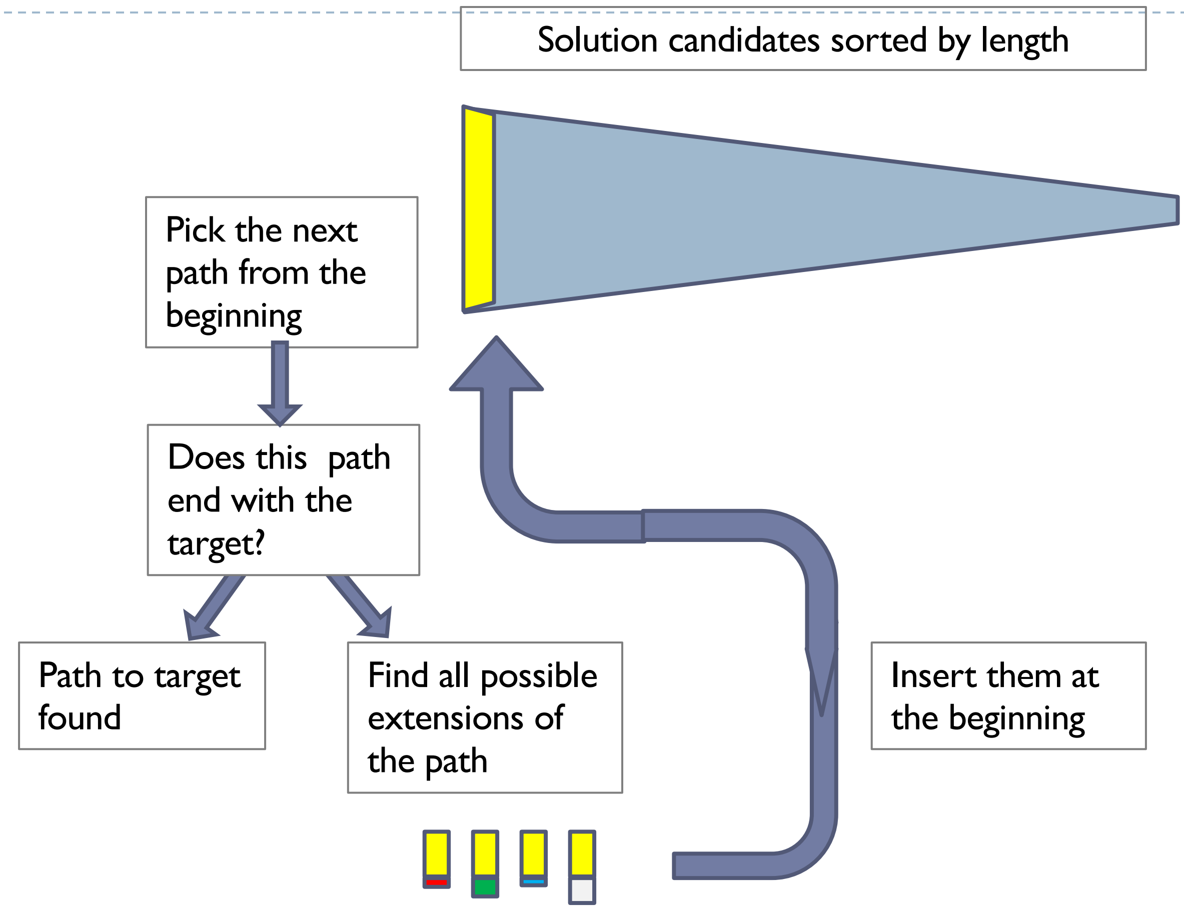
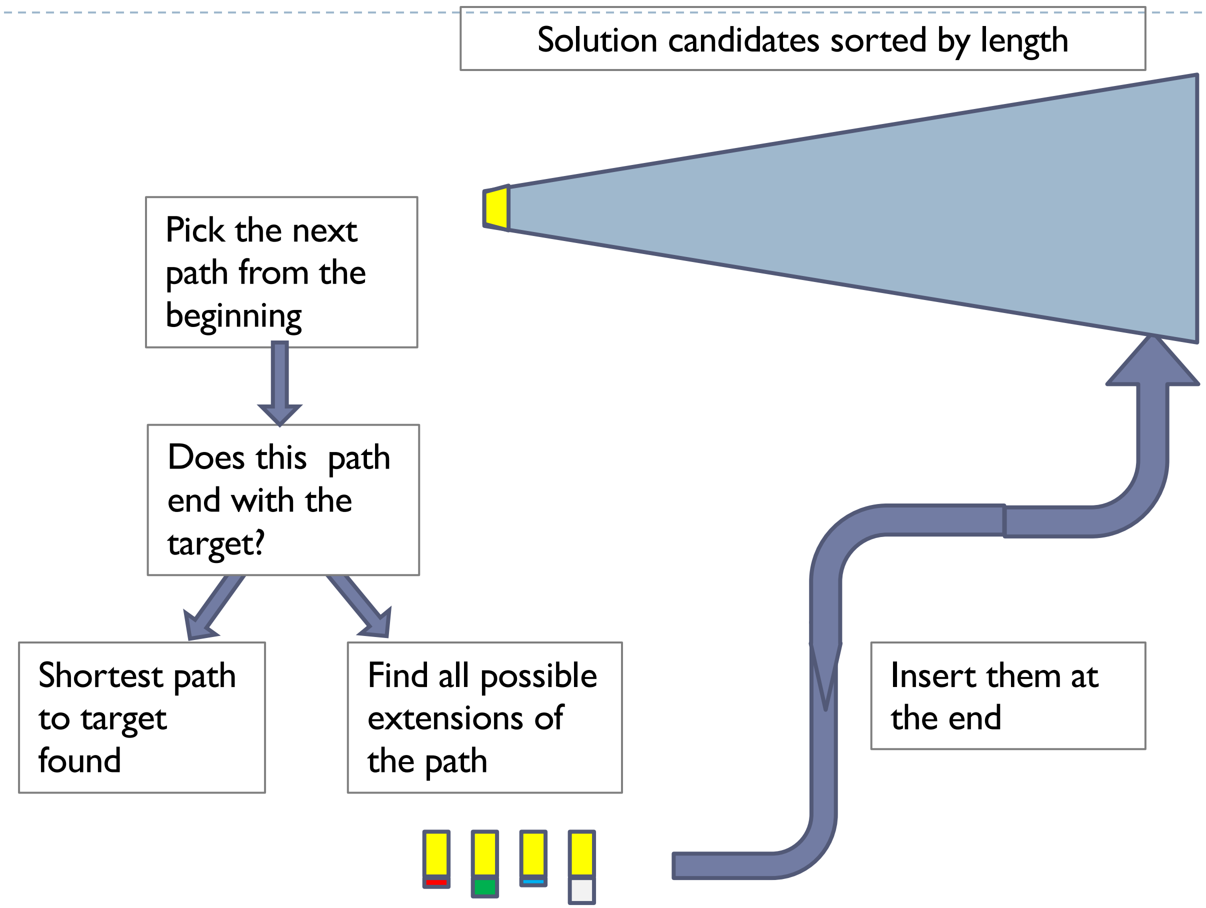
Modelling, Simulation & Optimisation (H9MSO)

3. Greedy and Heuristic Algorithms

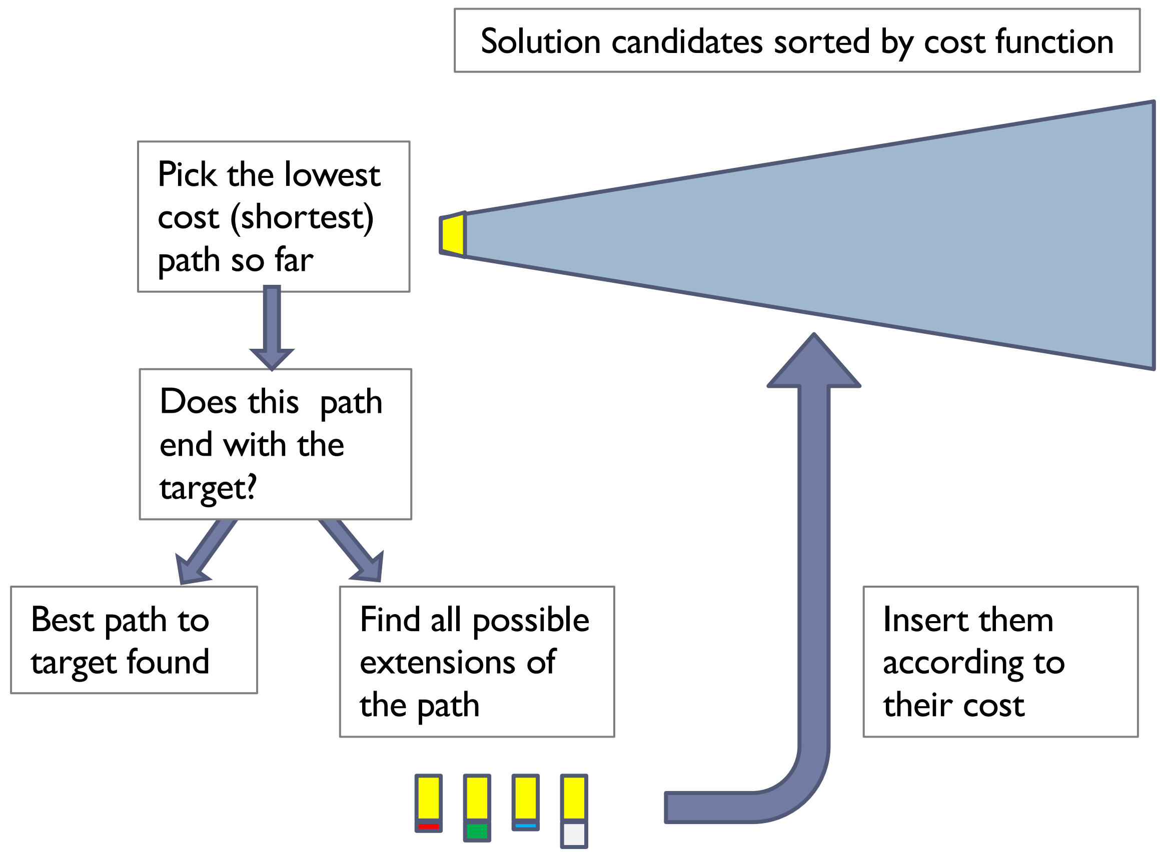
Using the
Depth First Algorithm



Using the Breadth First Algorithm



Using the Best First Algorithm to find the shortest path



Application

The Algorithm couldn't work in New York

NYSE –
Guggenheim Museum

12km

Why?

Application

The Algorithm couldn't work in New York

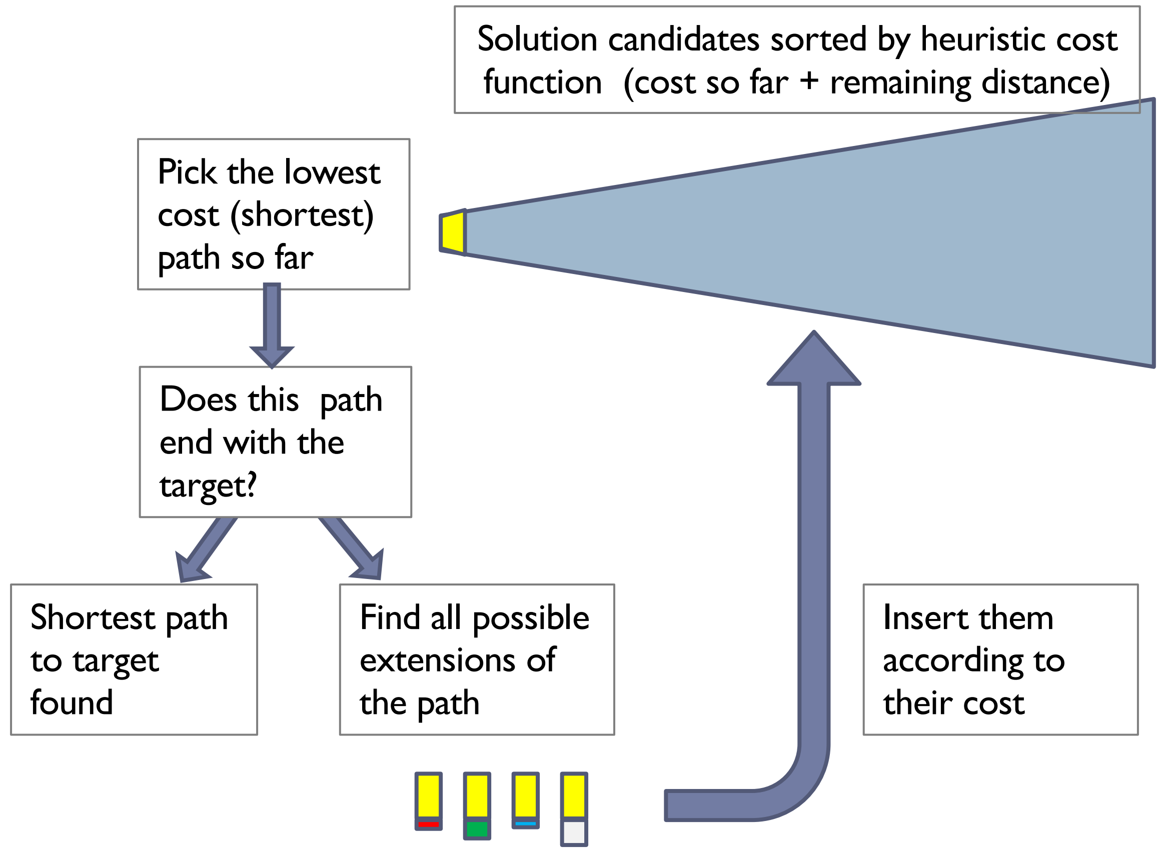
NYSE –
Guggenheim Museum

12km

Why?

- Many steps (116)
- Higher Branching Grade (3)
- $\approx 10^{55}$ candidates

Using the A* Algorithm to find the shortest path



The Idea…

We use a two-component cost function:

c(x) = distance travelled so far f(x) +

a heuristic function h(x), which is **lower bound**

for distance still to travel

c(x) = f(x) + h(x)

heuristic function h(x)

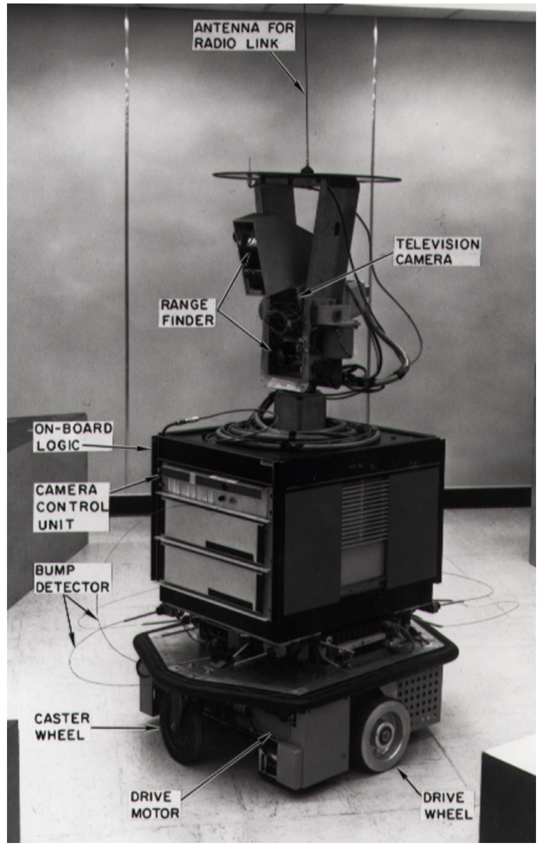
We choose

$$h(x)=d(x, \text{target})$$

The (Euclidean) distance still to travel is for sure shorter than the best path we can find

The invention of the Algorithm

Shakey, one of the first mobile robots (1968)



Source: Nils Nilsson, The Quest for AI. Cambridge University Press, 2009. https://ai.stanford.edu/~nilsson/QAI/qai.pdf

```python
def shortestPathDepthFirst(M, A, B):

    # candidates C are the paths so far,
    def insert(C, p):
        return [p]+C

    V, E = M
    assert(A in V and B in V)
    C = [[A]]
    count = 0

    while len(C)>0:
        # take the first candidate out of the list of candidates
        path = C[0]
        C = C[1:]
        count += 1
        if path[-1]==B:
            print(count, "steps")
            return path
        else:
            for (x, y) in E:
                if path[-1]==x and y not in path:
                    C = insert(C, path+[y])
                elif path[-1]==y and x not in path:
                    C = insert(C, path+[x])
    return None
```

Insert new, longer path p at the beginning

```python
def shortestPathBreadthFirst(M, A, B):

    # candidates C are the paths so far,
    def insert(C, p):
        return C+[p]

    V, E = M
    assert(A in V and B in V)
    C = [[A]]
    count = 0

    while len(C)>0:
        # take the first candidate out of the list of candidates
        path = C[0]
        C = C[1:]
        count += 1
        if path[-1]==B:
            print(count, "steps")
            return path
        else:
            for (x, y) in E:
                if path[-1]==x and y not in path:
                    C = insert(C, path+[y])
                elif path[-1]==y and x not in path:
                    C = insert(C, path+[x])
    return None
```

Insert new, longer path p at the end

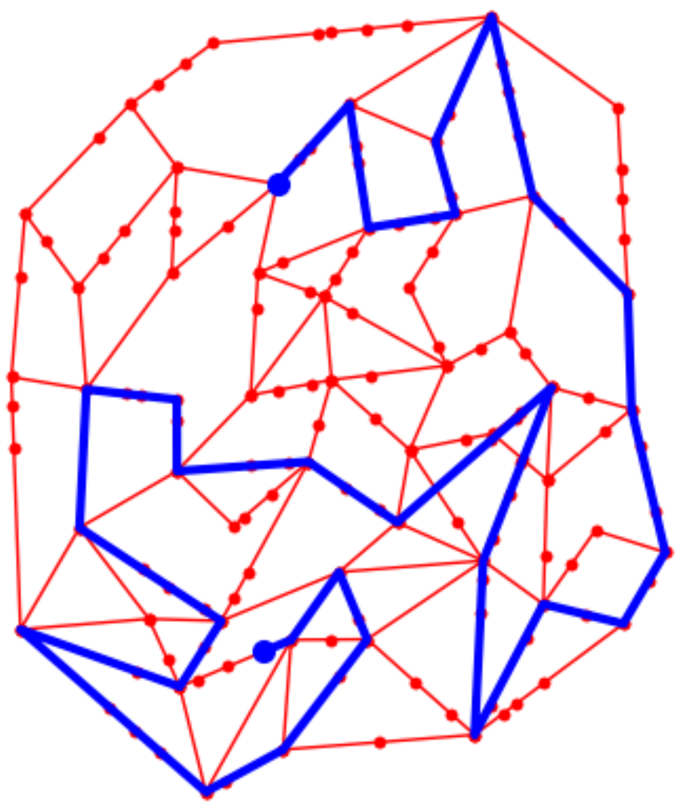
```python
def shortestPathBestFirst(M, A, B):

    # candidates C are the paths so far,
    def insert(C, p):
        pl = pathLength(p)
        for i in range(0, len(C)):
            if pathLength(C[i]) > pl:
                return C[:i]+[p]+C[i:]
        return C+[p]

    V, E = M
    assert(A in V and B in V)
    C = [[A]]
    count = 0

    while len(C)>0:
        # take the first candidate out of the list of candidates
        path = C[0]
        count += 1
        C = C[1:]
        if path[-1]==B:
            print(count, "steps")
            return path
        else:
            for (x, y) in E:
                if path[-1]==x and y not in path:
                    C = insert(C, path+[y])
                elif path[-1]==y and x not in path:
                    C = insert(C, path+[x])
    return None
```

Insert new, longer path p
where it fits in the middle

```python
def shortestPath(M, A, B):
    def h(p):
        return pathLength(p) + dist(p[-1],B)

    # candidates C are the paths so far,
    # sorted by the heuristic function
    def insert(C, p):
        hp = h(p)
        for i in range(len(C)):
            if h(C[i])>hp:
                return C[:i]+[p]+C[i:]
        return C+[p]

    V, E = M
    assert(A in V and B in V)
    C = [[A]]
    count = 0

    while len(C)>0:
        # take the first candidate out of the list of candidates
        path = C[0]
        C = C[1:]
        count += 1
        if path[-1]==B:
            print(count, "steps")
            return path
        else:
            for (x, y) in E:
                if path[-1]==x and y not in path:
                    C = insert(C, path+[y])
                elif path[-1]==y and x not in path:
                    C = insert(C, path+[x])
    return None
```

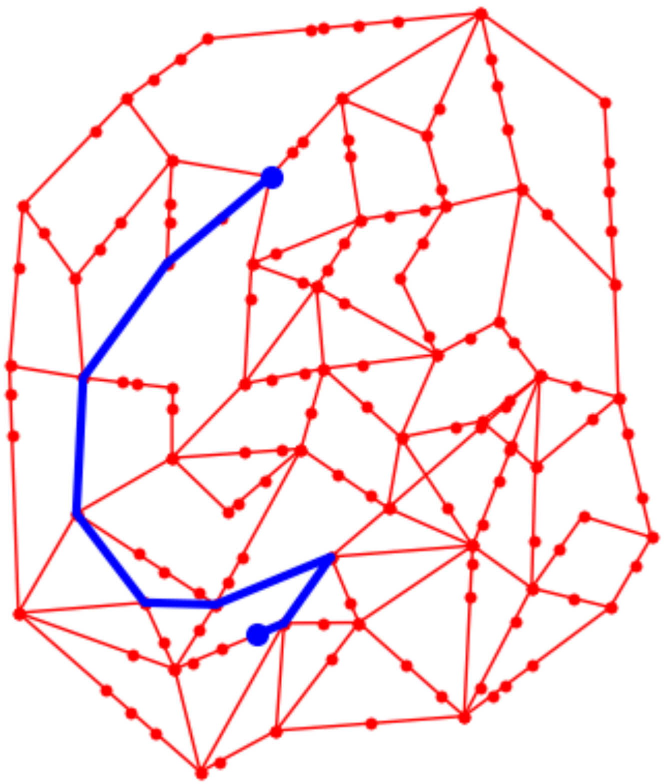
Insert new, longer path p where it fits in the middle

Shortest Path using Depth First Algorithm



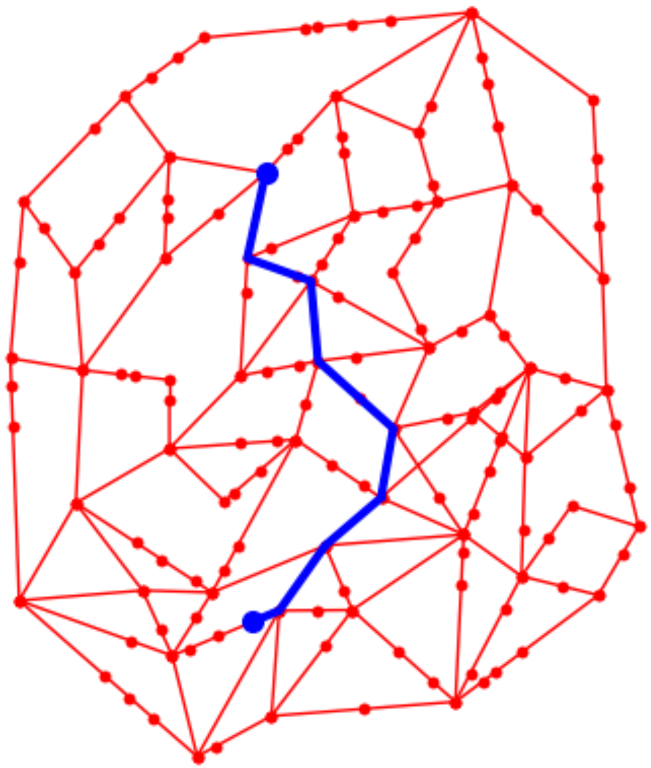
Number of iterations: 124
Number of steps for solution: 73
Length of solution:: 36,889

Shortest Path using Breadth First Algorithm



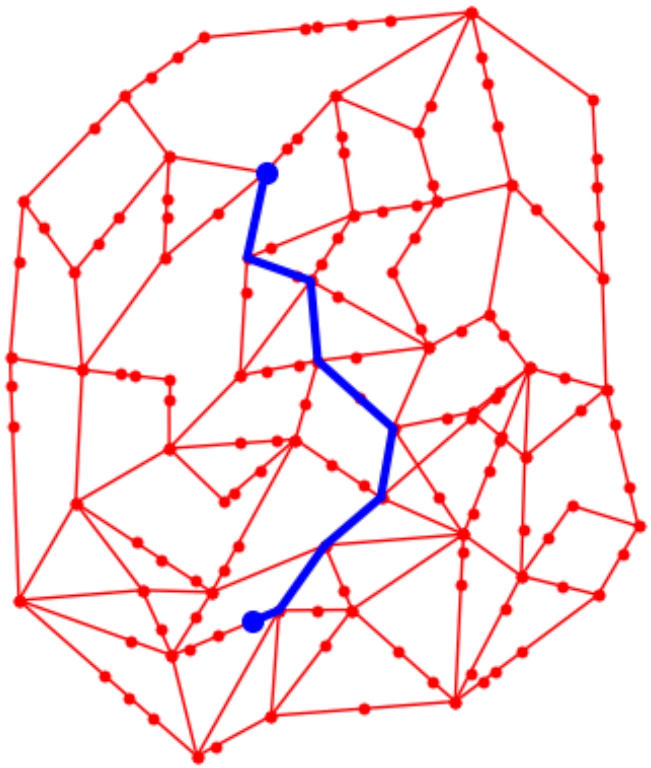
Number of iterations: 877
Number of steps for solution: 10
Length of solution:: 8,667

Shortest Path using Best First Algorithm



Number of iterations: 1,231
Number of steps for solution: 11
Length of solution:: 6,264

Shortest Path using A* Algorithm



Number of iterations: 65
Number of steps for solution: 11
Length of solution:: 6,264