Introdução à Inteligência Artificial - Trabalho 2

Alunos
Gabriel Toschi de Oliveira - Nº USP 9763039
Marcos Vinicius Volpato - Nº USP 9364872

Roteiro da apresentação

- Introdução do trabalho
- Detalhes da implementação
- Resultados dos testes
- Discussões dos resultados

Introdução do trabalho

- Estudar e testar eficiência de buscas em grafos
  - DFS, BFS, Best-First, A, A*
- Grafos-knn como objetos de estudo
- Coletar tempo de execução e caminhos percorridos

Detalhes da implementação

1. Criação de TADs para grafos e grafos-knn
2. Implementação dos algoritmos de busca
3. Geração das imagens dos grafos e caminhos
4. Automatização dos testes realizados

Detalhes da implementação

1. **Criação de TADs para grafos e grafos-knn**
2. Implementação dos algoritmos de busca
3. Geração das imagens dos grafos e caminhos
4. Automatização dos testes realizados

```
class Graph:
    def __init__(self):
        self.graphDict = {}
        self.gPrinter = GraphPrinter()

    def vertices(self):
        return list(self.graphDict.keys())

    def edges(self):
        edges = []

        for source in self.graphDict:
            for target in self.graphDict[source]:
                edges.append((source, target))

        return edges

    def neighbors(self, vertex):
        return self.graphDict[vertex]
```

Detalhes da implementação

1. **Criação de TADs para grafos e grafos-knn**
2. Implementação dos algoritmos de busca
3. Geração das imagens dos grafos e caminhos
4. Automatização dos testes realizados

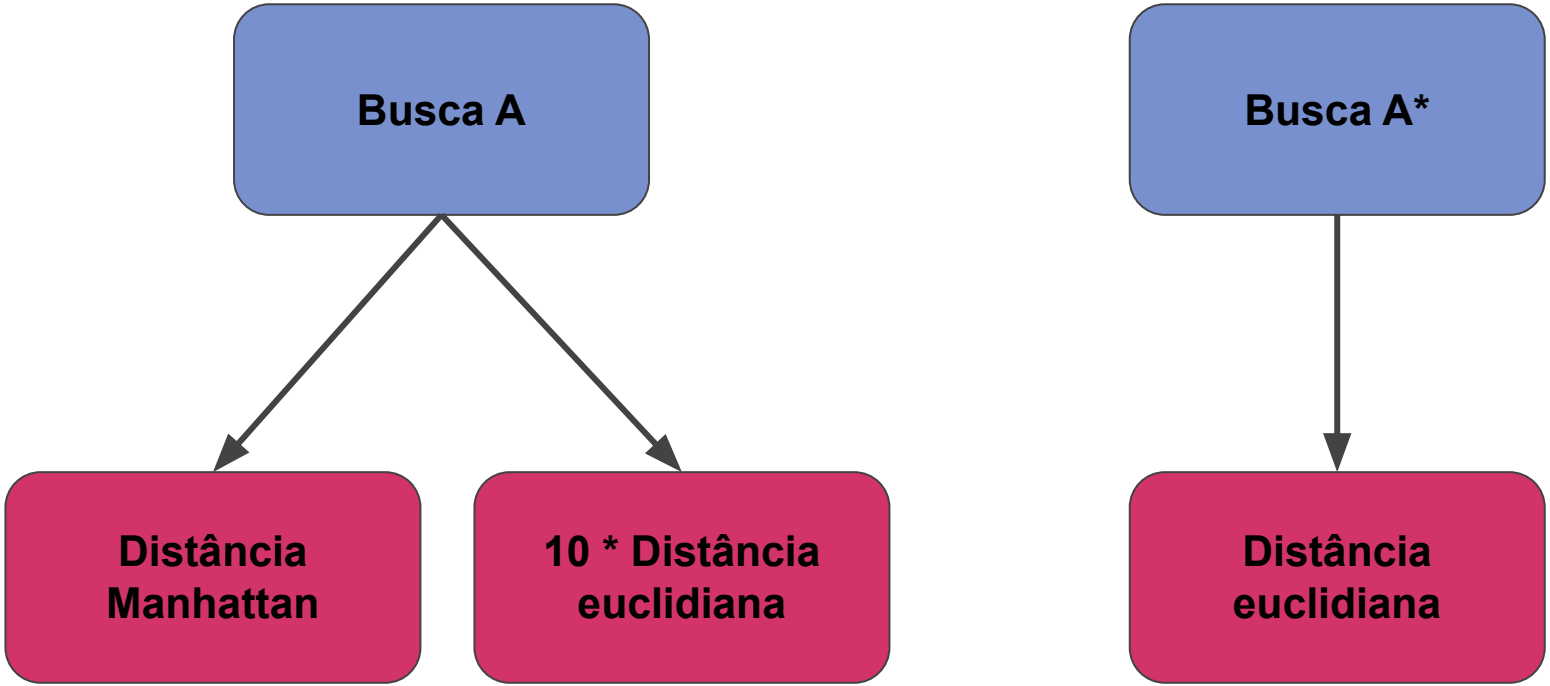
```python
class KNNGraph(Graph):
    def __init__(self, v, k):
        super().__init__()

        vertices = []

        while len(vertices) < v:
            vertex = (random.randint(0, v), random.randint(0, v))

            if vertex not in vertices:
                self.addVertex(vertex)
                vertices.append(vertex)

        for v in vertices:
            distances = []

            for v2 in vertices:
                if v != v2:
                    distances.append((v2, distance(v, v2)))

            distances.sort(key = lambda x: x[1])

            for i in range(k):
                self.addEdge(v, distances[i][0])
```

Detalhes da implementação

1. Criação de TADs para grafos e grafos-knn
2. **Implementação dos algoritmos de busca**
3. Geração das imagens dos grafos e caminhos
4. Automatização dos testes realizados

```python
def BFS(graph: Graph.Graph, start, end):
    visited = set()

    parent = dict()
    parent[start] = None

    queue = []
    queue.append(start)
    visited.add(start)

    while queue:
        current = queue.pop(0)

        if current == end:
            break

        for v in graph.neighbors(current):
            if v not in visited:
                queue.append(v)
                visited.add(v)
                parent[v] = current

    return parent
```

Heurísticas da busca A e A*



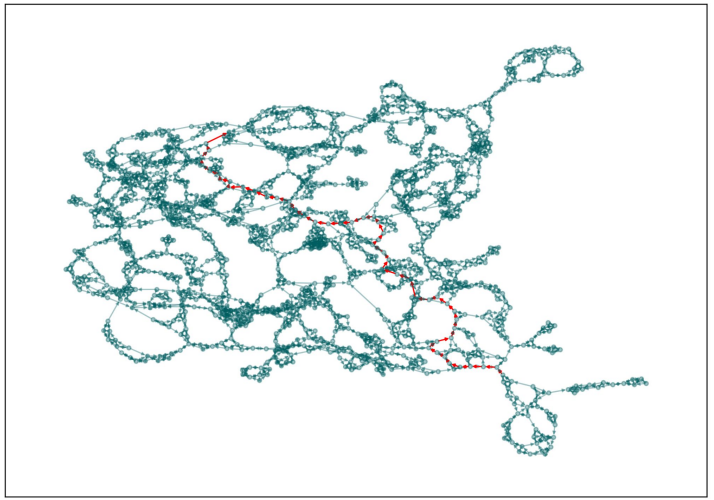
Detalhes da implementação

1. Criação de TADs para grafos e grafos-knn
2. **Implementação dos algoritmos de busca**
3. Geração das imagens dos grafos e caminhos
4. Automatização dos testes realizados

```
algorithms = {
    'DFS': DFSIterative,
    'BFS': BFS,
    'Best-First': bestFirst,
    'A*': aStar,
    'A (Manhattan)': aManhattan,
    'A (10 * Euclidian)': aEuclidian,
}
```

Detalhes da implementação

1. Criação de TADs para grafos e grafos-knn
2. Implementação dos algoritmos de busca
3. **Geração das imagens dos grafos e caminhos**
4. Automatização dos testes realizados



Detalhes da implementação

1. Criação de TADs para grafos e grafos-knn
2. Implementação dos algoritmos de busca
3. Geração das imagens dos grafos e caminhos
4. **Automatização dos testes realizados**

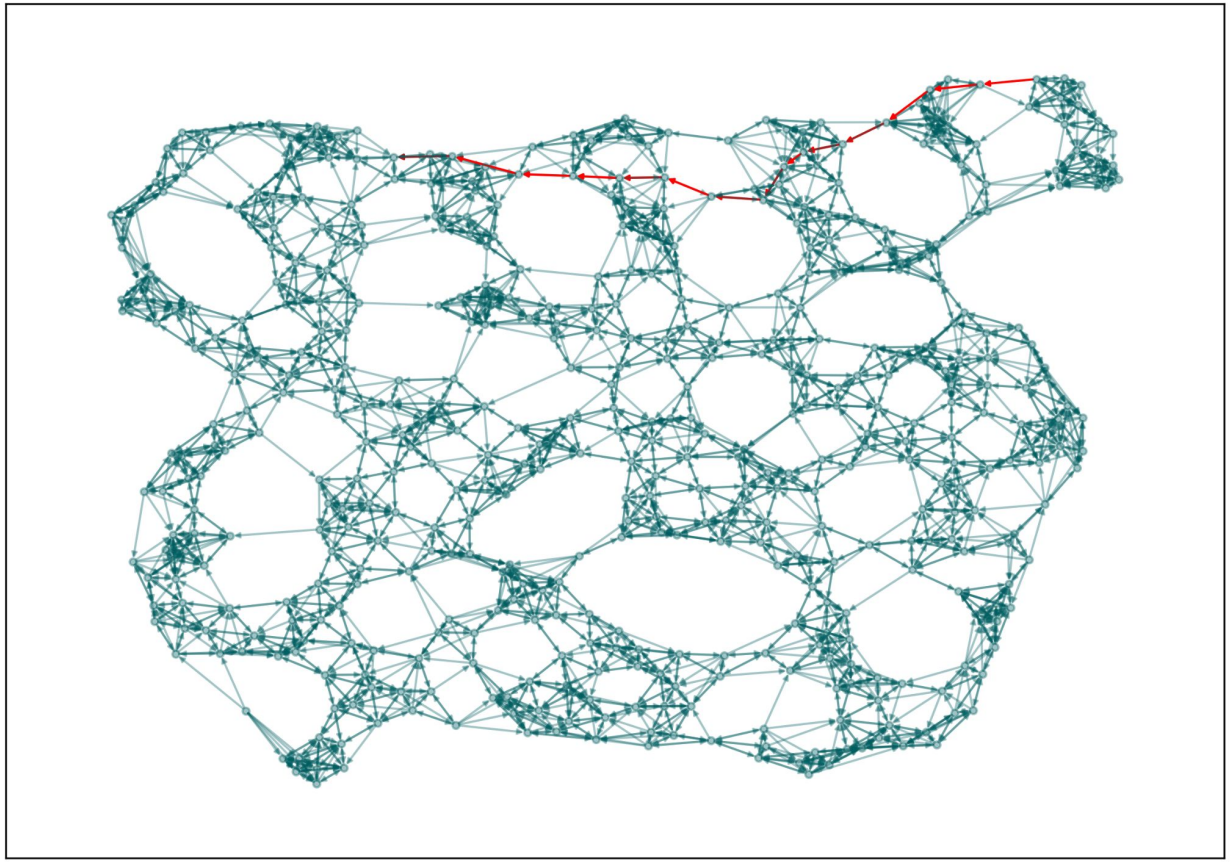
```
for exec in range(EXECUTIONS):
    start = time.time()
    lastParent = algorithms[algName](graph, startNode, endNode)
    end = time.time()

    executionTime = executionTime + (end - start)

executionTime = executionTime / EXECUTIONS

path = getPathByParents(lastParent, endNode)
```

Resultados coletados

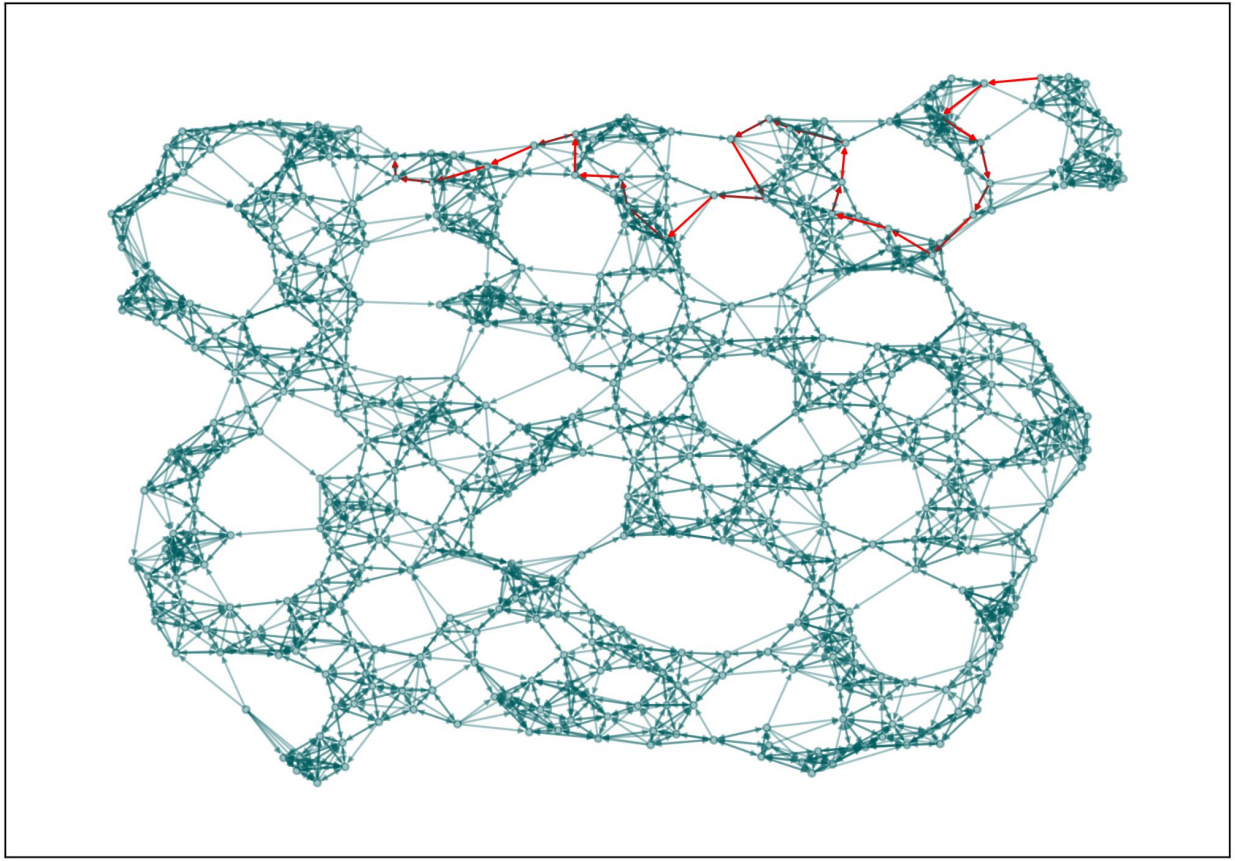
- 9 cenários de teste feitos a partir da variação do número de vértices e da quantidade de arestas entre eles
  - Número de vértices (**v**): 500, 1250 e 2500 vértices
  - Número de arestas (**k**): 3, 5 e 7 arestas por vértice
- 6 algoritmos a serem testados:
  - Busca em largura (BFS)
  - Busca em profundidade (DFS)
  - Busca best-first
  - Busca A (usando a distância Manhattan
  - Busca A (usando 10x a distância euclidiana)
  - Busca A* (usando a distância euclidiana)
- Dados coletados: tempo de execução e caminhos

Resumo dos resultados

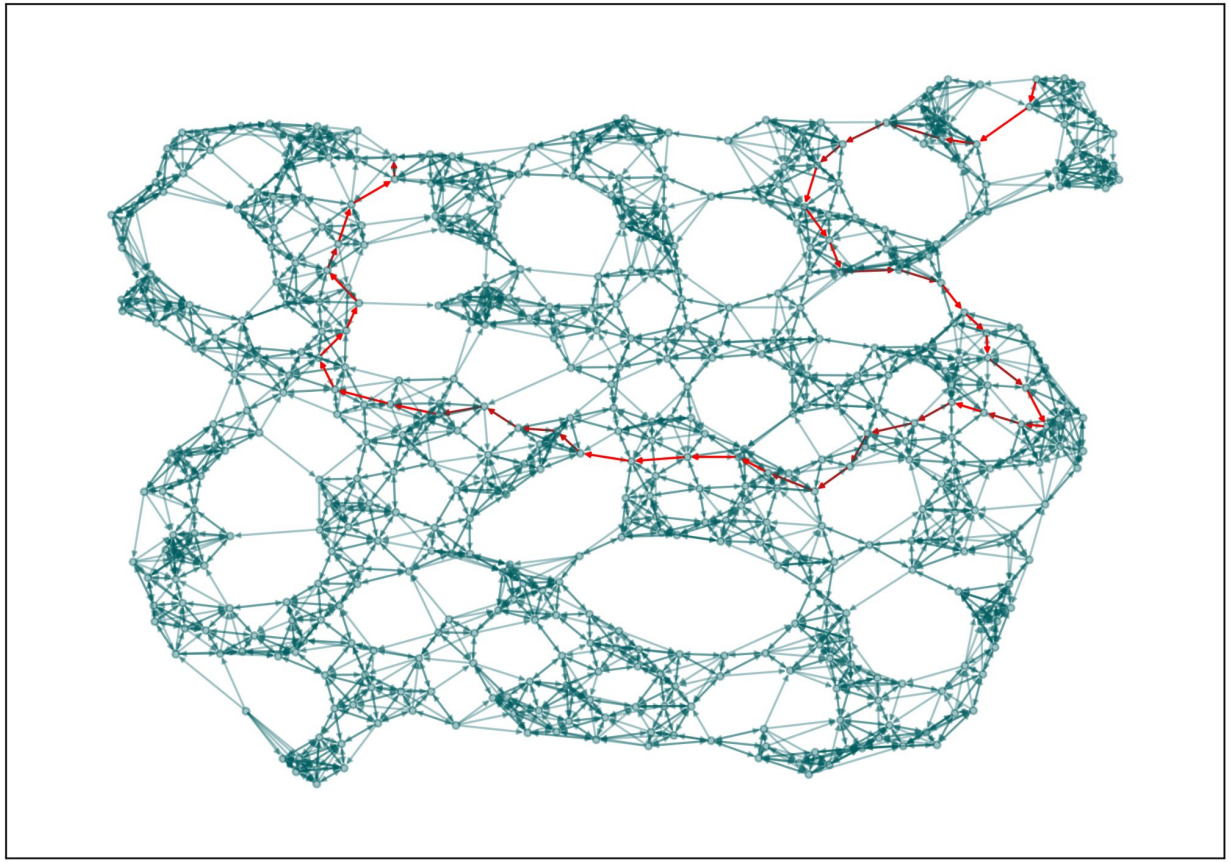
	DFS	BFS	Best-First	A*	A (Manhattan)	A (10*Euclidiana)
v = 500, k = 3	0,03924 ms / 31 v	0,10040 ms / 20 v	0,63419 ms / 42 v	0,43998 ms / 20 v	0,25554 ms / 20 v	0,19932 ms / 20 v
v = 500, k = 5	0,00024 ms / 24 v	0,08626 ms / 11 v	0,00087 ms / 22 v	0,00021 ms / 11 v	0,00015 ms / 12 v	0,00015 ms / 11 v
v = 500, k = 7	0,36535 ms / 26 v	0,12422 ms / 13 v	0,11824 ms / 43 v	0,44904 ms / 15 v	0,41547 ms / 14 v	0,29220 ms / 15 v
v = 1250, k = 3	0,11563 ms / 64 v	0,16160 ms / 50 v	1,15675 ms / 50 v	1,99203 ms / 50 v	2,33297 ms / 50 v	1,19376 ms / 50 v
v = 1250, k = 5	1,04103 ms / 74 v	0,27194 ms / 13 v	1,25980 ms / 26 v	0,41875 ms / 15 v	0,20795 ms / 15 v	0,20704 ms / 15 v
v = 1250, k = 7	0,04110 ms / 34 v	0,37946 ms / 13 v	4,65307 ms / 26 v	0,40259 ms / 13 v	0,41050 ms / 15 v	0,24700 ms / 13 v
v = 2500, k = 3	0,20208 ms / 53 v	0,16765 ms / 35 v	1,78976 ms / 50 v	1,25460 ms / 35 v	1,16353 ms / 37 v	0,49863 ms / 35 v
v = 2500, k = 5	1,23839 ms / 377 v	1,27205 ms / 50 v	10,77289 ms / 102 v	6,23621 ms / 52 v	0,95705 ms / 52 v	0,77395 ms / 55 v
v = 2500, k = 7	1,34363 ms / 290 v	1,89037 ms / 43 v	8,74752 ms / 94 v	4,68597 ms / 47 v	1,26567 ms / 49 v	1,07546 ms / 57 v



v = 500, k = 7, algoritmo A* (0,44904 ms / 15 v)



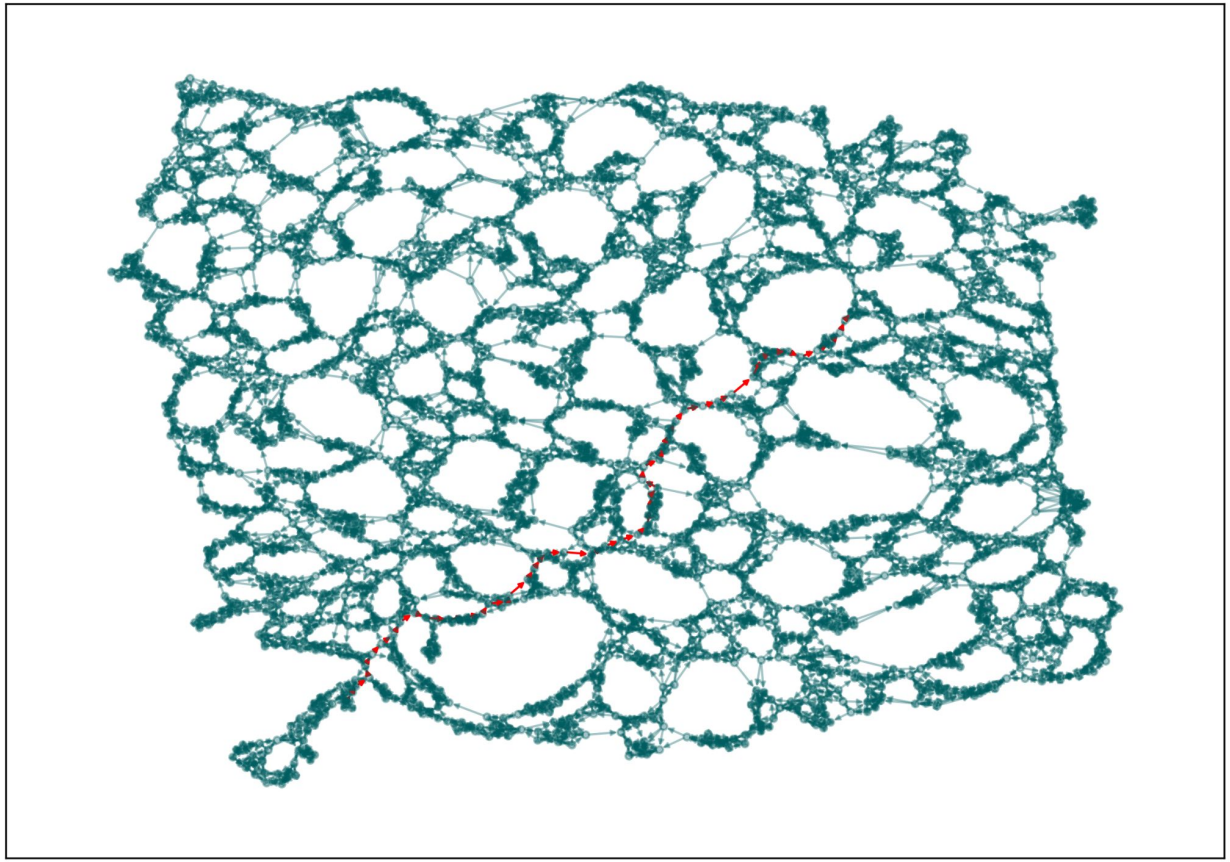
v = 500, k = 7, algoritmo DFS (0,36535 ms / 26 v)



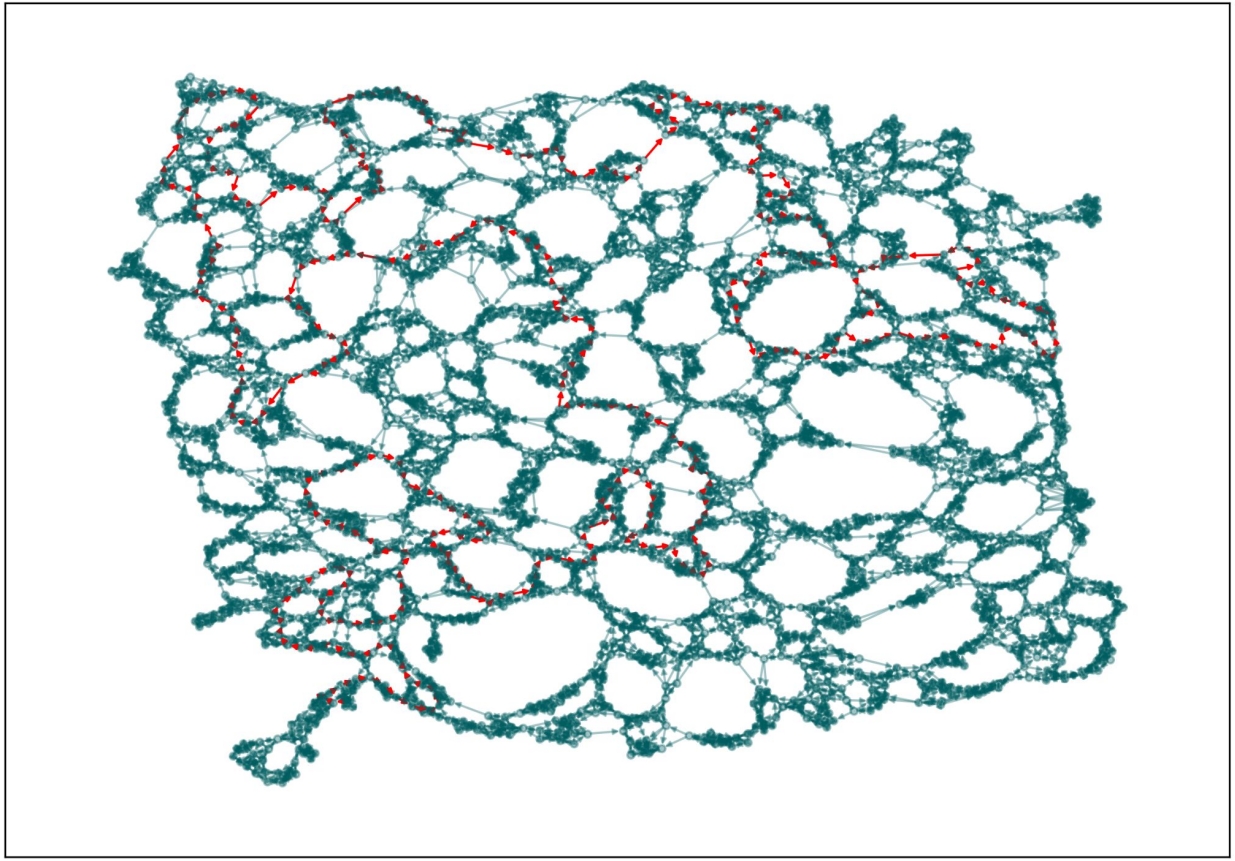
v = 500, k = 7, algoritmo best-first (0,11824 ms / 43 v)

Discussão dos resultados

- DFS
  - melhores tempos de execução em boa parte dos casos
  - caminhos até 3x maiores que os outros algoritmos (para v = 1250 e 2500)
  - relacionado com a natureza do algoritmo



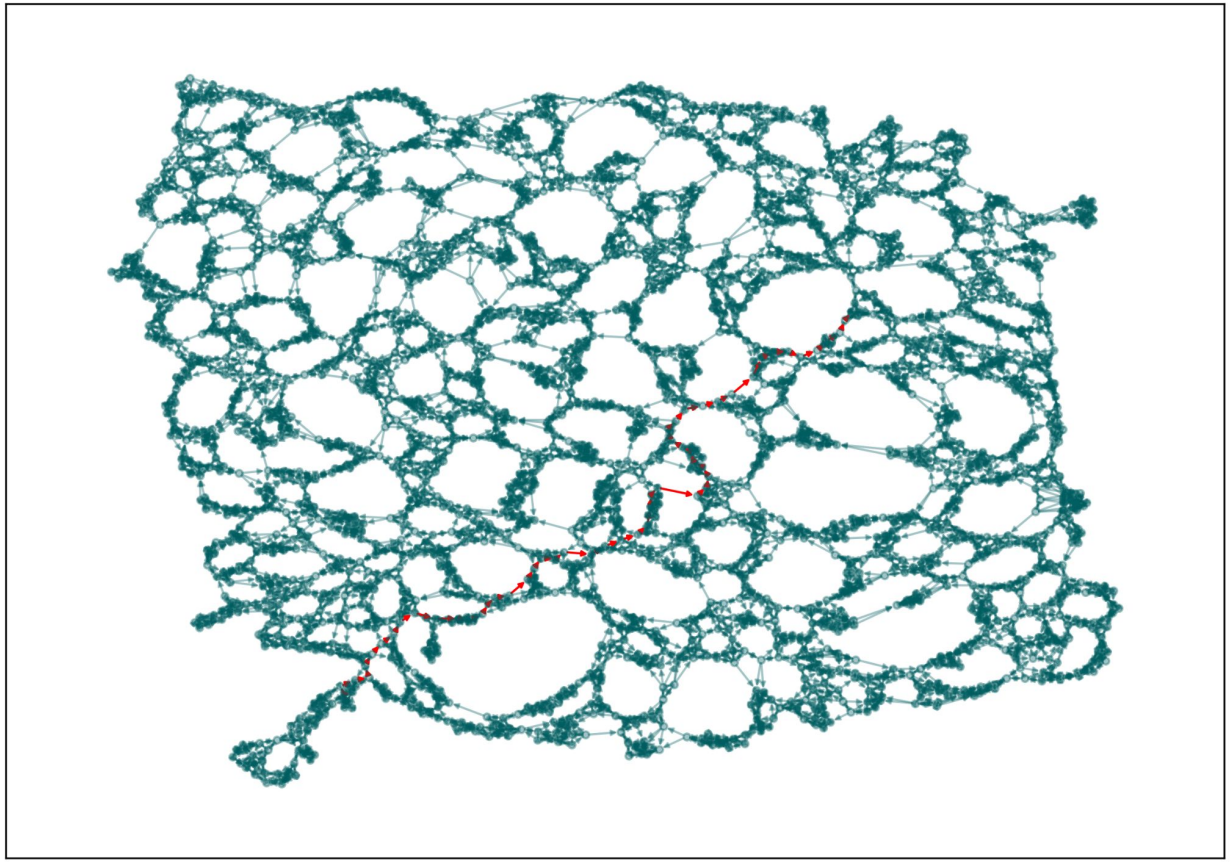
v = 2500, k = 5, algoritmo BFS (1,27205 ms / 50 v)



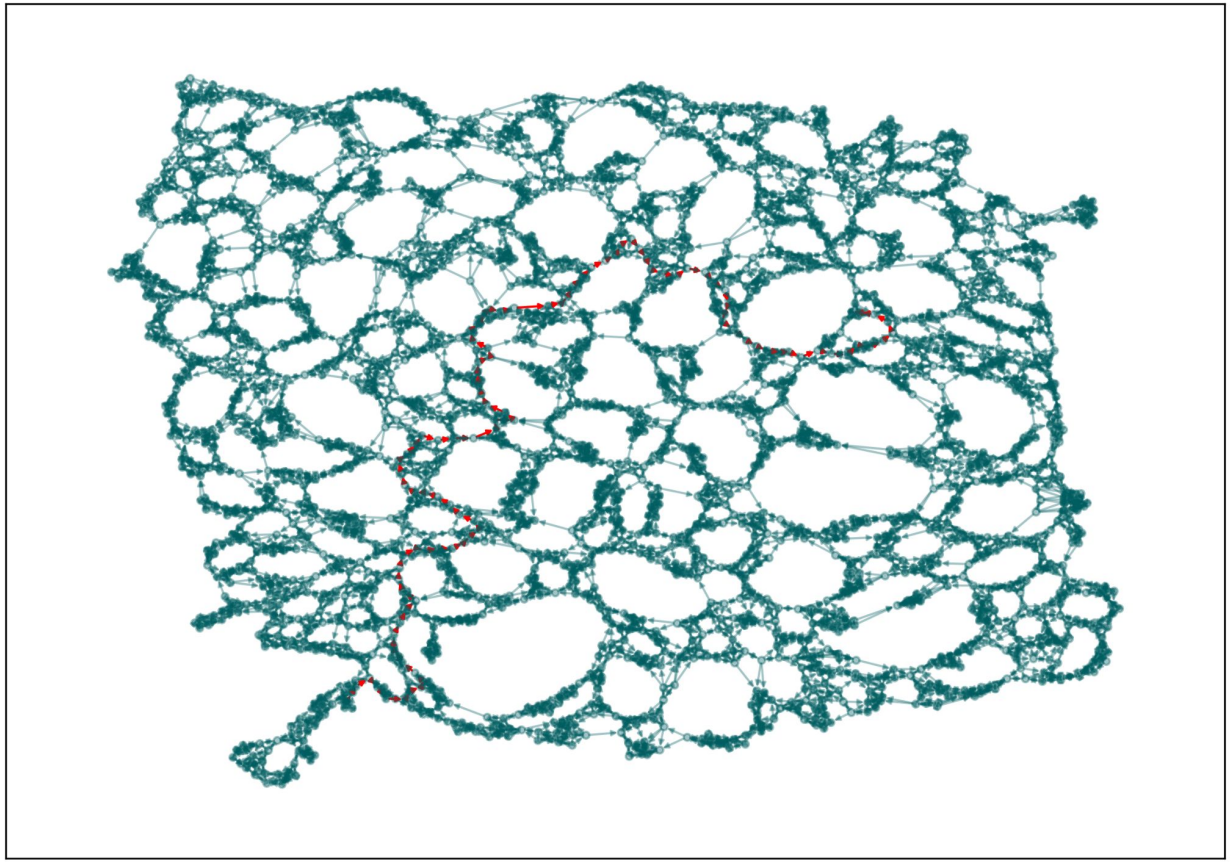
v = 2500, k = 5, algoritmo DFS (1,23839 ms / 377 v)

Discussão dos resultados

- Tempo de execução
  - para poucos vértices, DFS e BFS
  - para muitos vértices, buscas A (Manhattan e 10*Euclidiana)
  - diferenças não tão significativas em alguns casos
- Best-First com o pior desempenho em tempo
  - manutenção de uma lista de prioridades sem heurística
  - o mesmo algoritmo usando heurísticas se mostrou mais eficiente



v = 2500, k = 5, algoritmo A - 10*euclidiana (0,77395 ms / 55 v)



v = 2500, k = 5, algoritmo best-first (10,77289 ms / 102 v)

Discussão dos resultados

- Heurísticas para A e A*
  - distância euclidiana pura (A*) com desempenho sempre pior ou igual
  - distância euclidiana * 10 com desempenho melhor que Manhattan
  - relacionado com a natureza do grafo-knn
  - distância Manhattan talvez seja mais otimista do que se pensa

Obrigado!