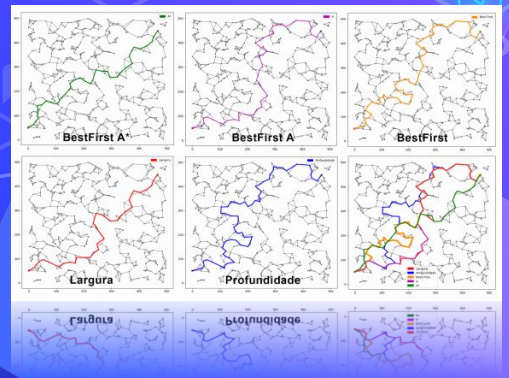
Análise de Algoritmos de Busca em Grafos KNN



Calvin Suzuki de Camargo
Guilherme Soares Silvestre

1.
Grafo KNN

Construção

- *Input*:
  - Número de vértices (*V*) e arestas (*K*)
- Vértices estão contidos em um plano cartesiano *V* x *V*
- Cada *V* deve conectar com *K* nós mais próximos
- Casos especiais necessitam regras mais específicas

Regras

- Cada vértice *V* é fonte de *K* arestas
- Vale a ligação inversa
- Mesmo que um nó já tenha sido fonte *K* arestas, ele pode ser chegada de novas arestas

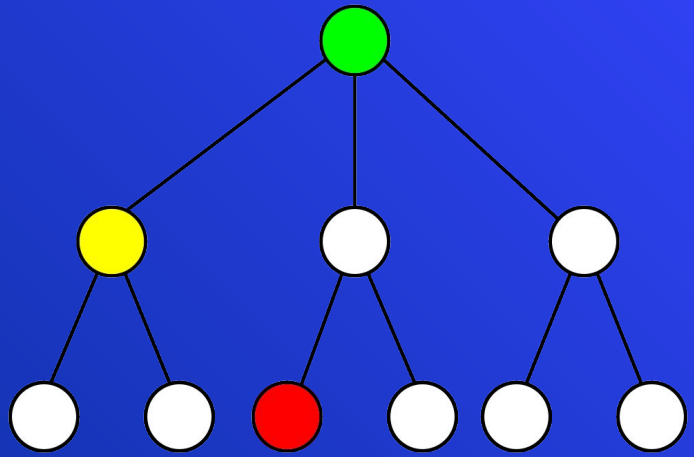
2.
Tipos de busca

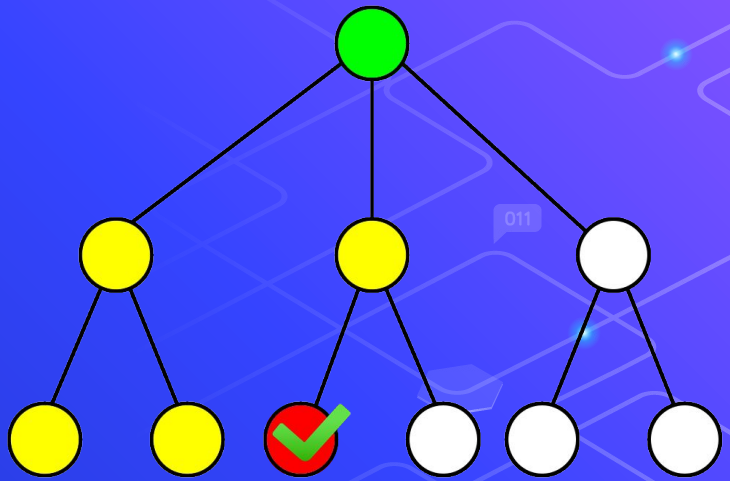
Busca exaustiva (cega)

- Também chamados de algoritmos de busca por **força-bruta** ou busca **não-informada**

- As buscas cegas a serem a avaliadas são:
  - **Busca por profundidade** (*Depth-first)*
  - **Busca por largura** (*Breadth-first*)

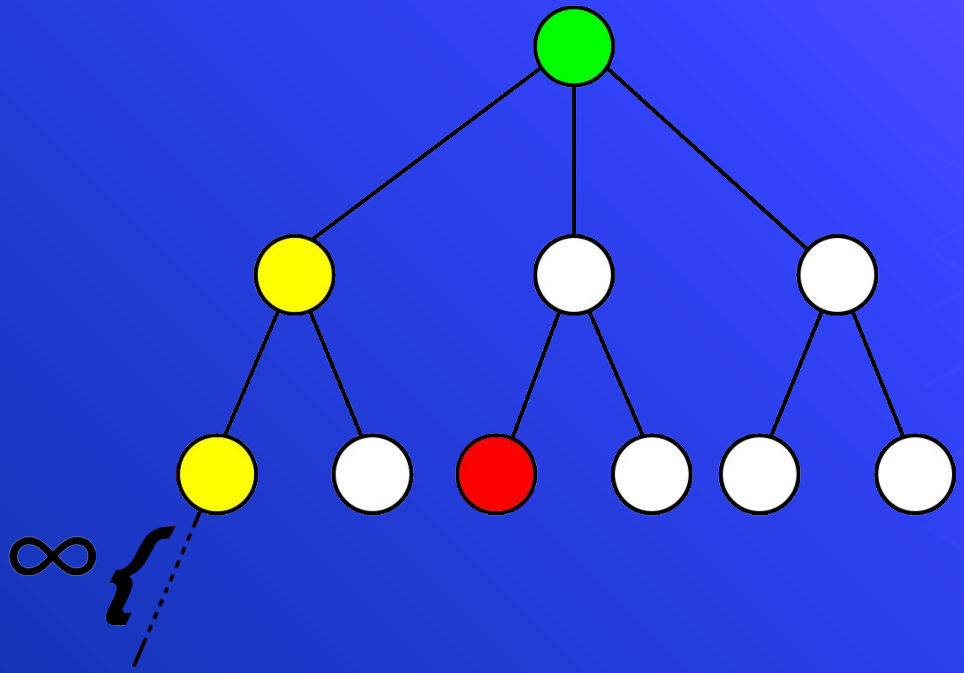
Busca por profundidade *(Depth)*





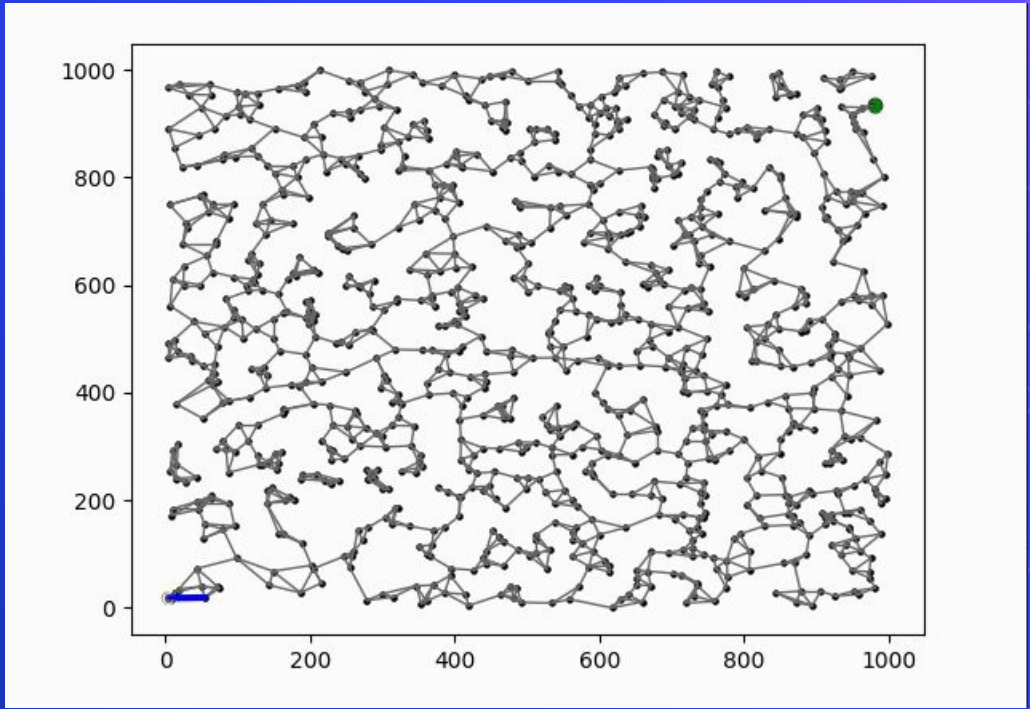
- (*last-in first-out*)

Busca por profundidade *(Depth)*

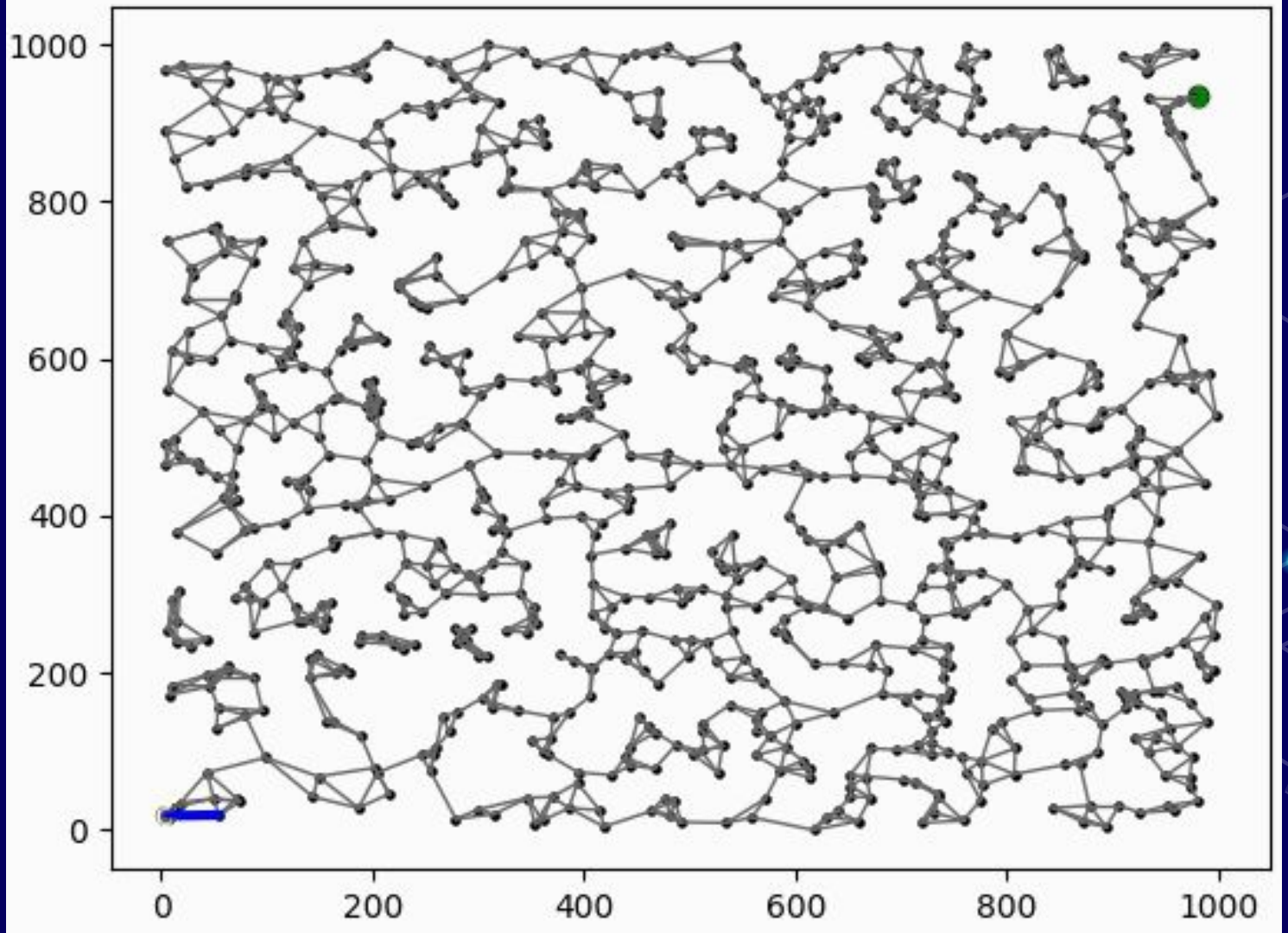


- **Incompleto**: pode ficar preso em um ramo infinito

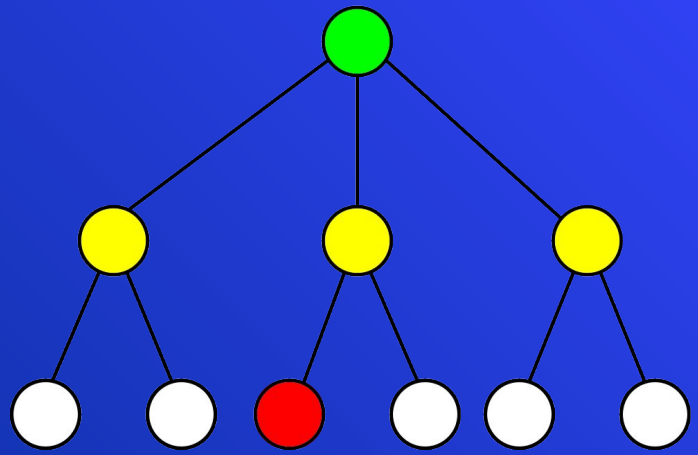
Busca por profundidade *(Depth)*

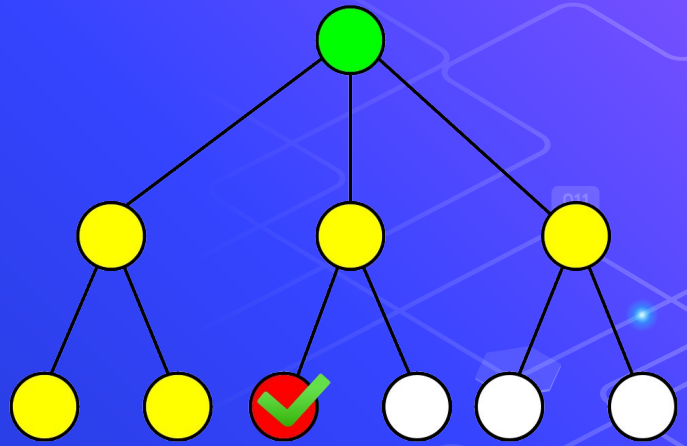


- Não garante o melhor caminho


1000
800
600
400
200
0
0
200
400
600
800
1000

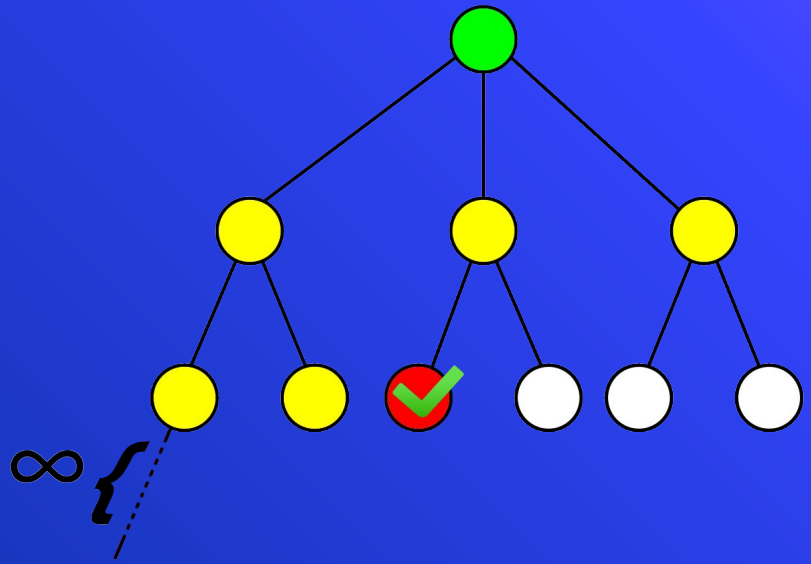
Busca por largura *(Breadth)*





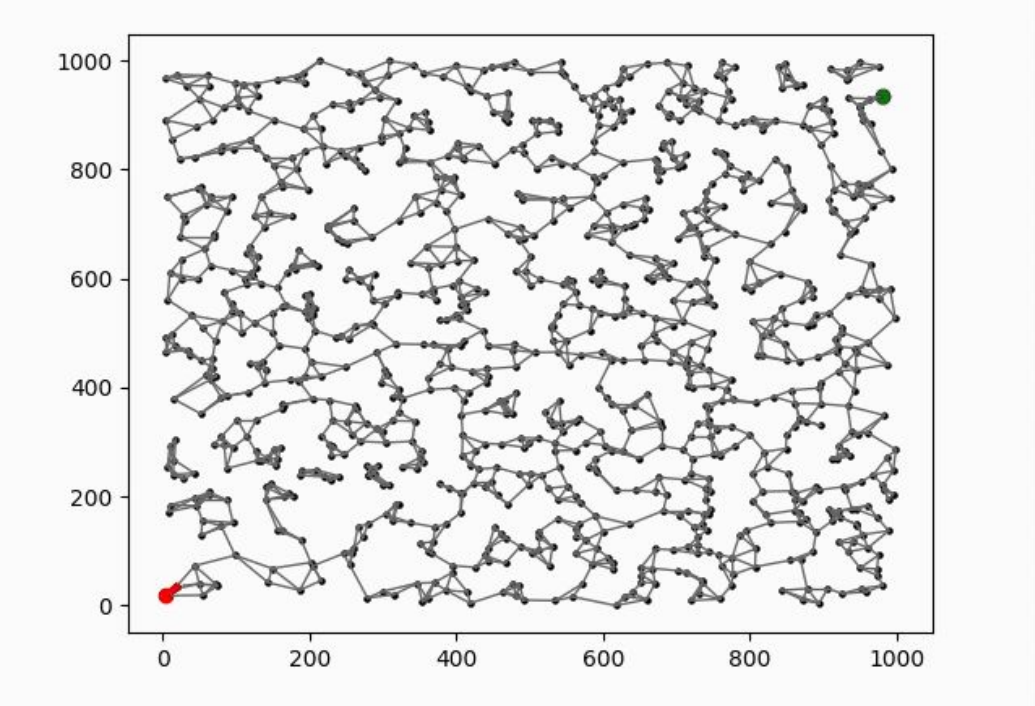
- (*first-in first-out*)

Busca por largura *(Breadth)*

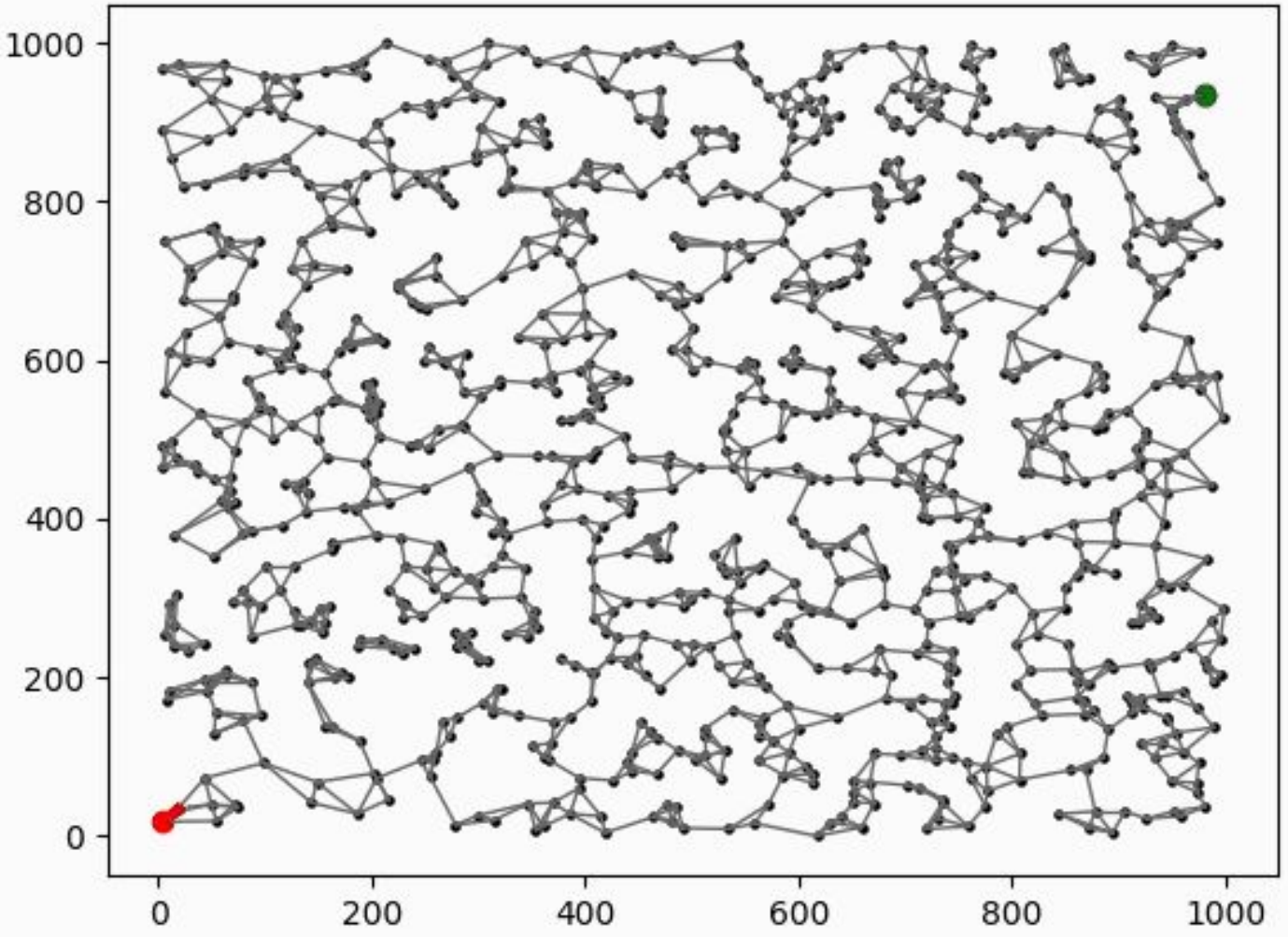


- **Completo**: garante encontrar todas as soluções

Busca por largura *(Breadth)*



Garante encontrar o caminho com menor número de vértices


1000
800
600
400
200
0
0
200
400
600
800
1000

Busca heurística

- É um algoritmo de busca que leva em conta alguma **informação auxiliar** sobre o seu alvo

- As buscas heurísticas implementadas são:
  - **Busca Gulosa** ou *Best first*
  - **Busca Algoritmo A**
  - **Busca Algoritmo A***

Busca heurística

- A busca calcula a heurística de cada vértice por meio de uma fórmula ‘F’:

$$F(v) = g(n) + h(n)$$

- Função ‘g’ calcula o custo do caminho feito
- Função ‘h’ calcula a estimativa para chegar até o alvo

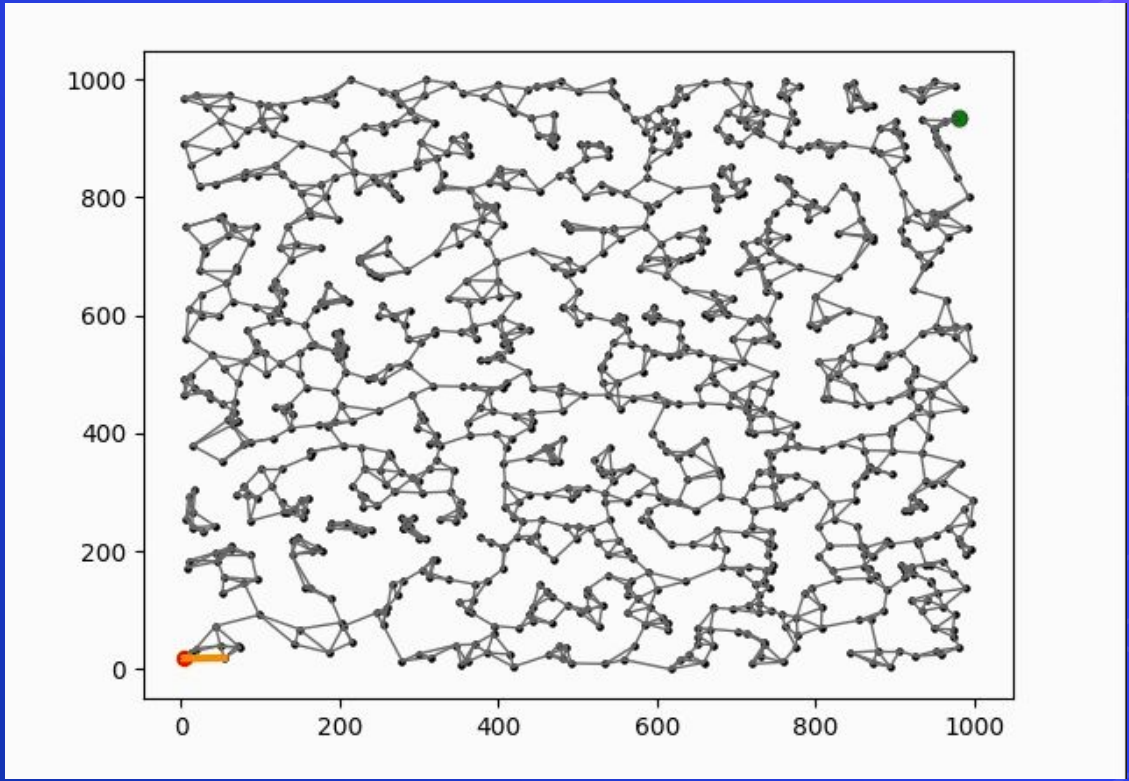
Busca Gulosa *(Best First)*

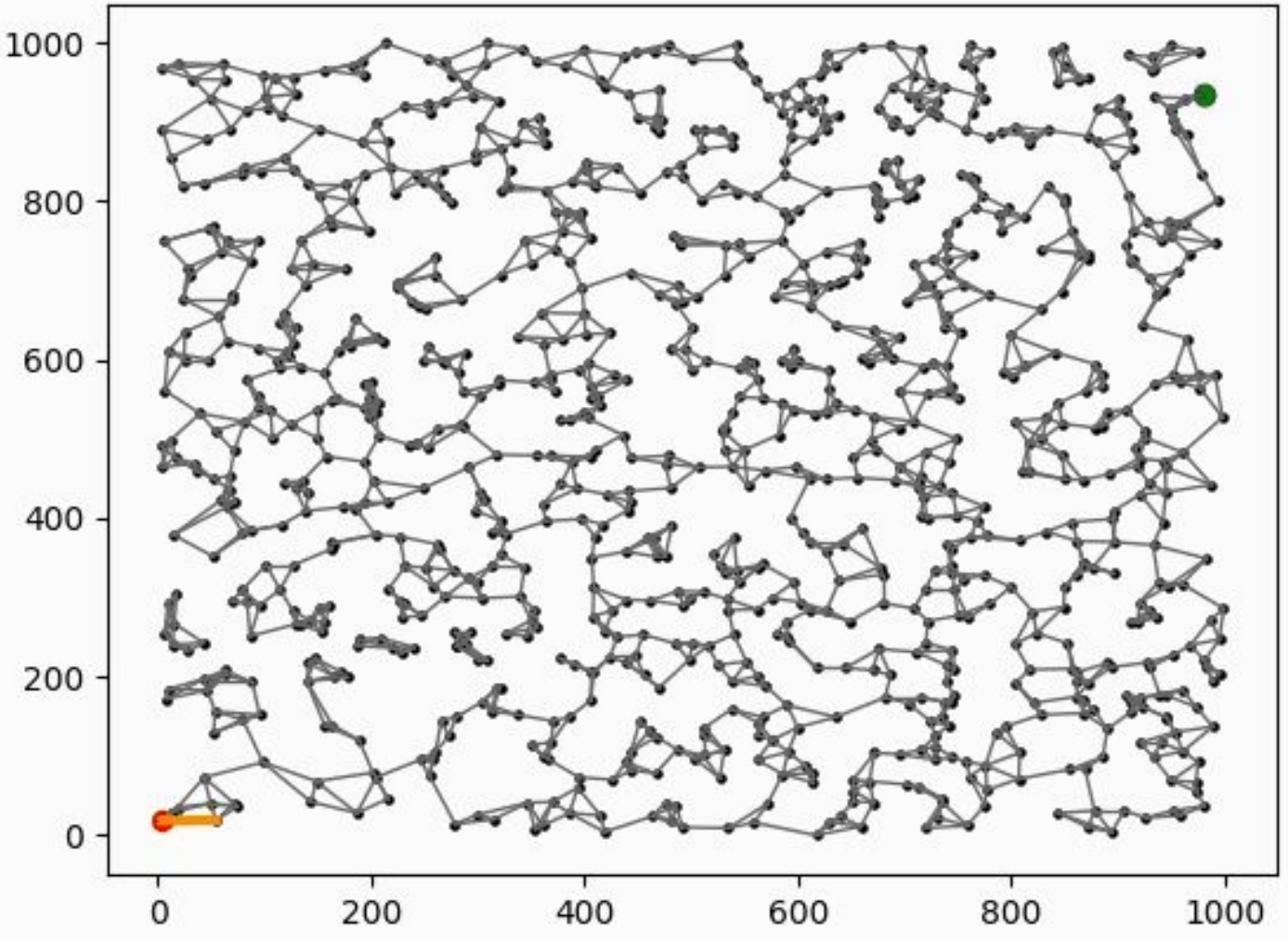
- A busca gulosa concentra-se somente nos vértices de menor distância euclidiana do alvo:

$$F(v) = G*g(n) + H*h(n)$$

- Usaremos:
  - G = 0
  - H = 1

Busca Gulosa *(Best First)*




1000
800
600
400
200
0
0
200
400
600
800
1000

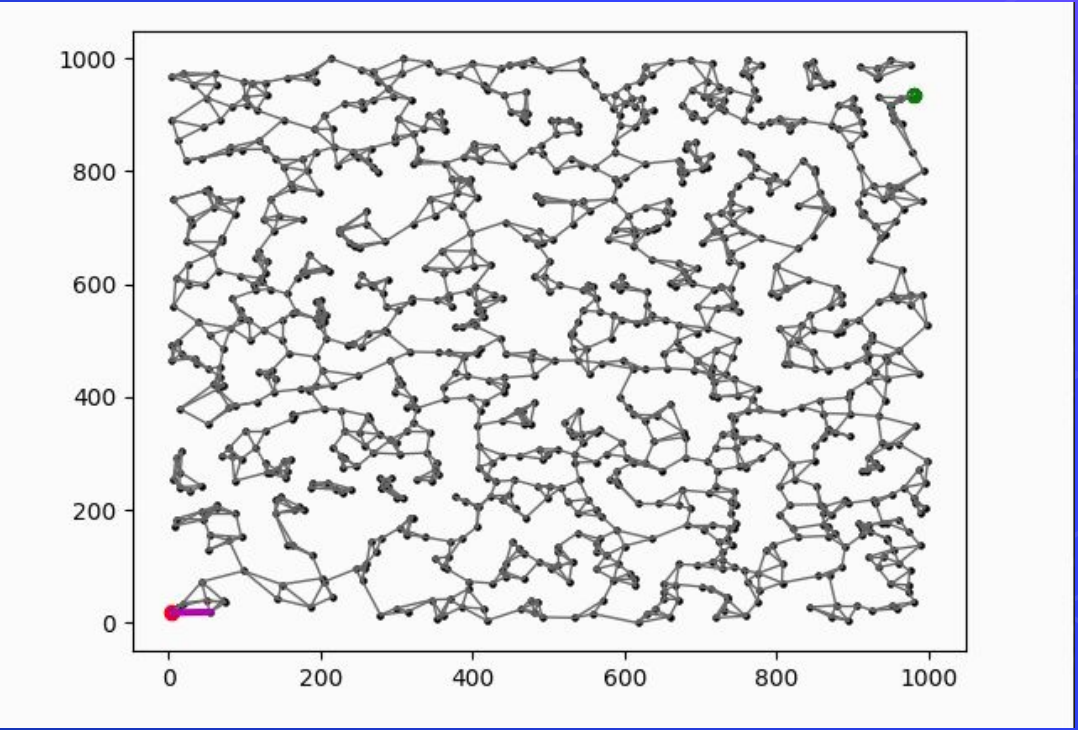
Busca Algoritmo A

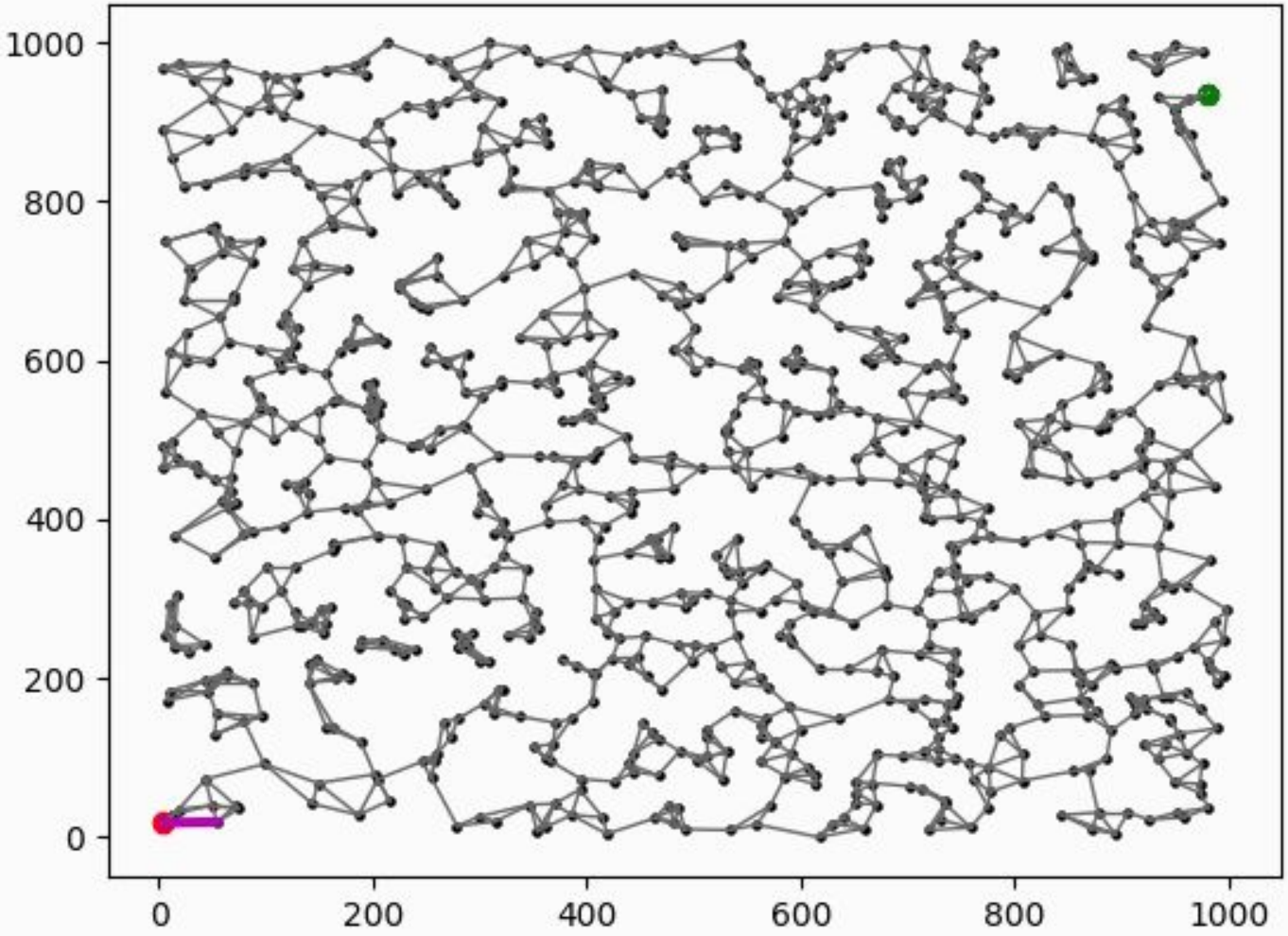
- Essa solução trata a distância até o alvo com um peso maior do que o caminho percorrido:

$$F(v) = G*g(n) + H*h(n)$$

- Usaremos:
  - G = 10
  - H = 1

Busca Algoritmo A




1000
800
600
400
200
0
0
200
400
600
800
1000

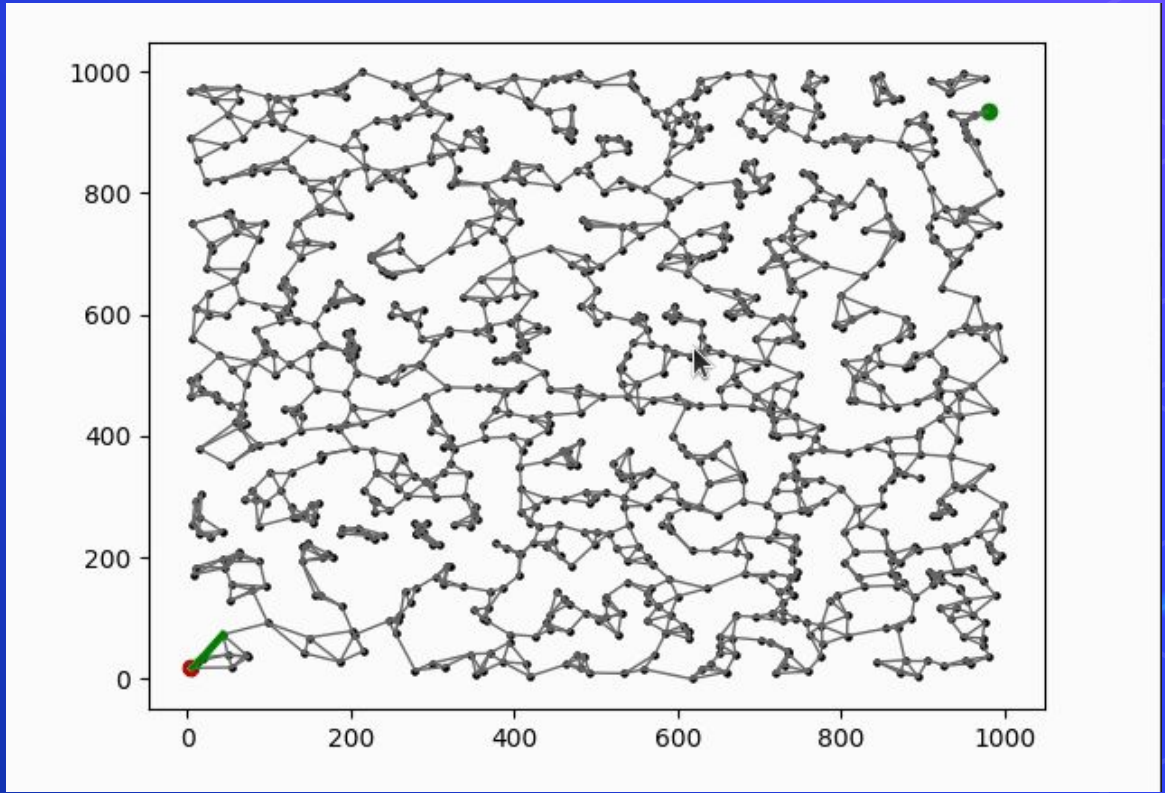
Busca Algoritmo A*

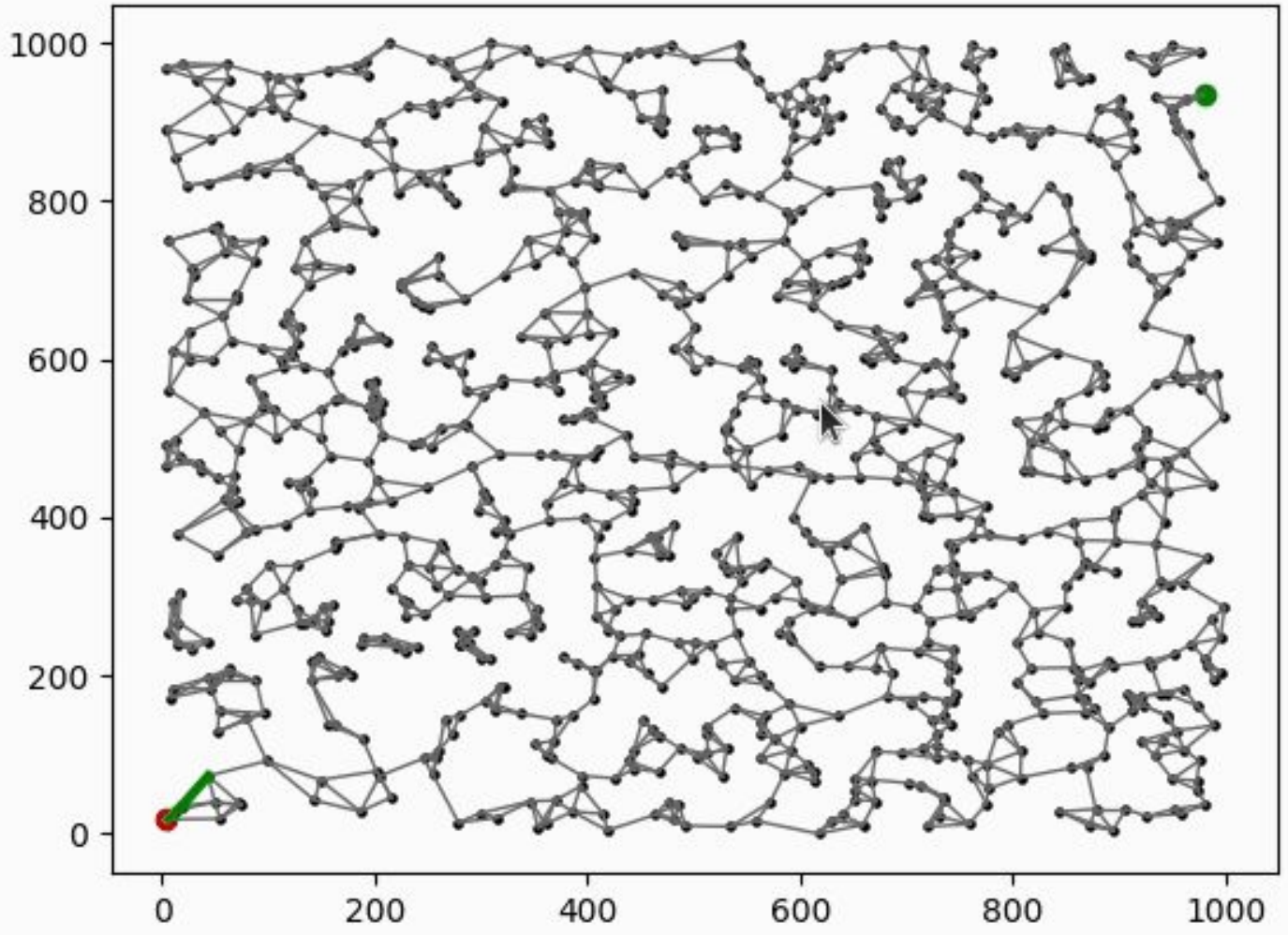
- Essa solução considera os valores heurísticos de caminho e distância até o alvo com pesos iguais:

$$F(v) = G*g(n) + H*h(n)$$

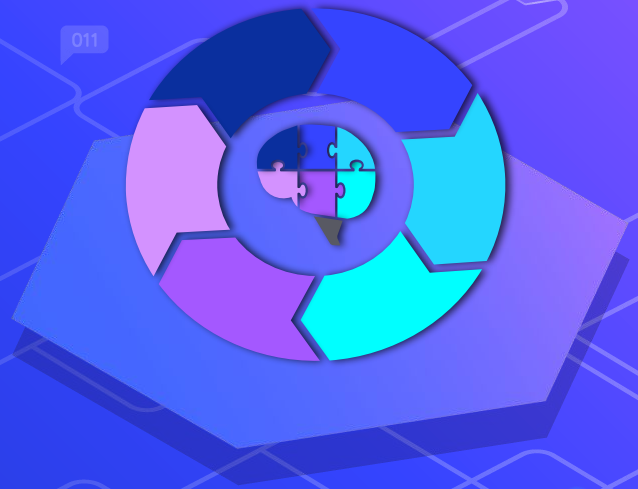
- Usaremos:
  - G = 1
  - H = 1

Busca Algoritmo A*




1000
800
600
400
200
0
0
200
400
600
800
1000

3.
Resultados e análise



Critérios

1. Tempo

Tempo de processamento até o encontro do caminho até o objetivo

2. Vértices

Número de vértices entre o ponto inicial e o objetivo

3. Distância

Distância percorrida pelo caminho até o objetivo

4. Memória

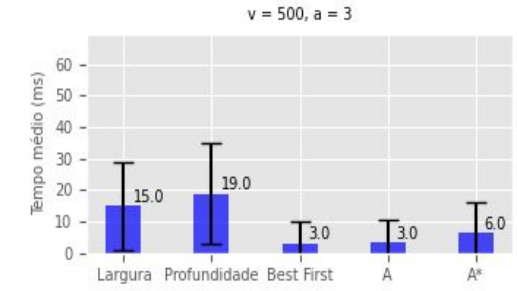
Consumo de memória para a escolha do caminho

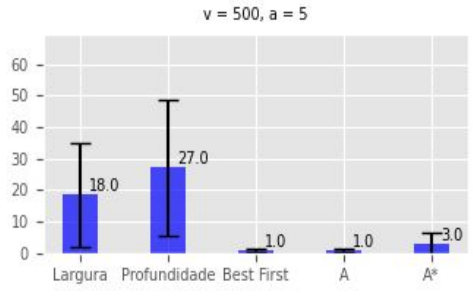
Ambiente de testes para Tempo, Vértices e Distância

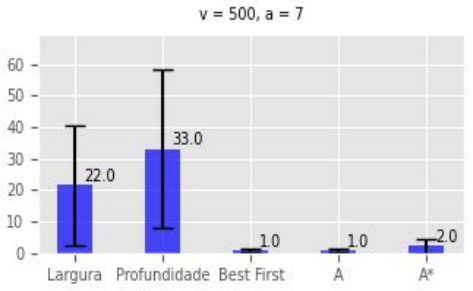
- 9 Grafos KNN foram feitos para a combinação de número de vértices: *V* = 500, 5000 e 10000, e arestas: *K* = *3, 5* e *7*
- 100 interações foram feitas de forma que para cada uma os 5 algoritmos eram acionados
- Para cada interação, o ponto inicial e o objetivo são gerados aleatoriamente.

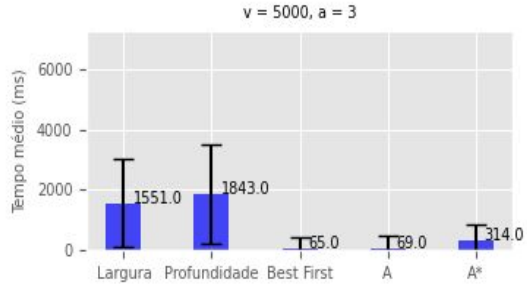
Tempo de processamento

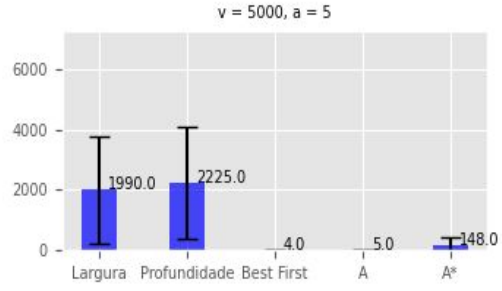
Em qual velocidade os algoritmos performam e para quais situações?

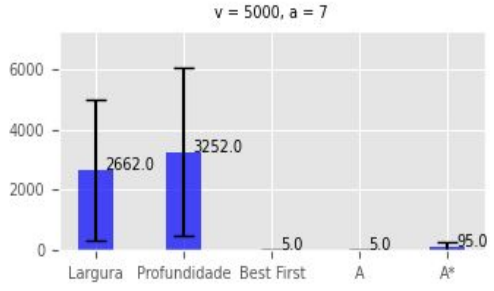

v = 500, a = 3
Tempo médio (ms)
0
10
20
30
40
50
60
15.0
19.0
3.0
3.0
6.0
Largura
Profundidade
Best First
A
A*

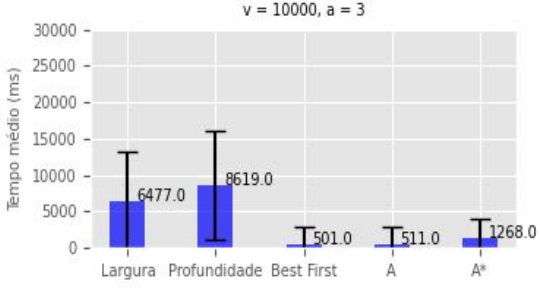

v = 500, a = 5
0
10
20
30
40
50
60
18.0
27.0
1.0
1.0
3.0
Largura
Profundidade
Best First
A
A*

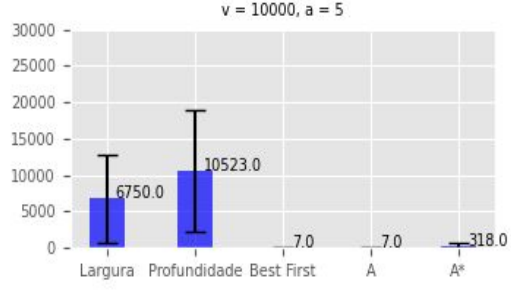

v = 500, a = 7
0
10
20
30
40
50
60
22.0
33.0
1.0
1.0
2.0
Largura
Profundidade
Best First
A
A*

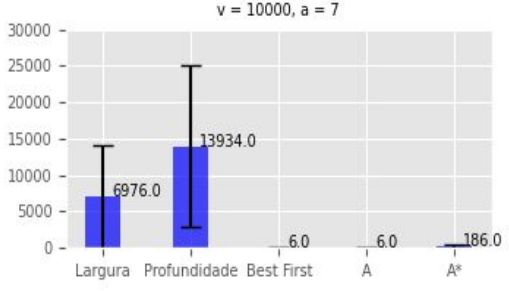

v = 5000, a = 3
Tempo médio (ms)
0
2000
4000
6000
1551.0
1843.0
65.0
69.0
314.0
Largura
Profundidade
Best First
A
A*


v = 5000, a = 5
0
2000
4000
6000
1990.0
2225.0
4.0
5.0
148.0
Largura
Profundidade
Best First
A
A*


v = 5000, a = 7
0
2000
4000
6000
2662.0
3252.0
5.0
5.0
95.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 3
Tempo médio (ms)
0
5000
10000
15000
20000
25000
30000
6477.0
8619.0
501.0
511.0
1268.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 5
0
5000
10000
15000
20000
25000
30000
6750.0
10523.0
7.0
7.0
318.0
Largura
Profundidade
Best First
A
A*

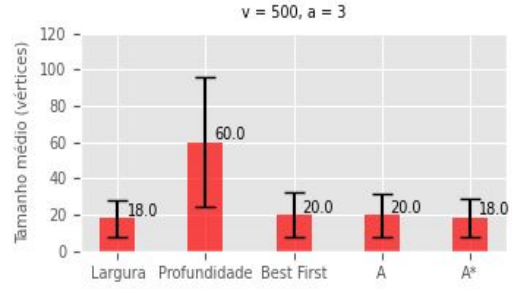

v = 10000, a = 7
0
5000
10000
15000
20000
25000
30000
6976.0
13934.0
6.0
6.0
186.0
Largura
Profundidade
Best First
A
A*

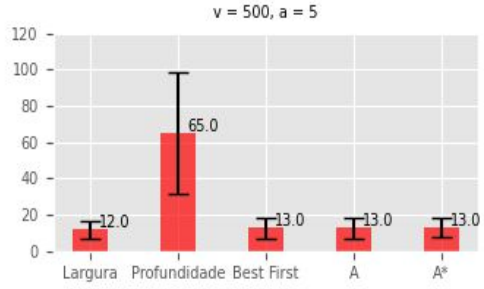
Tempo de processamento

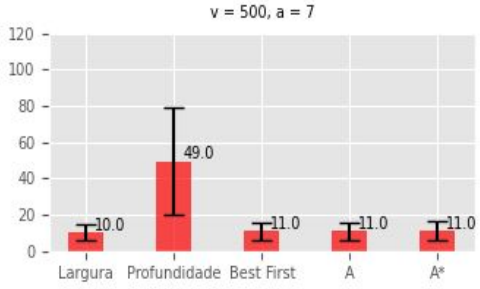
- Buscas cegas apresentam maior tempo
- Busca em profundidade apresenta maior tempo desvio padrão
- Buscas heurísticas são mais rápidas
- Best first apresenta maior velocidade

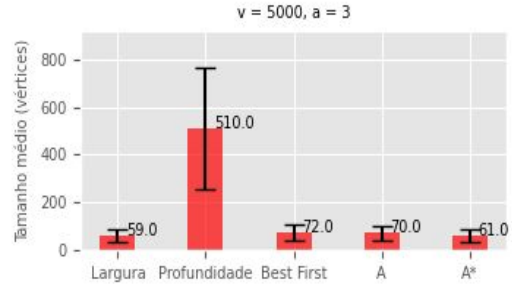
Número de vértices

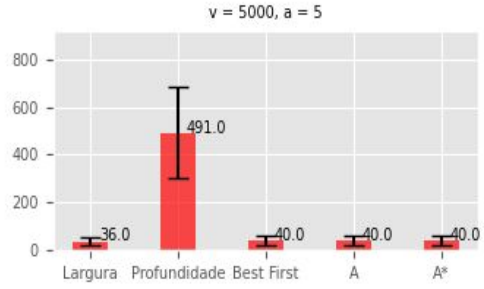
Qual é a eficiência dos algoritmos caso seja necessário alcançar o menor número de vértices?

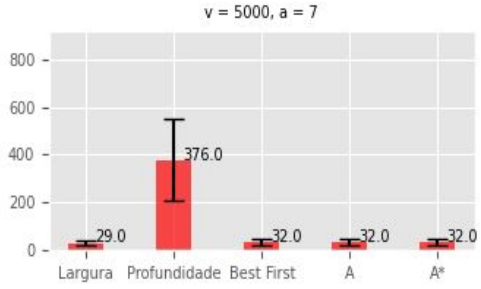

v = 500, a = 3
Tamanho médio (vértices)
120
100
80
60
40
20
0
18.0
60.0
20.0
20.0
18.0
Largura
Profundidade
Best First
A
A*

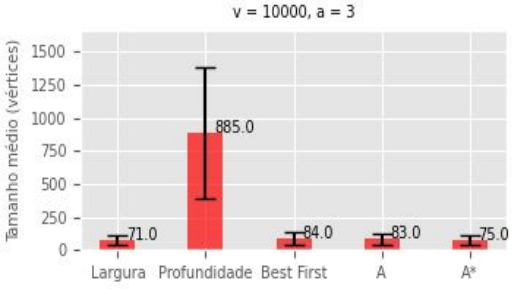

v = 500, a = 5
120
100
80
60
40
20
0
12.0
65.0
13.0
13.0
13.0
Largura
Profundidade
Best First
A
A*

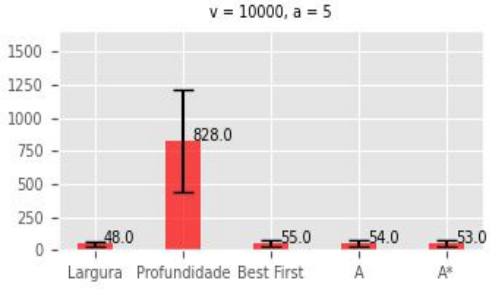

v = 500, a = 7
120
100
80
60
40
20
0
10.0
49.0
11.0
11.0
11.0
Largura
Profundidade
Best First
A
A*

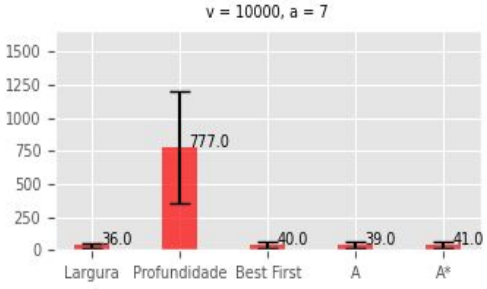

v = 5000, a = 3
Tamanho médio (vértices)
800
600
400
200
0
59.0
510.0
72.0
70.0
61.0
Largura
Profundidade
Best First
A
A*


v = 5000, a = 5
800
600
400
200
0
36.0
491.0
40.0
40.0
40.0
Largura
Profundidade
Best First
A
A*


v = 5000, a = 7
800
600
400
200
0
29.0
376.0
32.0
32.0
32.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 3
Tamanho médio (vértices)
1500
1250
1000
750
500
250
0
71.0
885.0
84.0
83.0
75.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 5
1500
1250
1000
750
500
250
0
48.0
828.0
55.0
54.0
53.0
Largura
Profundidade
Best First
A
A*

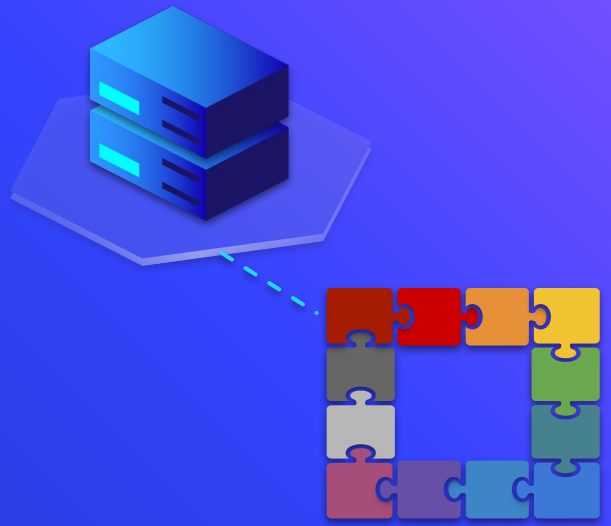

v = 10000, a = 7
1500
1250
1000
750
500
250
0
36.0
777.0
40.0
39.0
41.0
Largura
Profundidade
Best First
A
A*

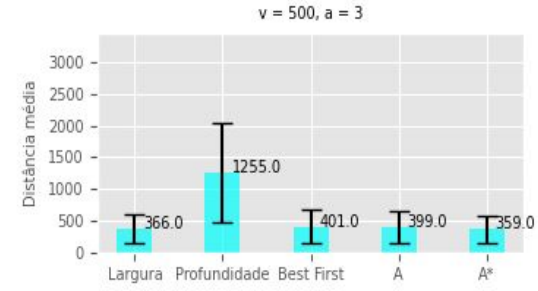
Número de vértices

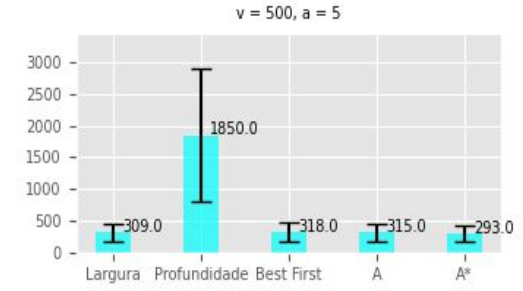
- Busca por Profundidade apresenta o maior caminho
- Busca em Largura apresenta o menor caminho
- Buscas heurísticas apresentam similaridade nesse quesito comparáveis com a Largura
- Dentre os heurísticos, A* apresenta leve vitória

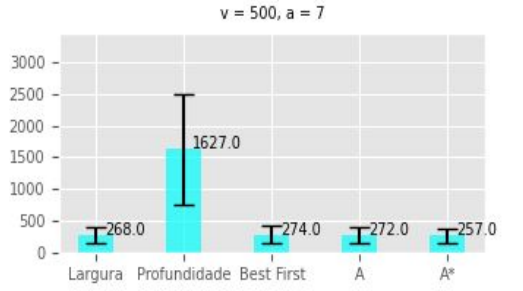
Distância do caminho

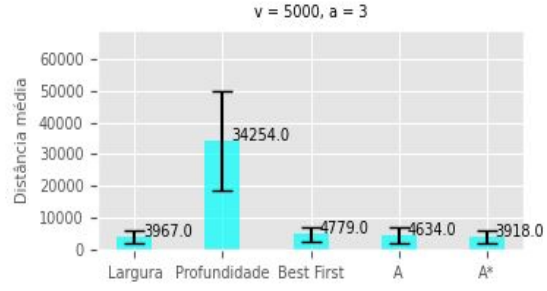
Qual é a eficiência dos algoritmos caso seja necessário alcançar a menor distância?

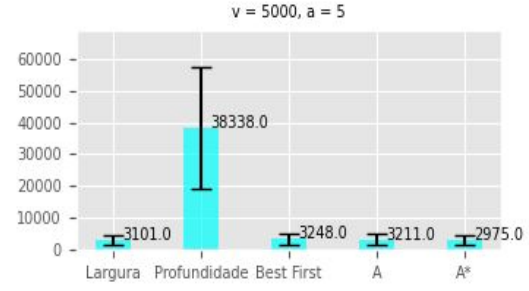


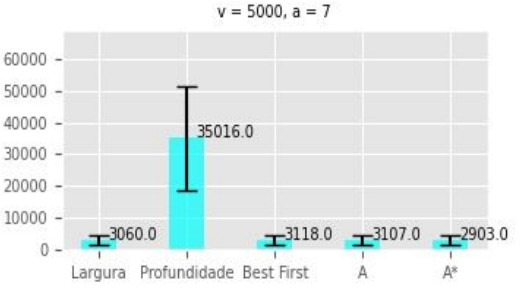

v = 500, a = 3
Distância média
3000
2500
2000
1500
1000
500
0
366.0
1255.0
401.0
399.0
359.0
Largura
Profundidade
Best First
A
A*

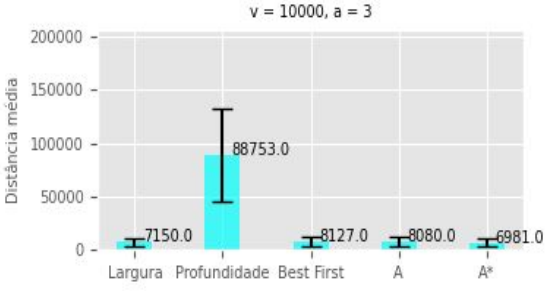

v = 500, a = 5
3000
2500
2000
1500
1000
500
0
309.0
1850.0
318.0
315.0
293.0
Largura
Profundidade
Best First
A
A*

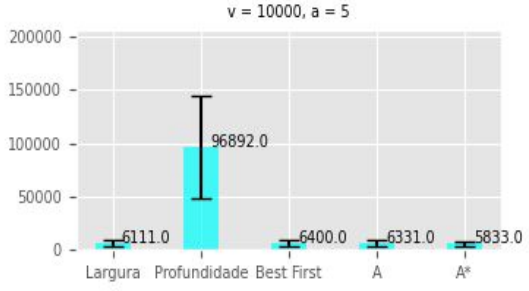

v = 500, a = 7
3000
2500
2000
1500
1000
500
0
268.0
1627.0
274.0
272.0
257.0
Largura
Profundidade
Best First
A
A*

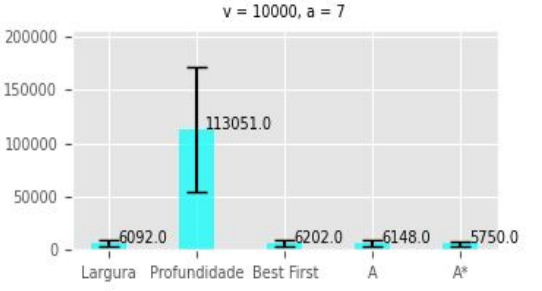

v = 5000, a = 3
Distância média
60000
50000
40000
30000
20000
10000
0
3967.0
34254.0
4779.0
4634.0
3918.0
Largura
Profundidade
Best First
A
A*


v = 5000, a = 5
60000
50000
40000
30000
20000
10000
0
3101.0
38338.0
3248.0
3211.0
2975.0
Largura
Profundidade
Best First
A
A*


v = 5000, a = 7
60000
50000
40000
30000
20000
10000
0
3060.0
35016.0
3118.0
3107.0
2903.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 3
Distância média
200000
150000
100000
50000
0
7150.0
88753.0
8127.0
8080.0
6981.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 5
200000
150000
100000
50000
0
6111.0
96892.0
6400.0
6331.0
5833.0
Largura
Profundidade
Best First
A
A*


v = 10000, a = 7
200000
150000
100000
50000
0
6092.0
113051.0
6202.0
6148.0
5750.0
Largura
Profundidade
Best First
A
A*

Distância do caminho

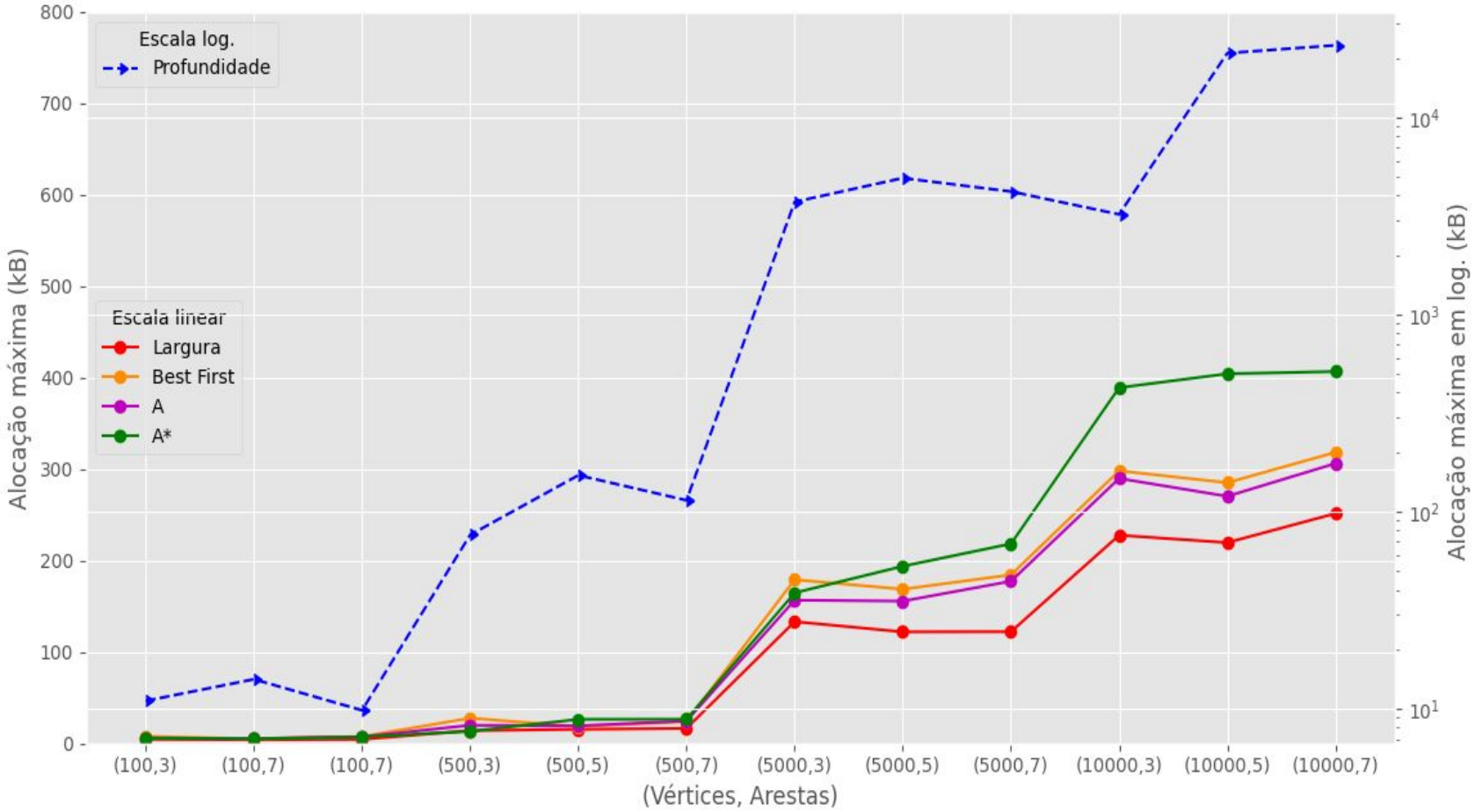
- Busca em Profundidade alcança caminho maior
- Buscas heurísticas apresentam tamanho de caminho comparáveis
- A* alcança o menor caminho devido a análise pessimista

Consumo máximo de memória

Qual é o consumo de memória dos métodos a partir desta implementação?

Ambiente de testes para Memória

- Para cada um dos 9 grafos gerados, os pontos críticos (mais afastados) foram mapeados.
- O algoritmo age no limite de cada grafo, buscando os dois pontos que anotamos como mais afastados
- O objetivo é alcançar um majorante para o consumo de memória em cada situação.
- A biblioteca utilizada foi o *tracemalloc* para o monitoramento. Foi medido o **pico de memória** em cada situação para cada método.

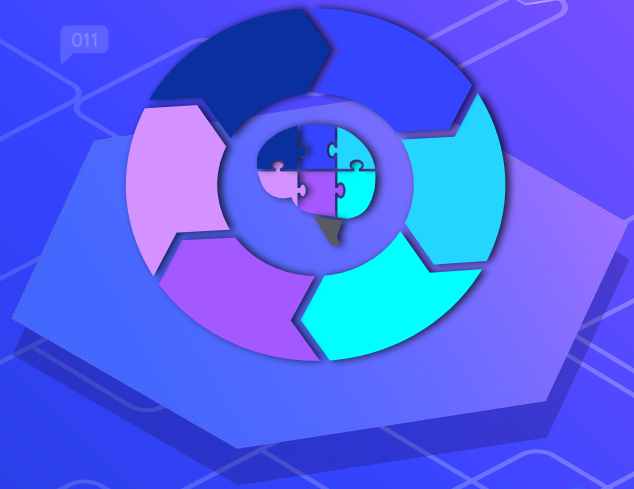

Escala log.
Profundidade
Escala linear
Largura
Best First
A
A*
Alocação máxima (kB)
Alocação máxima em log. (kB)
(Vértices, Arestas)
800
700
600
500
400
300
200
100
0
10^4
10^3
10^2
10^1
(100,3)
(100,7)
(100,7)
(500,3)
(500,5)
(500,7)
(5000,3)
(5000,5)
(5000,7)
(10000,3)
(10000,5)
(10000,7)

Consumo máximo de memória

- Buscas heurísticas e Largura apresentam consumo máximo de memória similar

- Busca em Profundidade apresenta consumo de memória muito maior
  - Fator principal: tamanho de caminho em vértices ~20x maior em grafos de 10000 vértices

3.
Conclusão



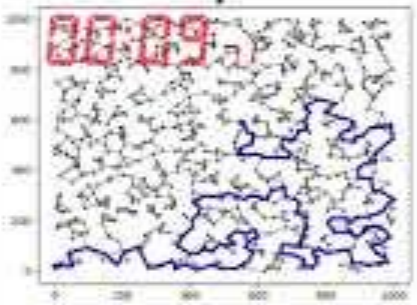
Best & Worst

	Critérios			
	Tempo	**Vértices**	**Distância**	**Memória**
Melhor	Best First	Largura	A*	Largura
Pior	Profundidade	Profundidade	Profundidade	Profundidade

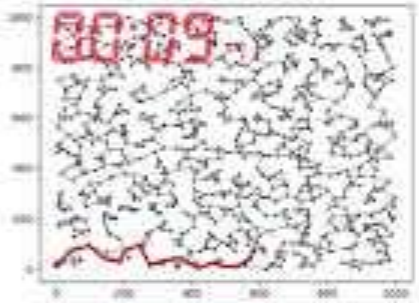
Cards

	Largura	Profundidade	Best First	A	A*
Tempo	Ruim	Pior	Melhor	Bom	Bom
Vértices	Melhor	Pior	Bom	Bom	Bom
Distância	Bom	Pior	Bom	Bom	Melhor
Memória	Melhor	Pior	Bom	Bom	Bom

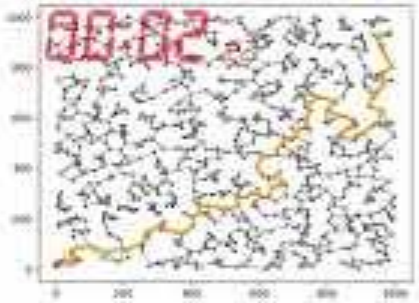
Depth



Breadth

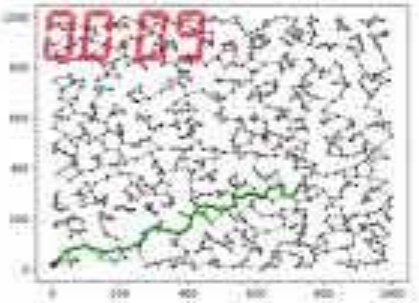


BestFirst

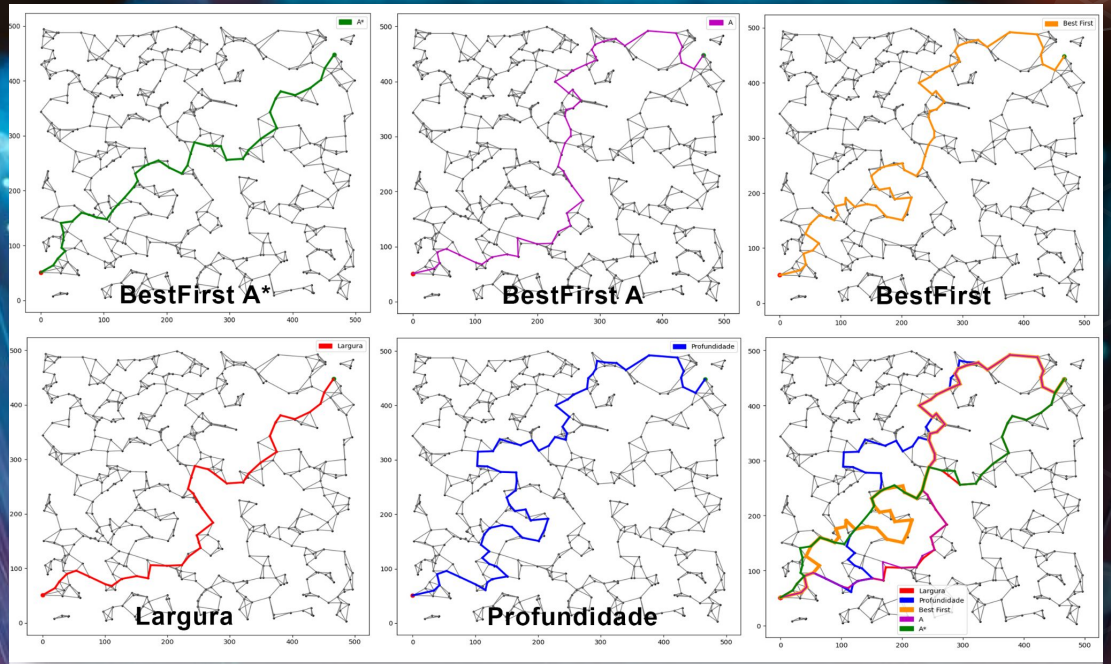


A Algorithm

A* Algorithm



Obrigado!


BestFirst A*
BestFirst A
BestFirst
Largura
Profundidade
Largura
Profundidade
Best First
A
A*