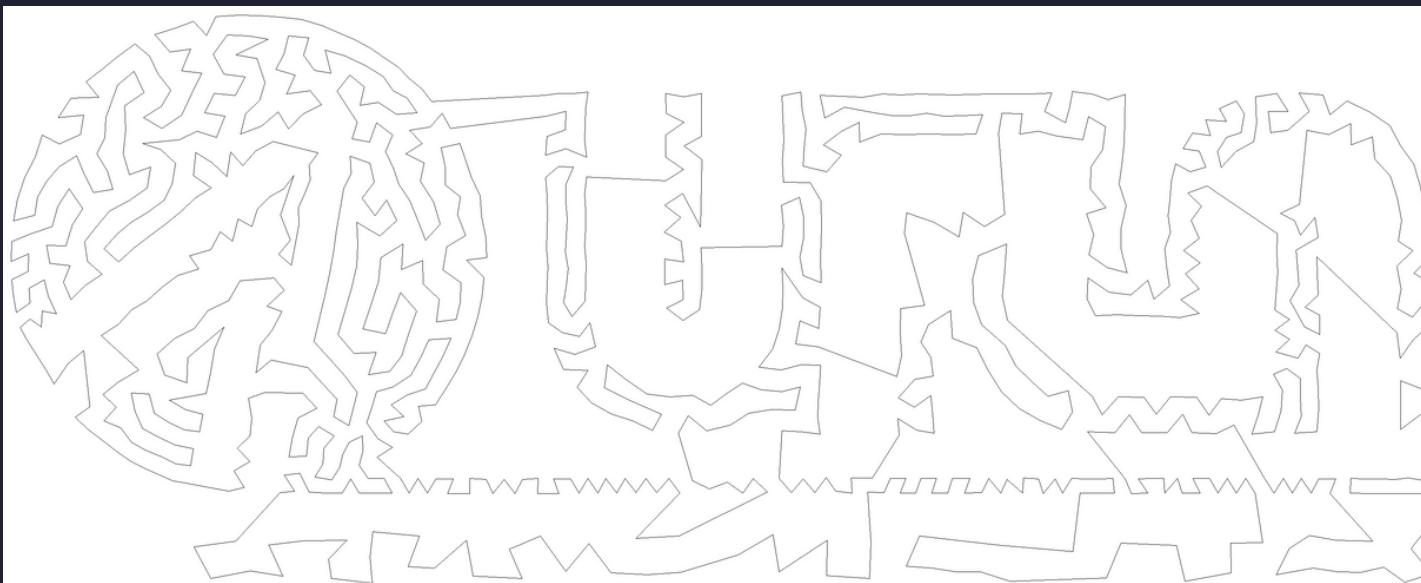
Imagens - 1024



TSP Art



Vizinho mais próximo

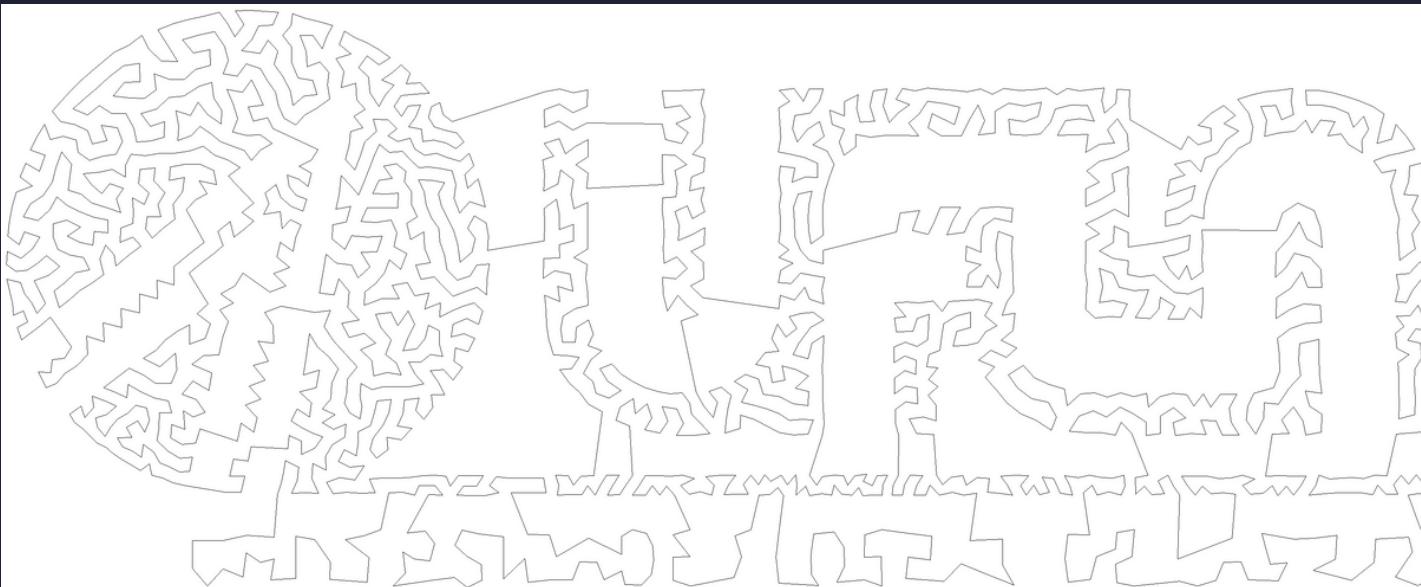


Vizinho + 2-opt



Random + 2-opt

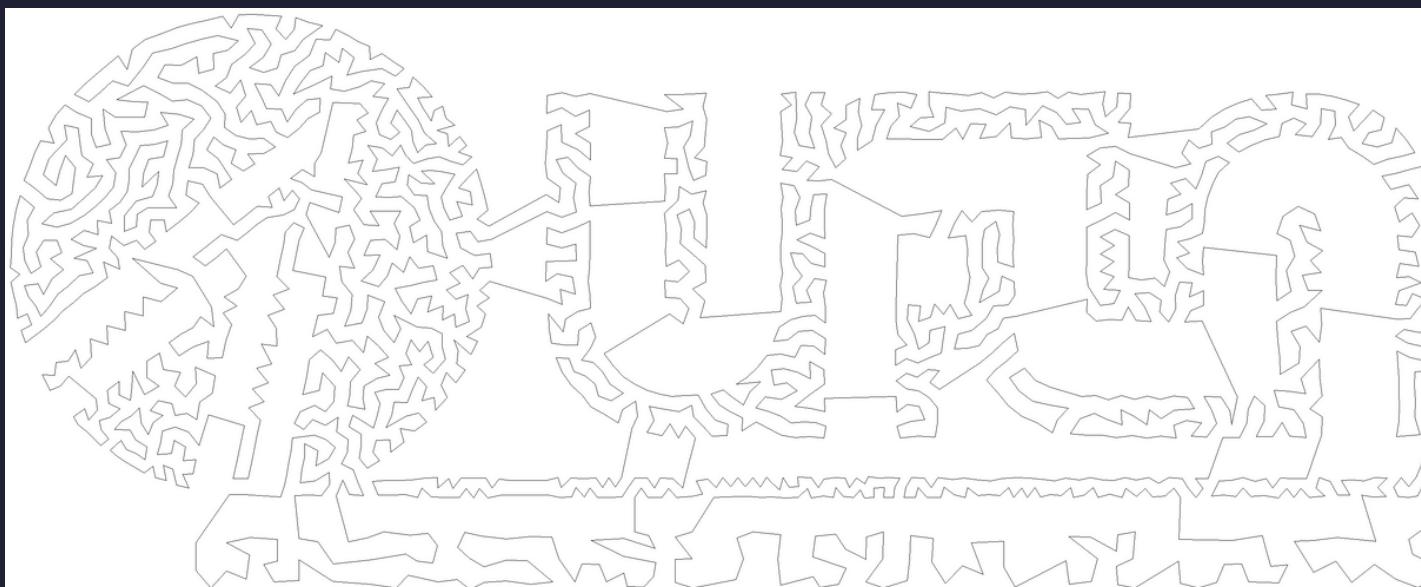
Imagens - 2048 pontos



TSP Art



Vizinho mais próximo

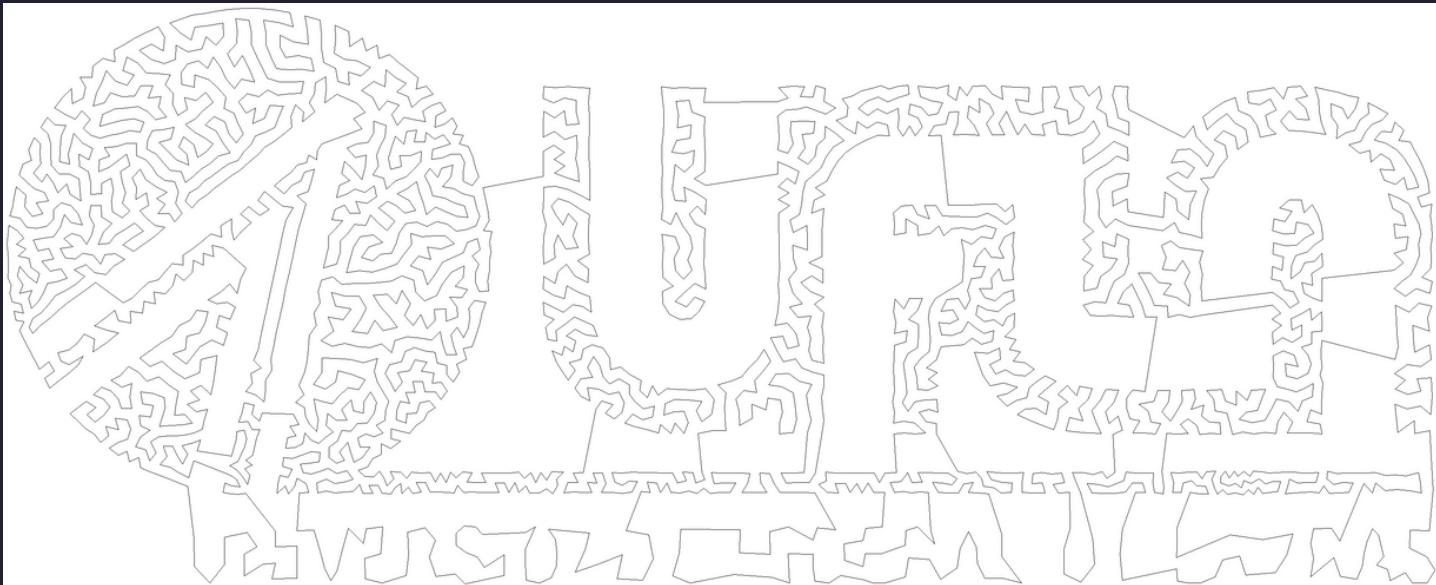


Vizinho + 2-opt

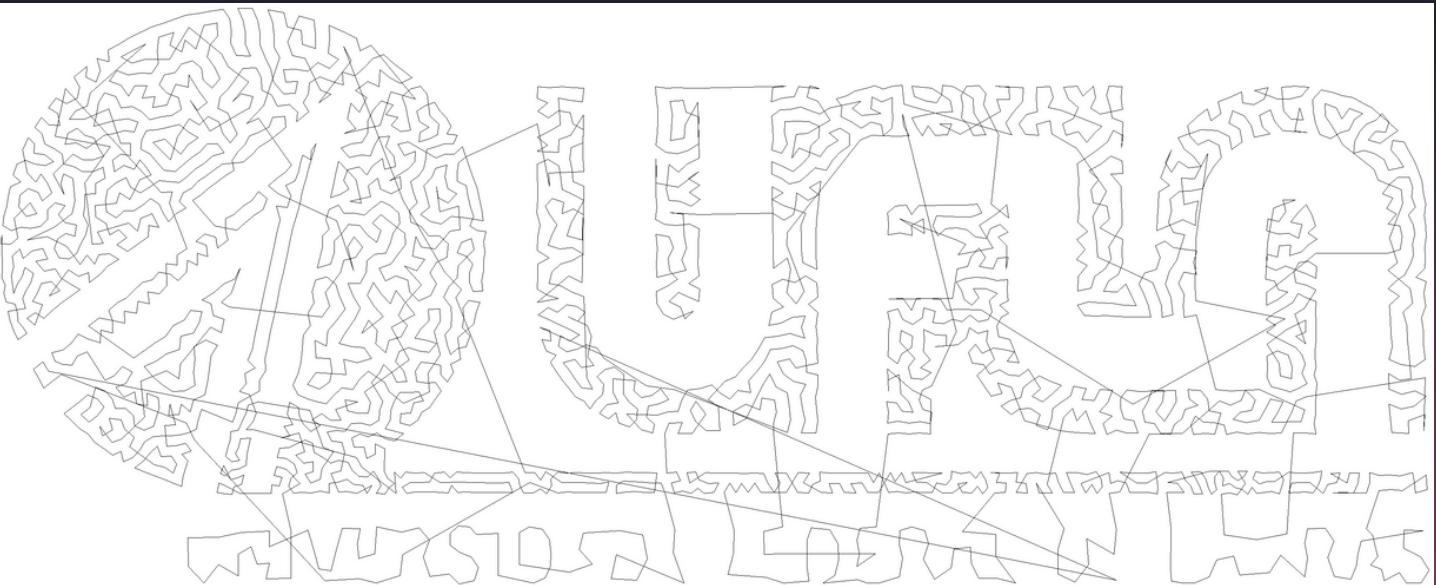


Random + 2-opt

Imagens - 3072 pontos



TSP Art



Vizinho mais próximo



Vizinho + 2-opt



Random + 2-opt

Resultados

Observa-se que a solução do Google OR-Tools apresenta a menor distância se comparada as demais abordagens. O pior resultado foi a heurística construtiva de inserção do vizinho mais próximo.

TSP Art

PONTOS -> DISTÂNCIA

1024 -> 46968

2048 -> 66023

3072 -> 80061

Vizinho mais próximo

PONTOS -> DISTÂNCIA

1024 -> 55994

2048 -> 80538

3072 -> 96434

Vizinho 2-opt

PONTOS -> DISTÂNCIA

1024 -> 48394

2048 -> 68058

3072 -> 81236

Random 2-opt

PONTOS -> DISTÂNCIA

1024 -> 50768

2048 -> 70053

3072 -> 84033

Resultados

Mediante o cálculo do desvio - $[(f(Y) - f(X))/f(X)] * 100\%$, é possível perceber que todas as abordagens apresentaram rotas com distâncias superiores a opção 1, sendo a menor 1,47% e a maior 21,98%.

TSP Art

PONTOS → DISTÂNCIA

1024 → 46968

2048 → 66023

3072 → 80061

Vizinho mais próximo

PONTOS → DESVIO

1024 → 19,22%

2048 → 21,98%

3072 → 20,45%

Vizinho 2-opt

PONTOS → DESVIO

1024 → 3,04%

2048 → 3,08%

3072 → 1,47%

Random 2-opt

PONTOS → DESVIO

1024 → 8,09%

2048 → 6,10%

3072 → 4,96

Considerações finais

- Todas as abordagens são eficazes em criar uma imagem.
- A imagem gerada fica mais nítida com o aumento do número de pontos (1024, 2028 e 3072).
- A abordagem original de Mack, utilizando o Google OR-Tools é a mais eficiente dentre as opções testadas.
- O 2-opt se mostrou uma boa opção como heurística de melhoria.

TSP Art

Melhor solução

Vizinho + 2-opt

Melhor opção implementada

Random + 2-opt

Terceira melhor opção

Vizinho + próximo

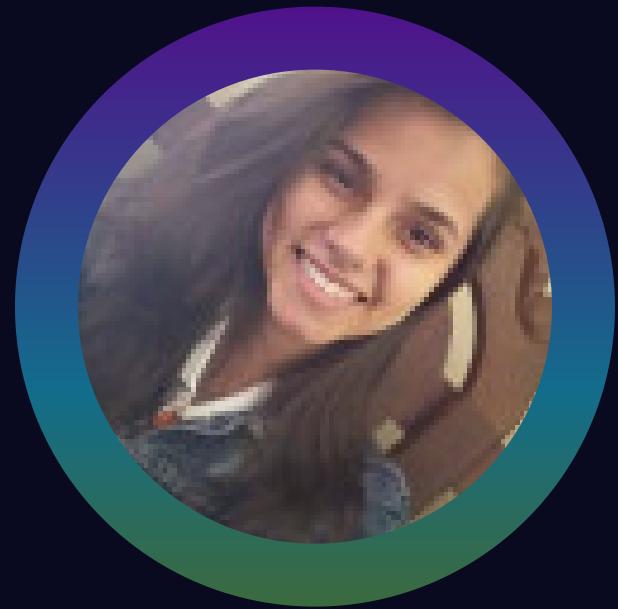
Pior resultado

Grupo



ANA TORRES

Ciência da
Computação



LARISSA OLIVEIRA

Ciência da
Computação



TATIANA CANTELLE

Sistemas de
informação