Bio-Inspired Algorithms

Local Search

Daniel Vaz

daniel.ramosvaz@esiee.fr

2026-02-12

Last Lecture

Bio-inspired algorithms:

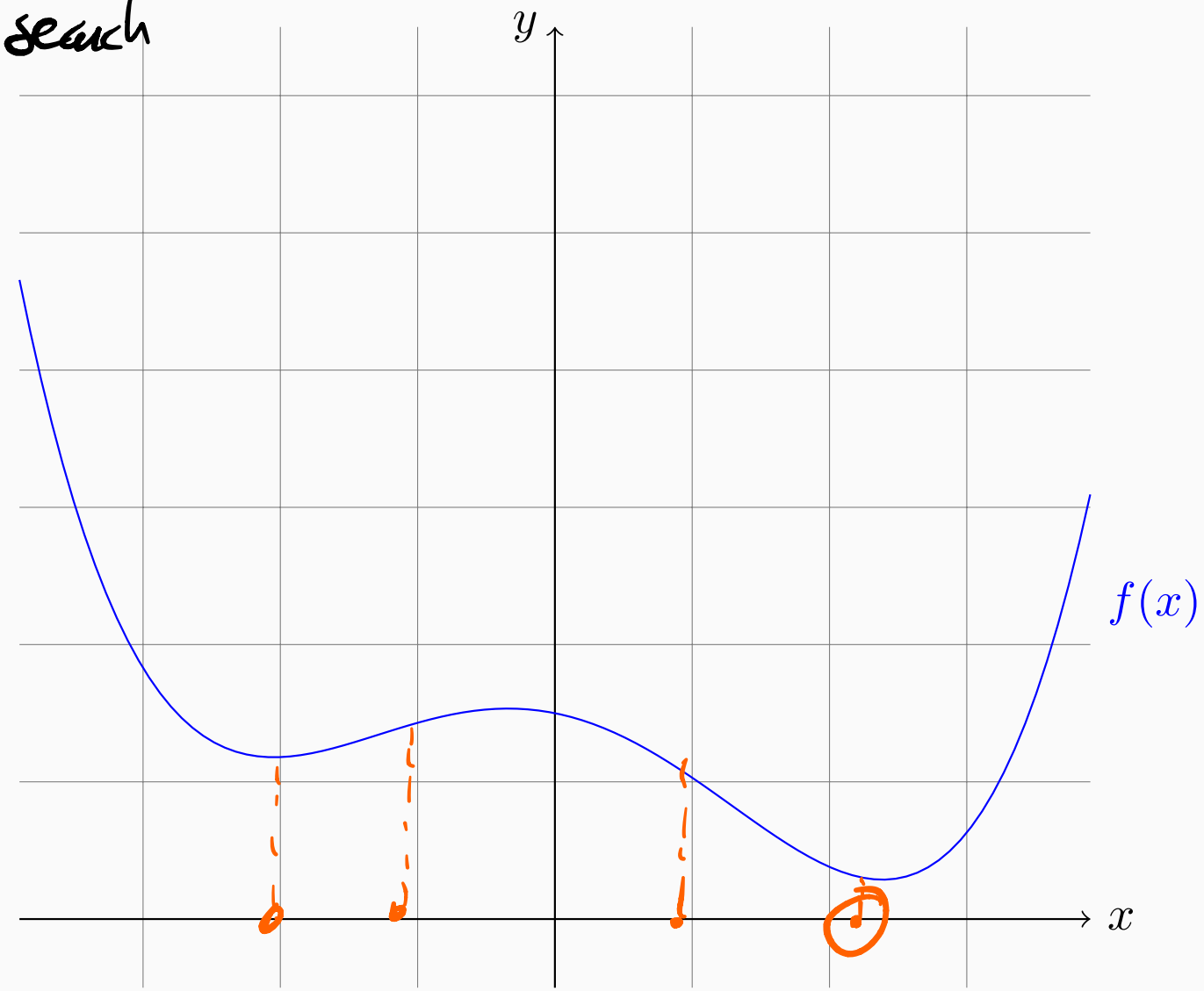
- Inspired in nature
- Mostly for optimization problems
  - Find the solution with the best fitness
- NP-hard problems: as hard as any in NP
- We will use heuristic approaches
  - which use practical knowledge

Some simple heuristics:

- Random search: tests random sampled solutions
- Hill climbing: moves to better neighbors
- A*: uses heuristic to center shortest path search

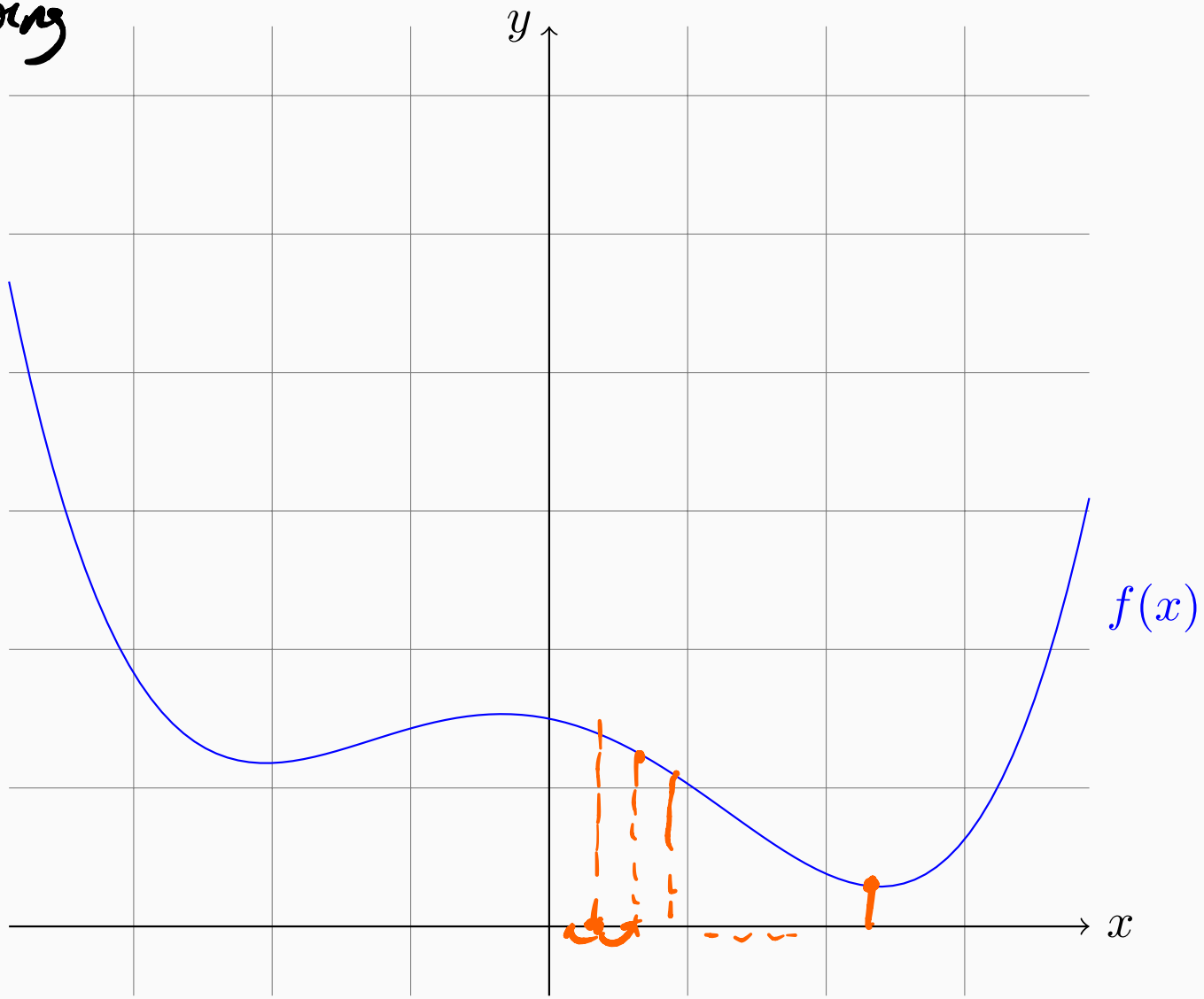
Example

Random search

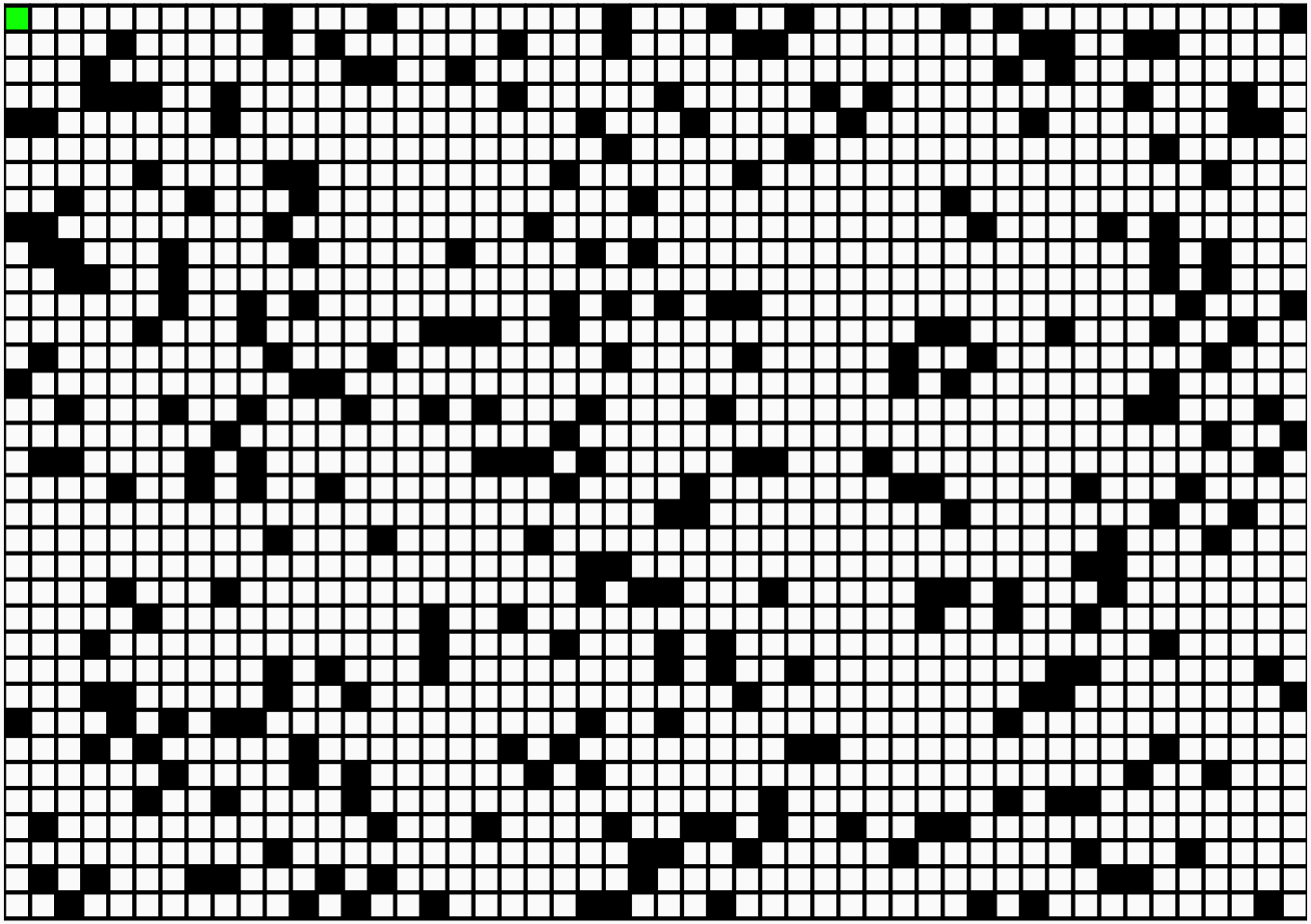


Example

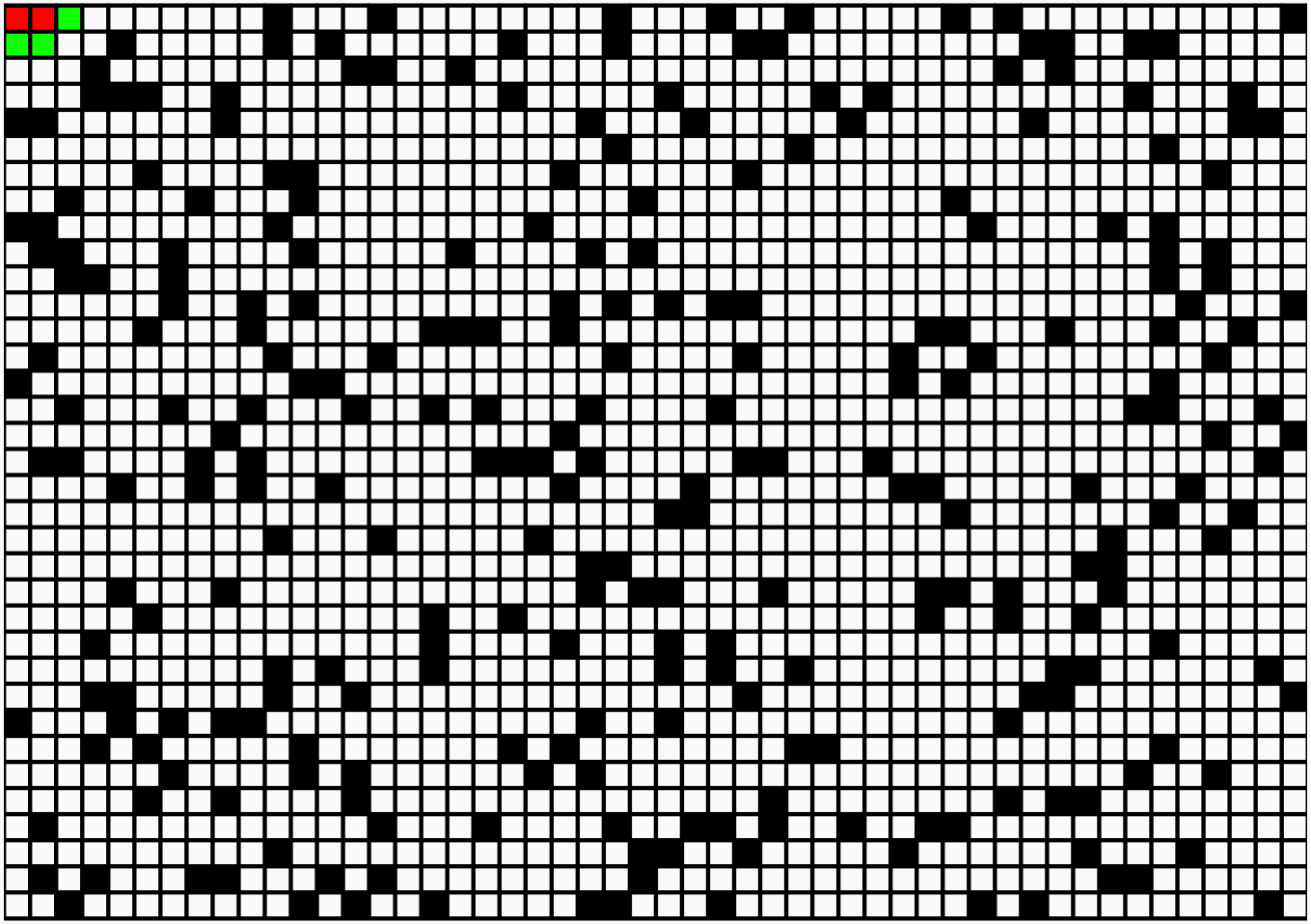
Hill Climbing



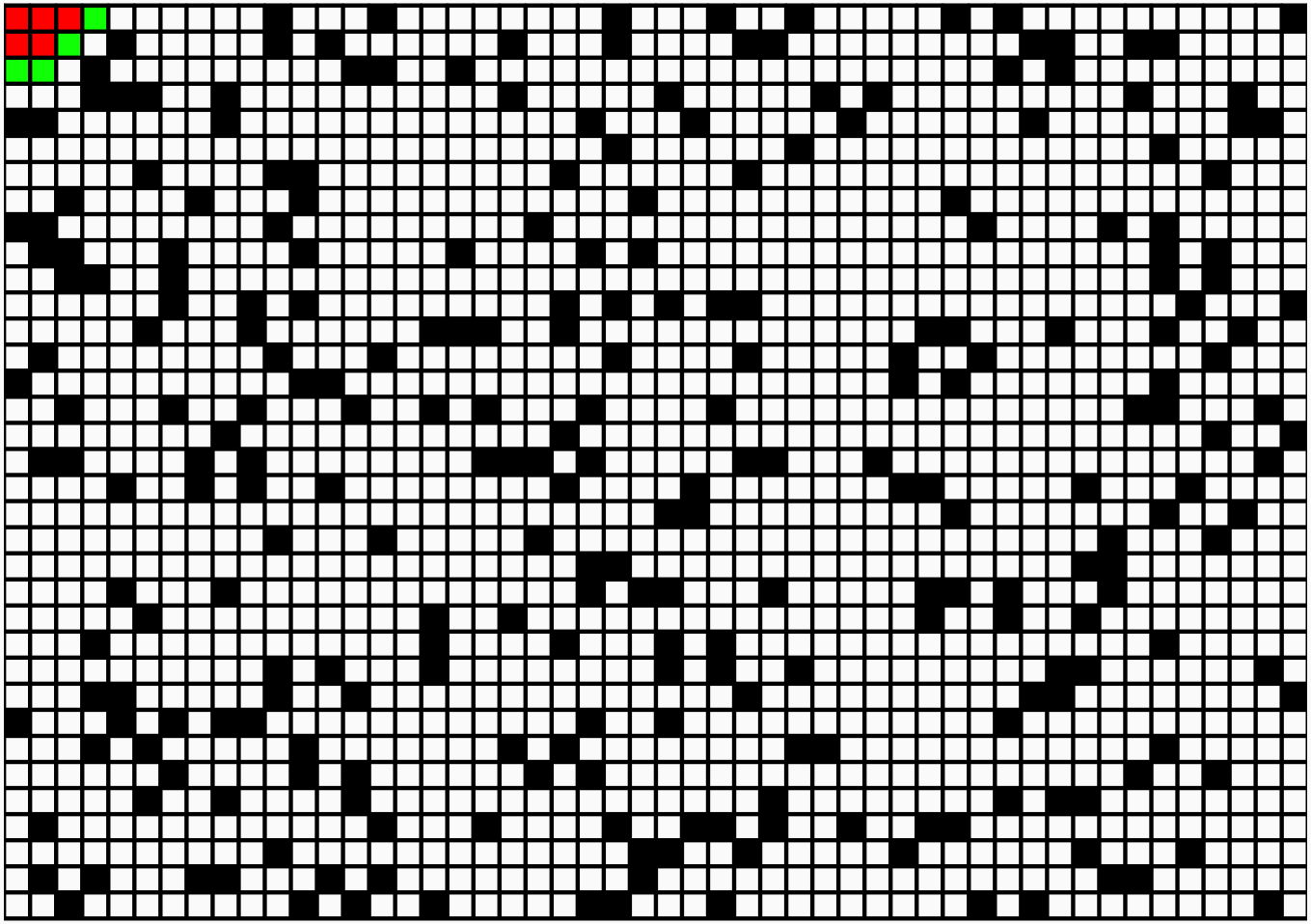
A* Algorithm



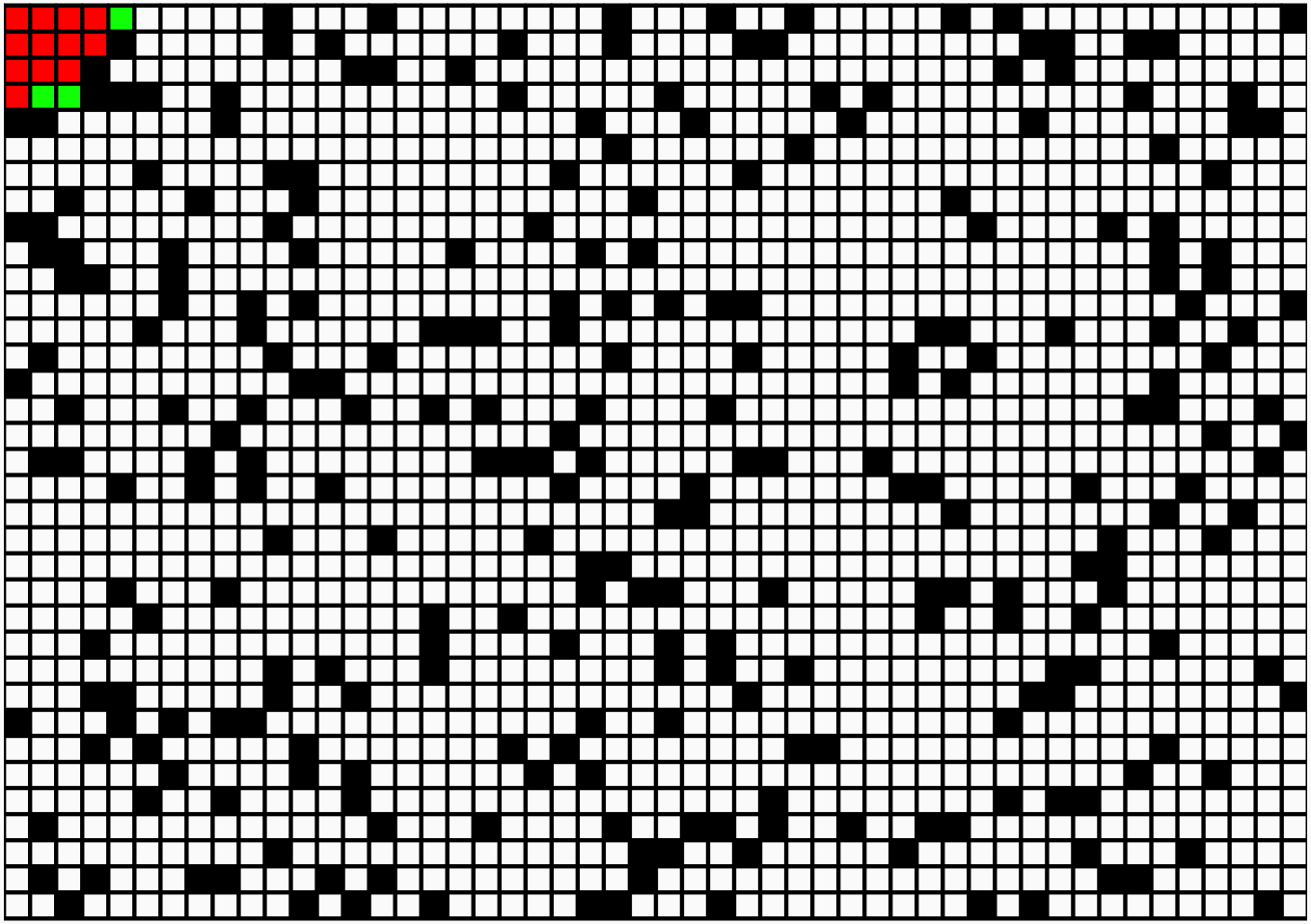
A* Algorithm



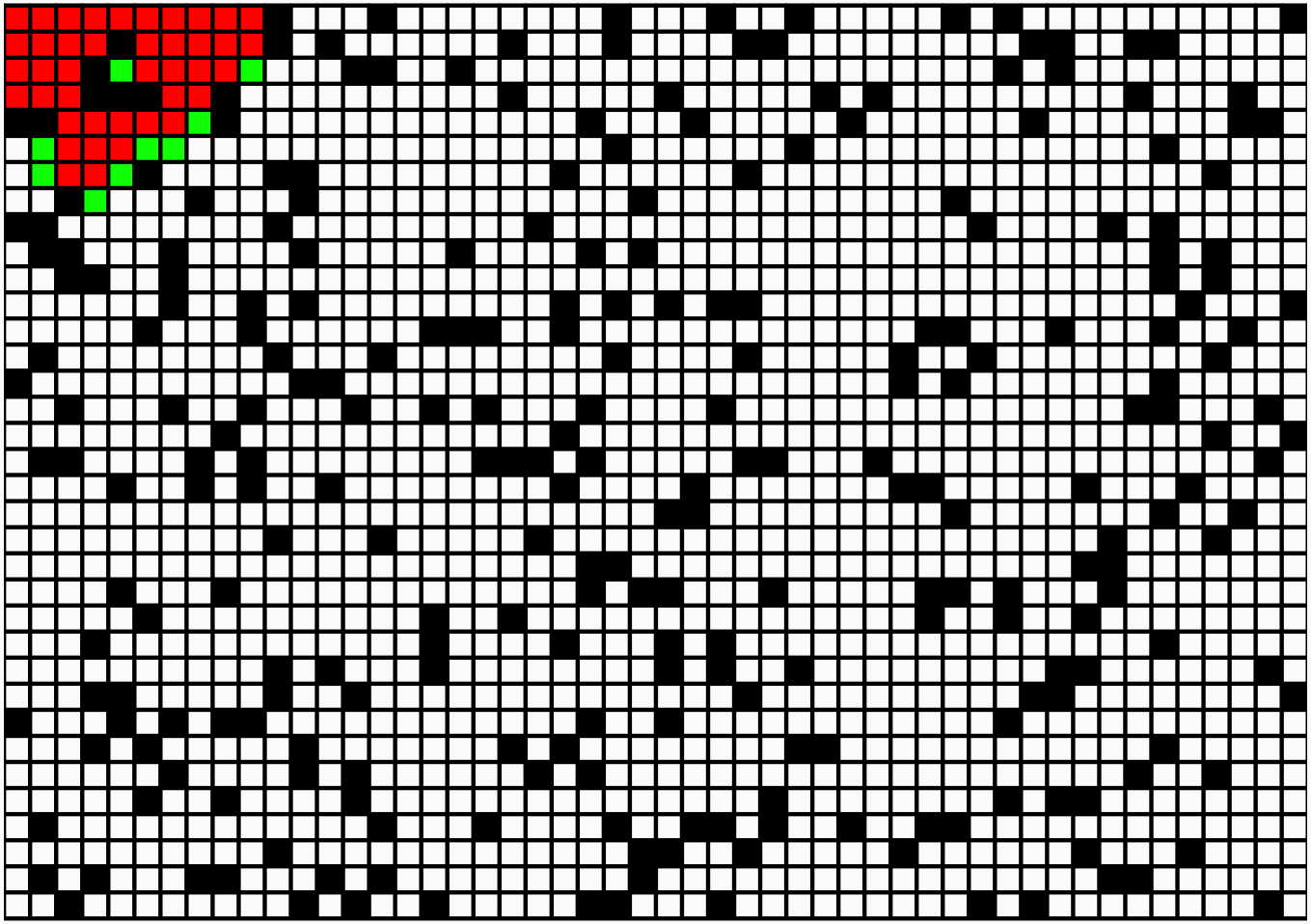
A* Algorithm



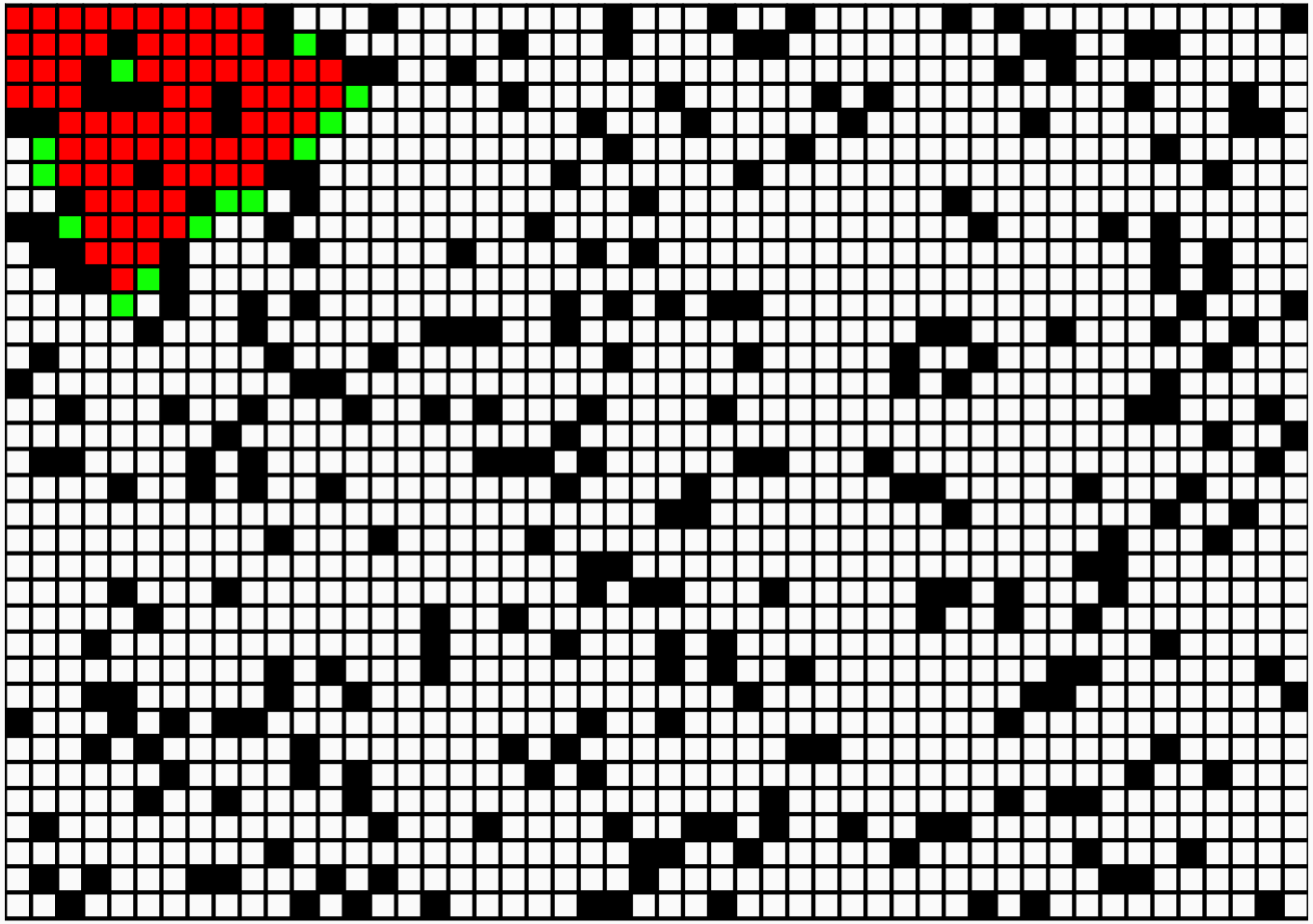
A* Algorithm



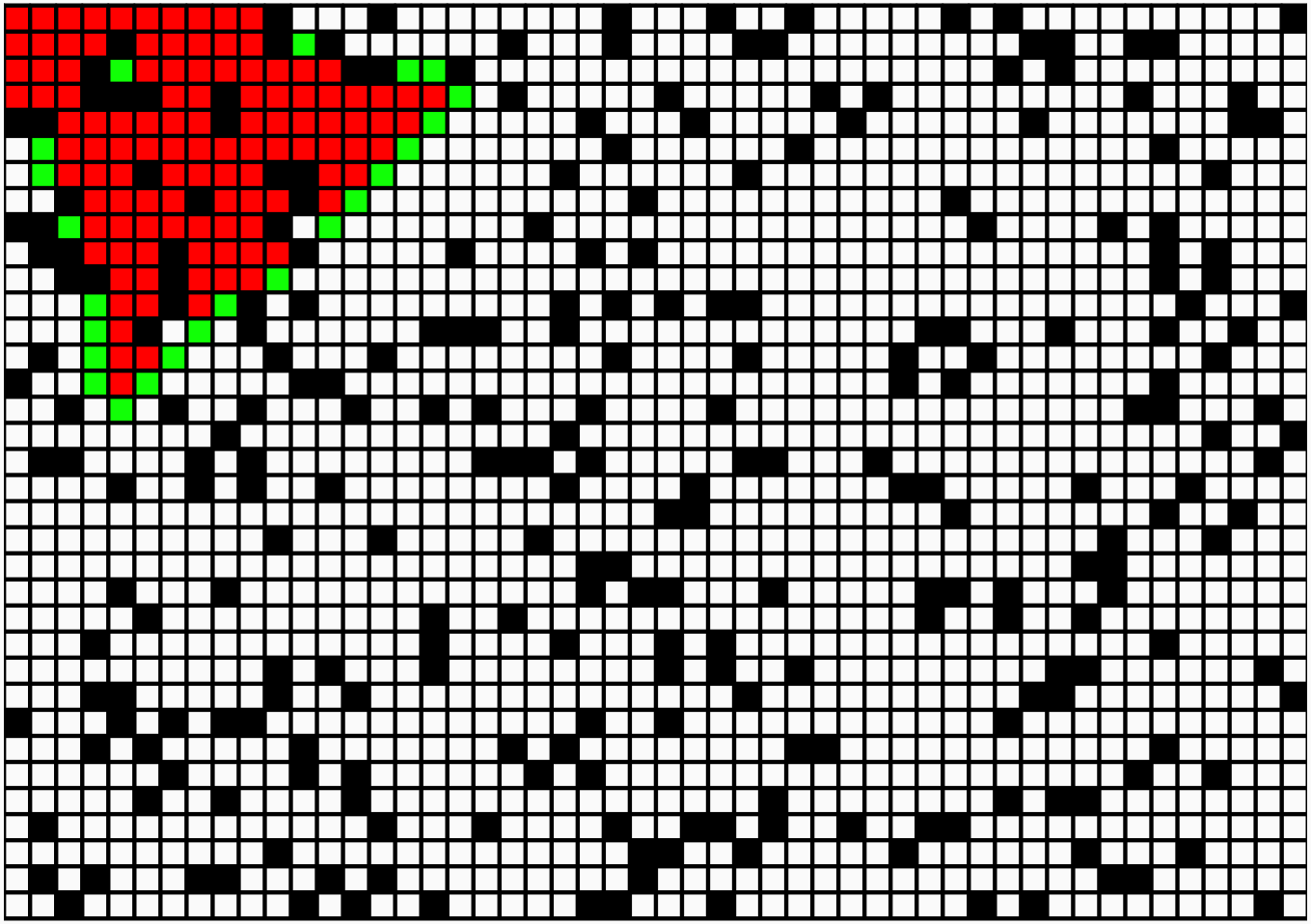
A* Algorithm



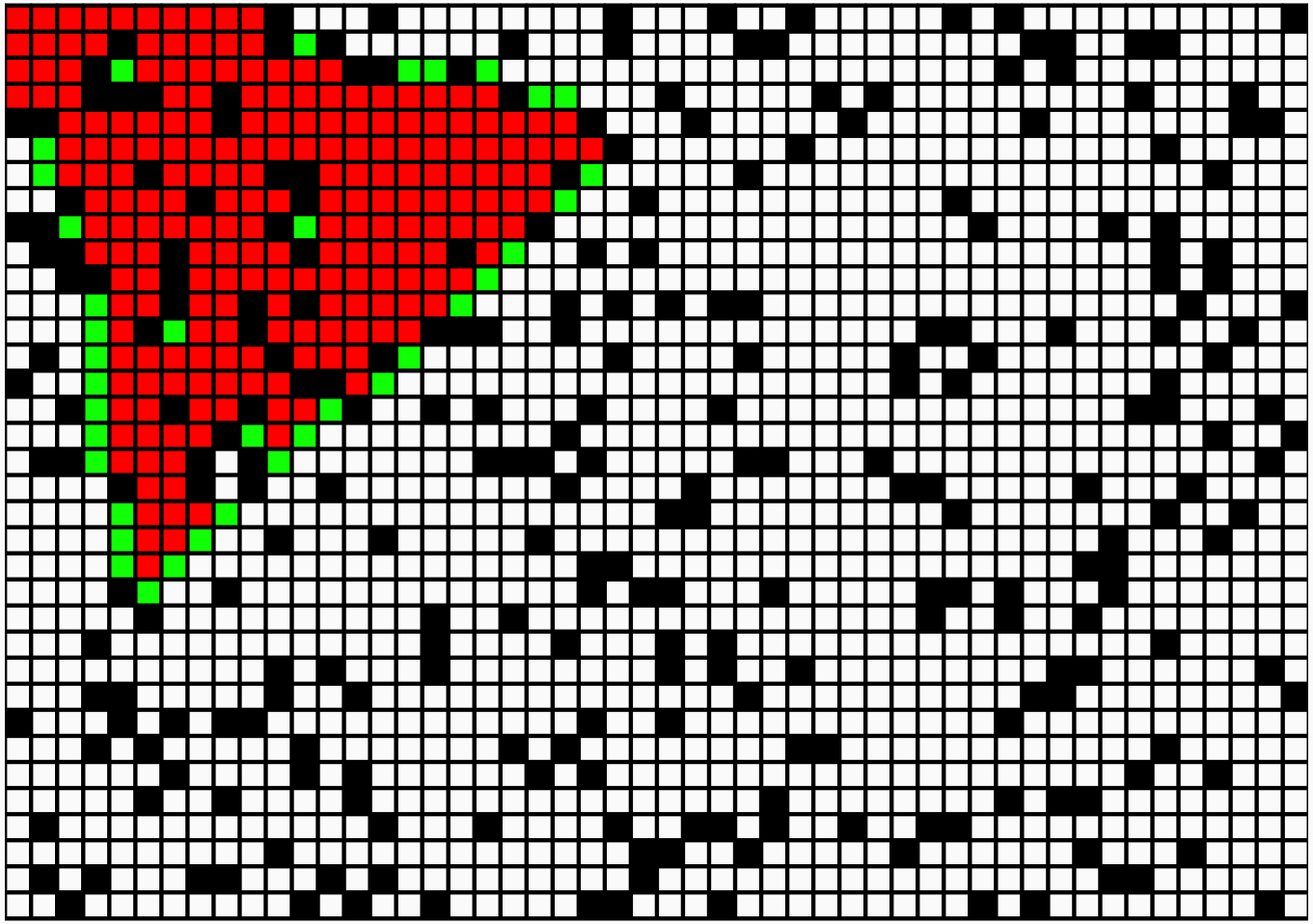
A* Algorithm



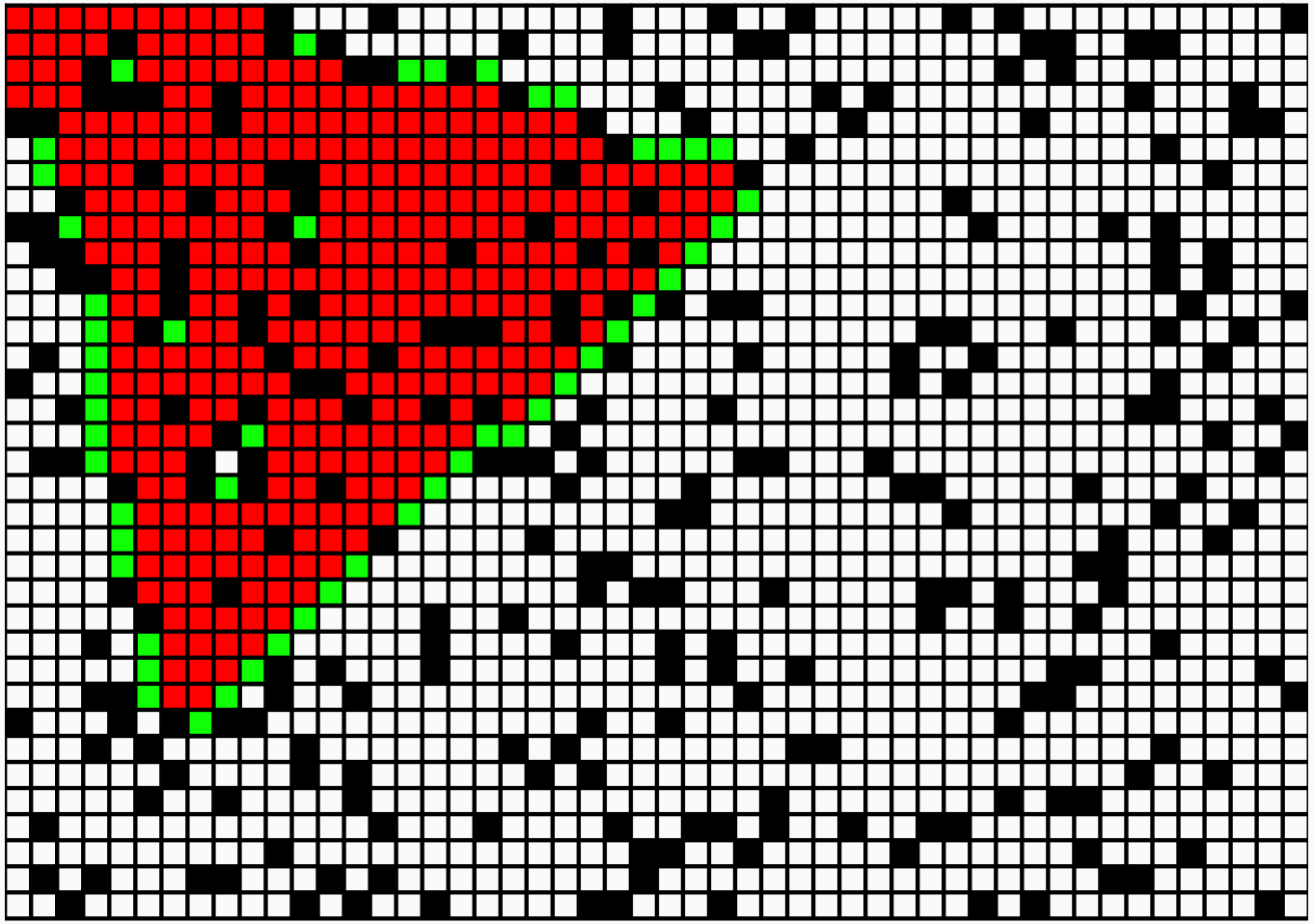
A* Algorithm



A* Algorithm

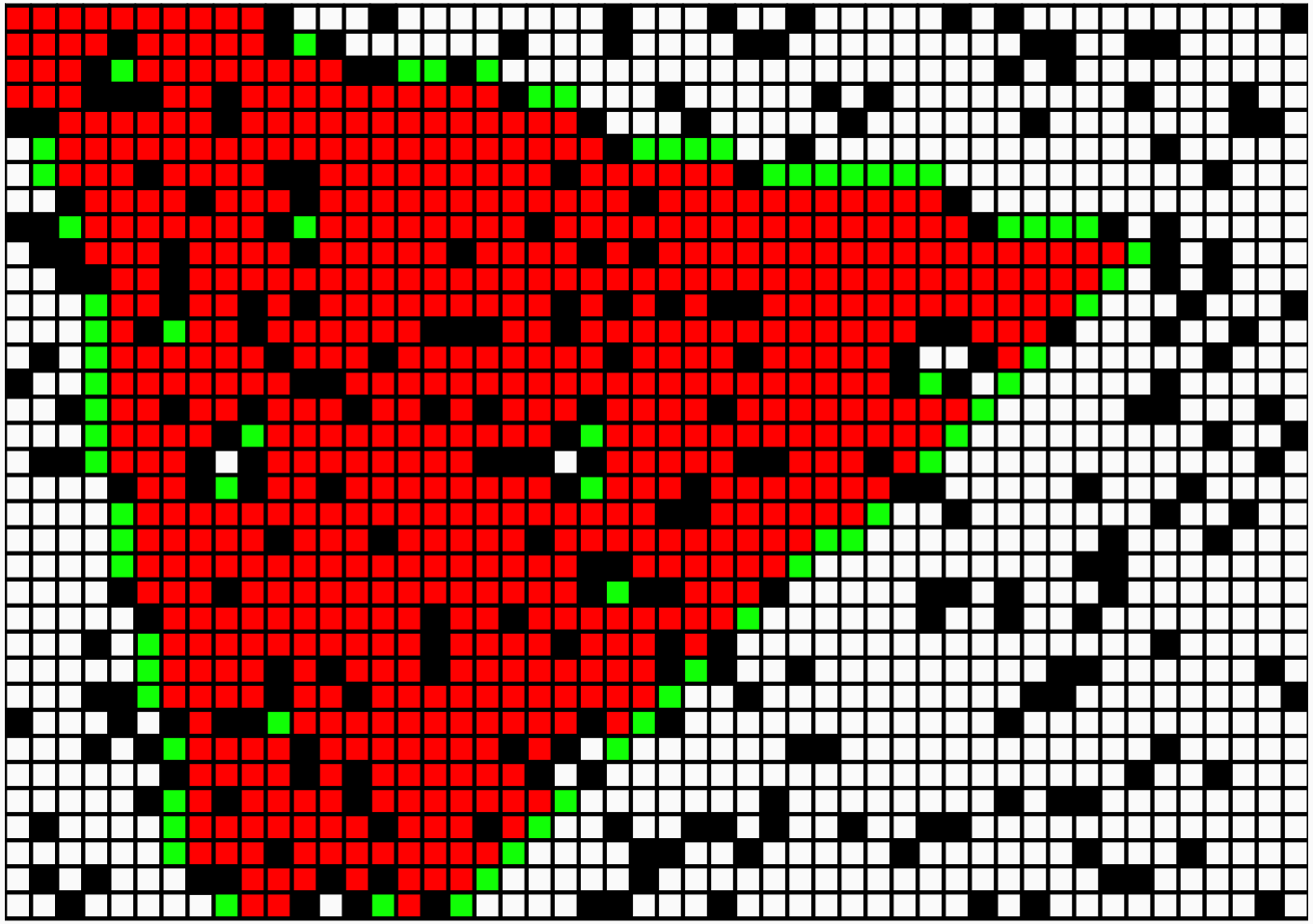


A* Algorithm

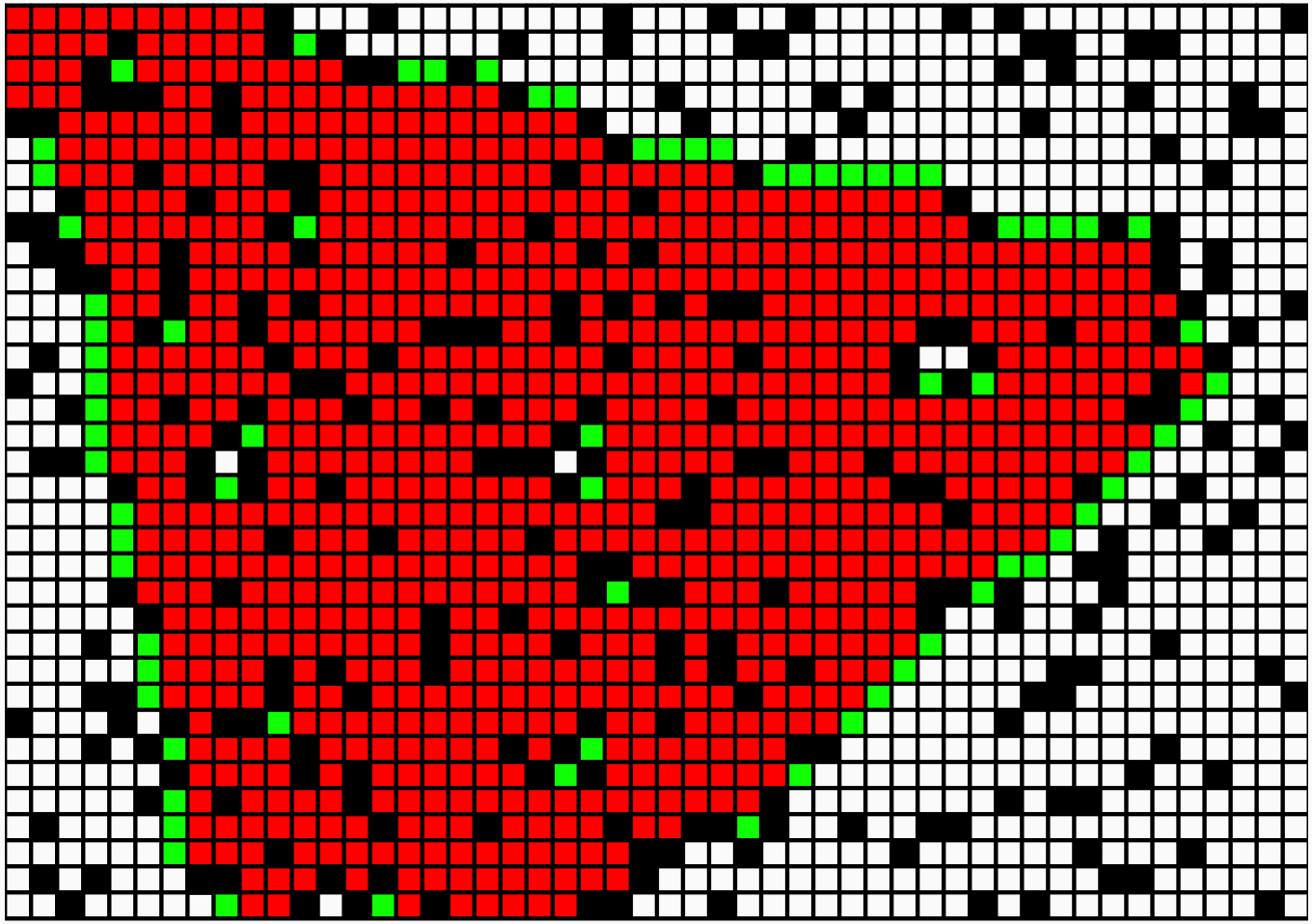


A* Algorithm

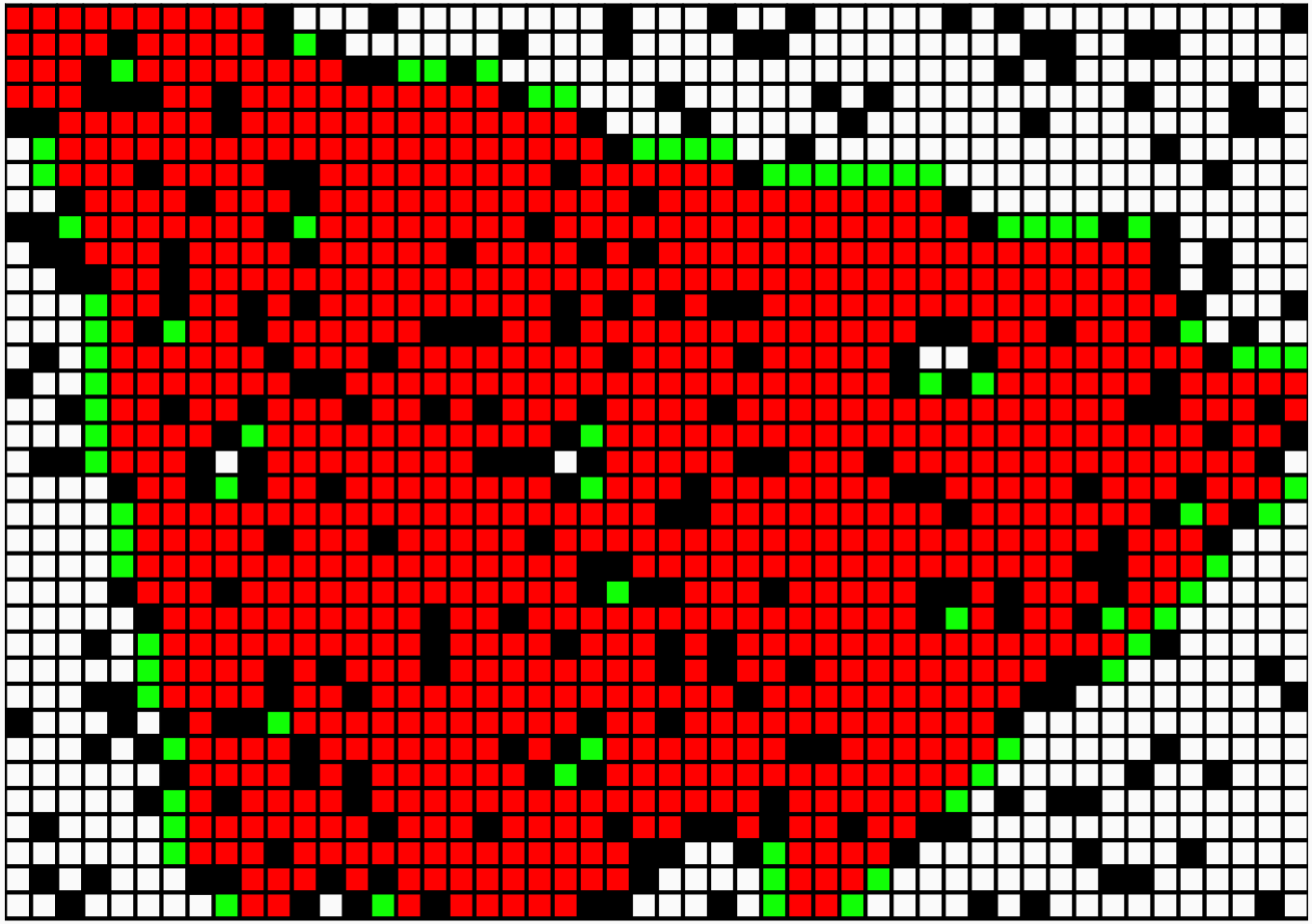
A* Algorithm



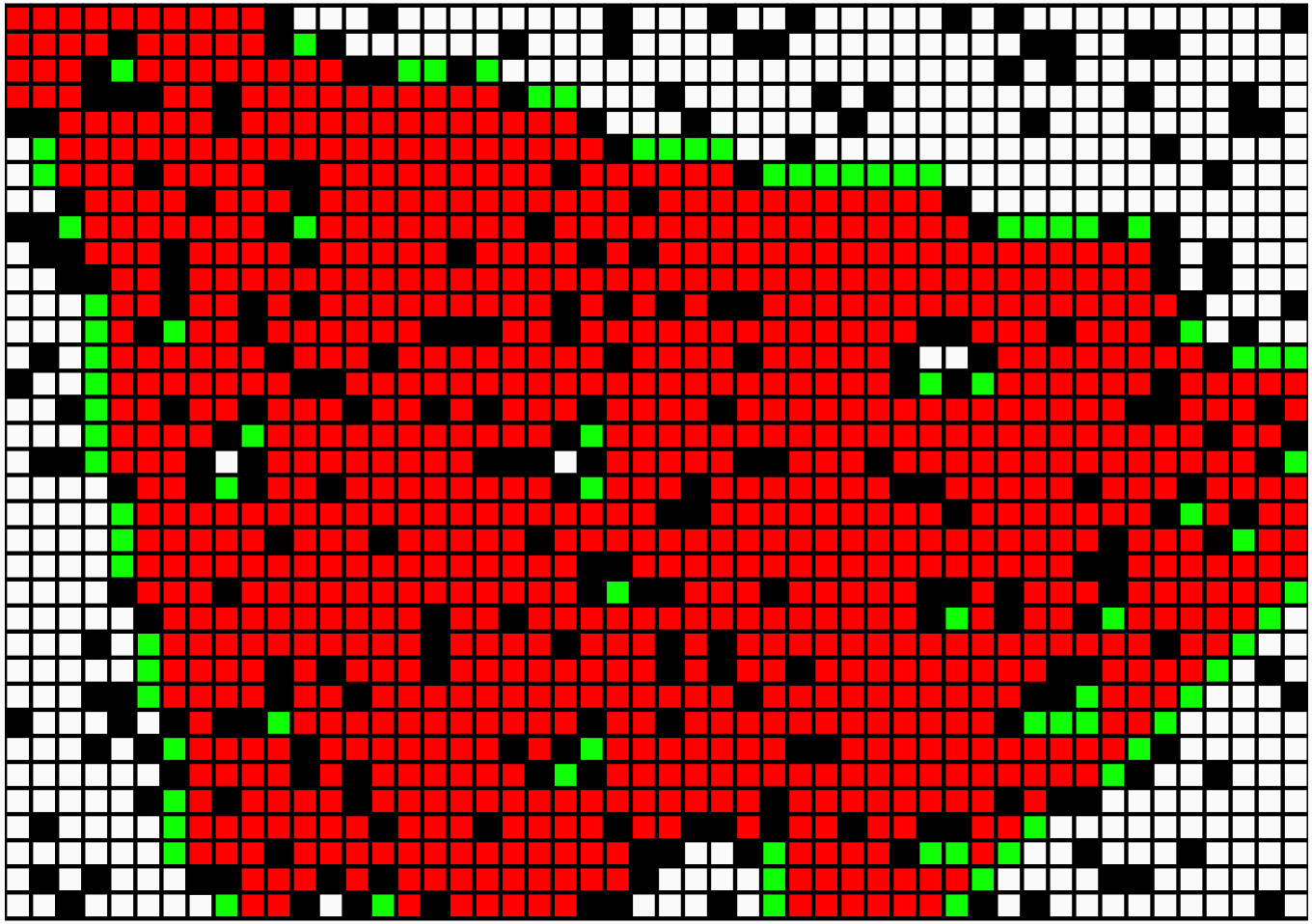
A* Algorithm



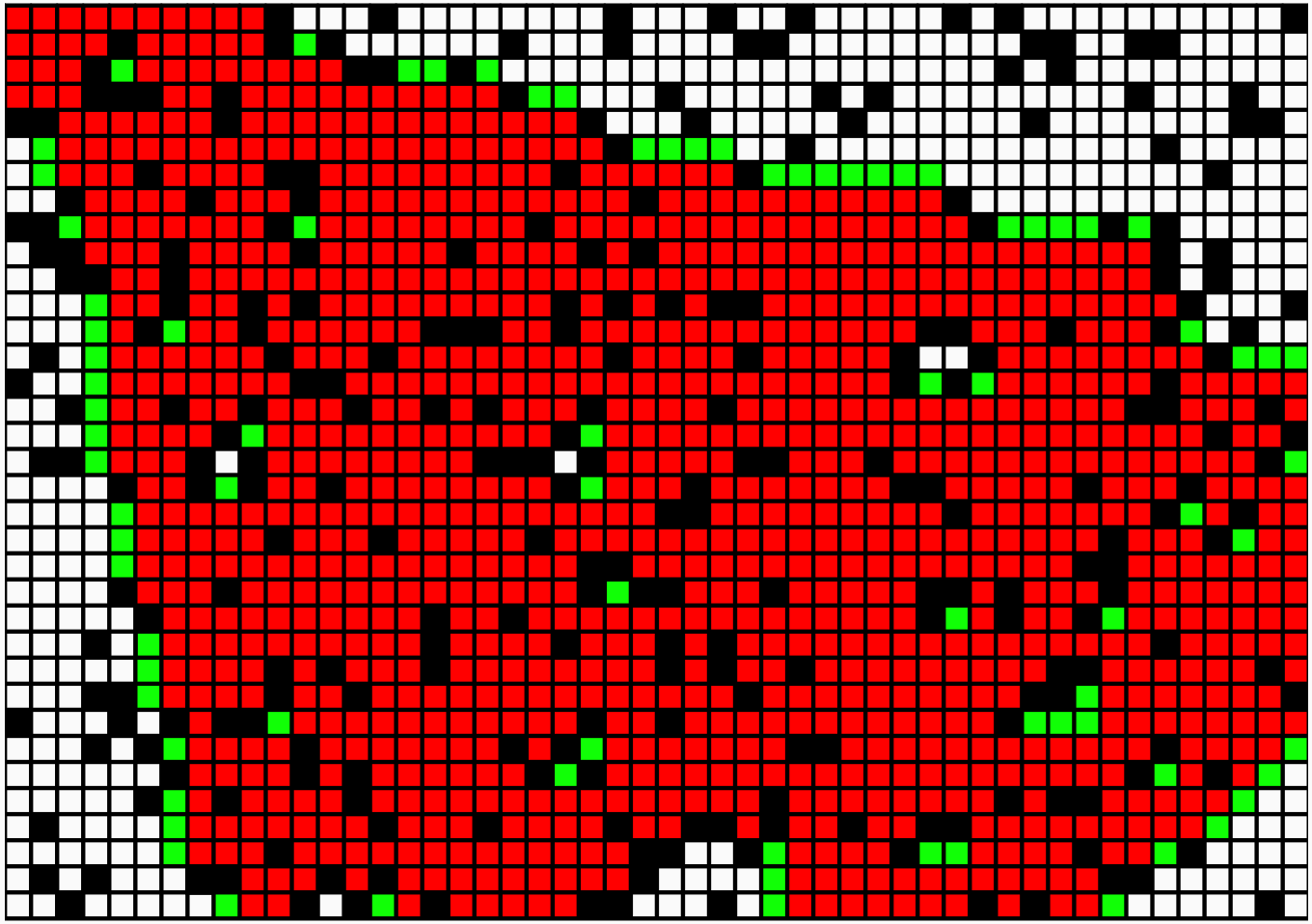
A* Algorithm



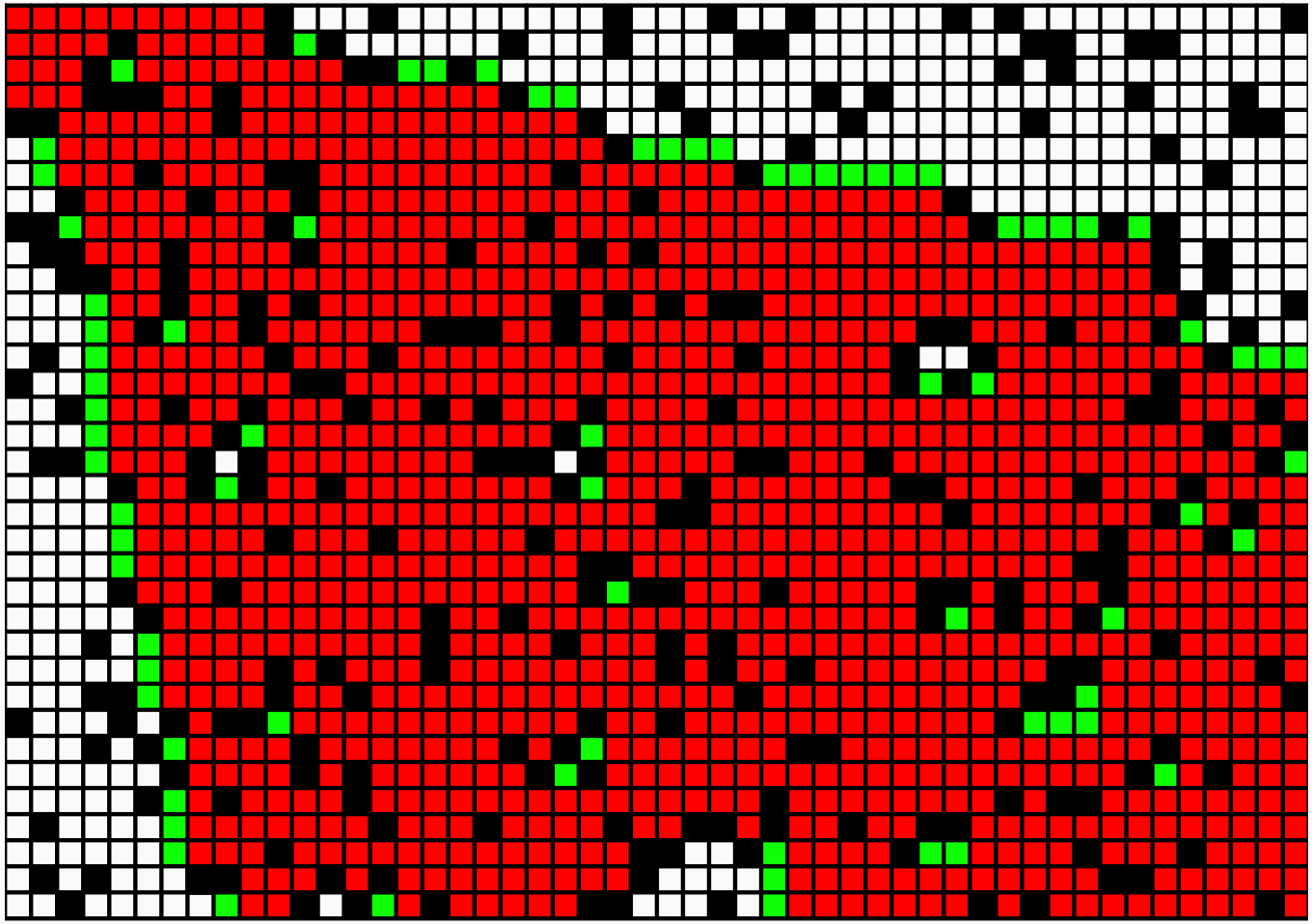
A* Algorithm



A* Algorithm



A* Algorithm



Local Search

Local Search

Base principle

- We have a single solution
- We improve it step by step

Inspiration

- Very natural behaviour
- Given a solution, try to move in the right direction

Local Search

What we need to know:

- How to represent a solution
- What is a move

Characterization:

- Exploitation: Using local information to make small steps
- Exploration: Moving blindly to find better solutions globally

Representing a solution

How to represent a solution?

- Very problem-dependent

Some main approaches:

- Values: binary, integers, reals, strings
- Represent different dimensions as elements of a vector

Examples:

- Real function optimization: vector of reals
  - $(3.5, 2.4)$
- Secret binary vector: vector of 0s and 1s
  - $(0, 1, 1, 0, 0)$
- Shortest path: sequence of vertices

What is a move

What is a move?

Defined by the concept of neighbor:

- Move: replacing a solution by one of its neighbors
  - usually with better fitness

Depending on the representation:

- Defined by distance: $N(s) = \{s' : d(s, s') \leq x\}$
- Defined by operations $\{f_i\}$: $N(s) = \{f_i(s) : \forall\, i\}$

Local Search

Algorithm LocalSearch

Input: domain D, fitness f

Output: solution $x \in D$

1: $x_0 := \textsc{InitialSolution}(D)$
2: $i := 1$
3: **while** not $\textsc{Finished}$ **do**
4: $\quad N := \textsc{Neighbors}(x_i, D)$
5: $\quad x_{i+1} := \textsc{Selection}(N, x_i)$ *// May keep the same solution*
6: $\quad i := i + 1$
7: **end while**
8: **return** x_{i-1}

Local Search

Hill Climbing

Hill Climbing

Algorithm HillClimbingFunc

Input: n, domain D, function $f : D \to \mathbb{R}$

Output: solution $x \in D$

```
1: x := RandomSolution(D)
2: for i = 1 to n − 1 do
3:     c := RandomNeighbor(x)
4:     if f(c) < f(x) then
5:         x := c
6:     end if
7: end for
8: return x
```

Local Search

Algorithm LocalSearch

Input: domain D, fitness f

Output: solution $x \in D$

1: $x_0 := \textsc{InitialSolution}(D)$
2: $i := 1$
3: **while** not $\textsc{Finished}$ **do**
4: $\quad N := \textsc{Neighbors}(x, D)$
5: $\quad x_i := \textsc{Selection}(N, x)$ *// May keep the same solution*
6: $\quad i := i + 1$
7: **end while**
8: **return** x_{i-1}

Local Search

Greedy Local Search

Greedy Local Search

Idea:

- Choose the neighbor minimizing the fitness
- Stop once there is no improvement

Properties:

- Pure exploitation, no exploration
- Can be inefficient (depending on the neighborhood)
- Very general
- Can get trapped in local minima

Greedy Local Search

Algorithm GreedyLocalSearch

Input: domain D, fitness f

Output: solution $x \in D$

1: $x_0 := \textsc{RandomSolution}(D)$
2: $i := 1$
3: **while true do**
4: $\quad N := \textsc{Neighbors}(x, D)$
5: $\quad x_i := \text{argmin}\{f(c) : c \in N\}$
6: $\quad$ **if** $f(x_i) \geq f(x_{i-1})$ **then**
7: $\quad\quad$ **break**
8: $\quad$ **end if**
9: $\quad i := i + 1$
10: **end while**
11: **return** x_i

Greedy Local Search (Python)

```python
def greedy_local_search(D, f):
    x = D.sample()
    while True:
        N = D.neighbors(x)
        fc, c = min( (f(c), c) for c in N )
        if f(c) < f(x):
            x = c
        else:
            break
    return x

D = DomainBinVec(n)  # n=10
f = lambda x: sum(secret[i] != x[i] for i in range(n) )
x = greedy_local_search(D, f)
print(f"f({x}) = {f(x)}")
# f((0, 0, 0, 0, 1, 1, 1, 1, 1, 0)) = 0
```

Greedy Local Search (Python)

```python
@dataclass
class DomainBinVec:
    n: int
    def sample(self):
        return tuple(randrange(2) for i in range(self.n))
    def neighbor(self, x):
        i = randrange(self.n)
        return x[:i] + (1-x[i],) + x[i+1:]
    def neighbors(self, x):
```

```
        L = []
        for i in range(n):
            L.append(x[:i] + (1-x[i],) + x[i+1:])
        return L
```

Local Search

Examples

Examples

Examples

- Minimizing a function
- TSP
- Group work scheduling
- Maze solving

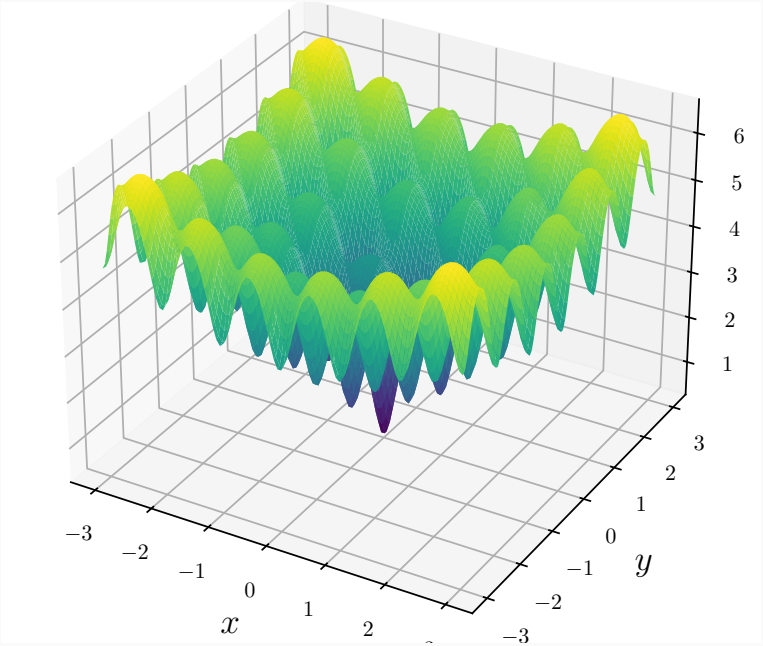
What we need:

- Representation
- Neighborhood

Example 1: Minimizing a Function

Input:

- Function to minimize (or maximize)
- Domain $D \subseteq \mathbb{R}^d$



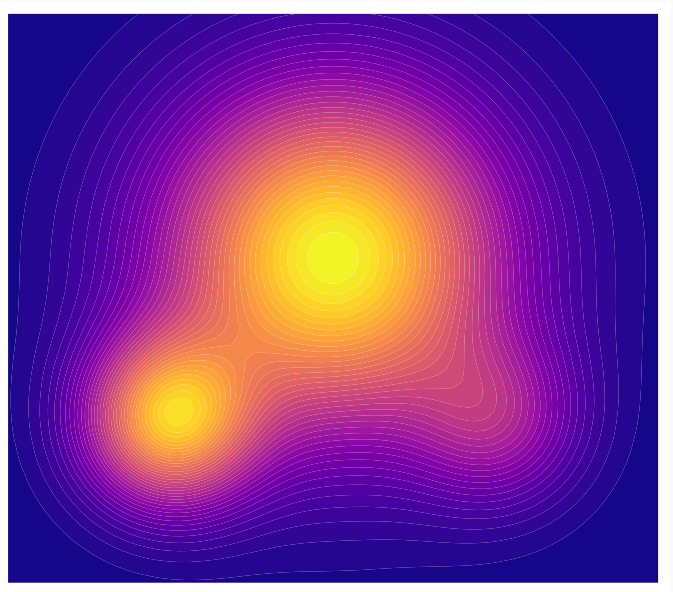
Example 1: Minimizing a Function

Input:

- Function to minimize (or maximize)
- Domain $D \subseteq \mathbb{R}^d$

Example:

- Sum of Gaussians
- $f(x) = \sum_i e^{-(x-x_i)^2-(y-y_i)^2}$



Example 1: Minimizing a Function

Solution:

- Vector in $\mathbb{R}^d$
- Precision?

Neighborhood:

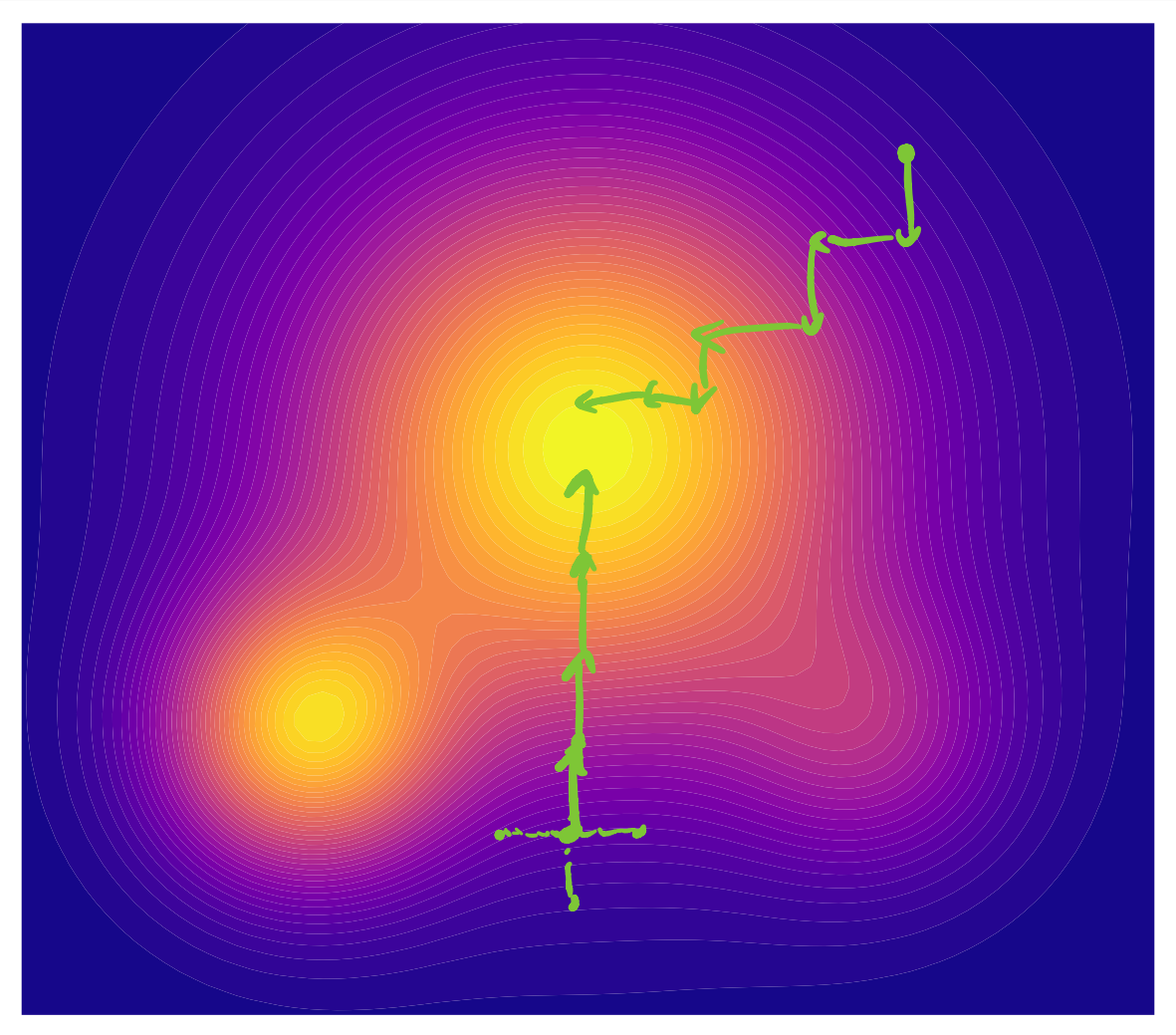
1. Single change at i: $x'_i = x_i \pm \lambda, \quad x'_j = x_j, j \neq i;$
2. Multiple change: $x'_j = x_j + \{-\lambda, 0, \lambda\}$
3. Gradient descent: $x' = x - \lambda \nabla f(x)$

(not local search)

number
2^d
3^d

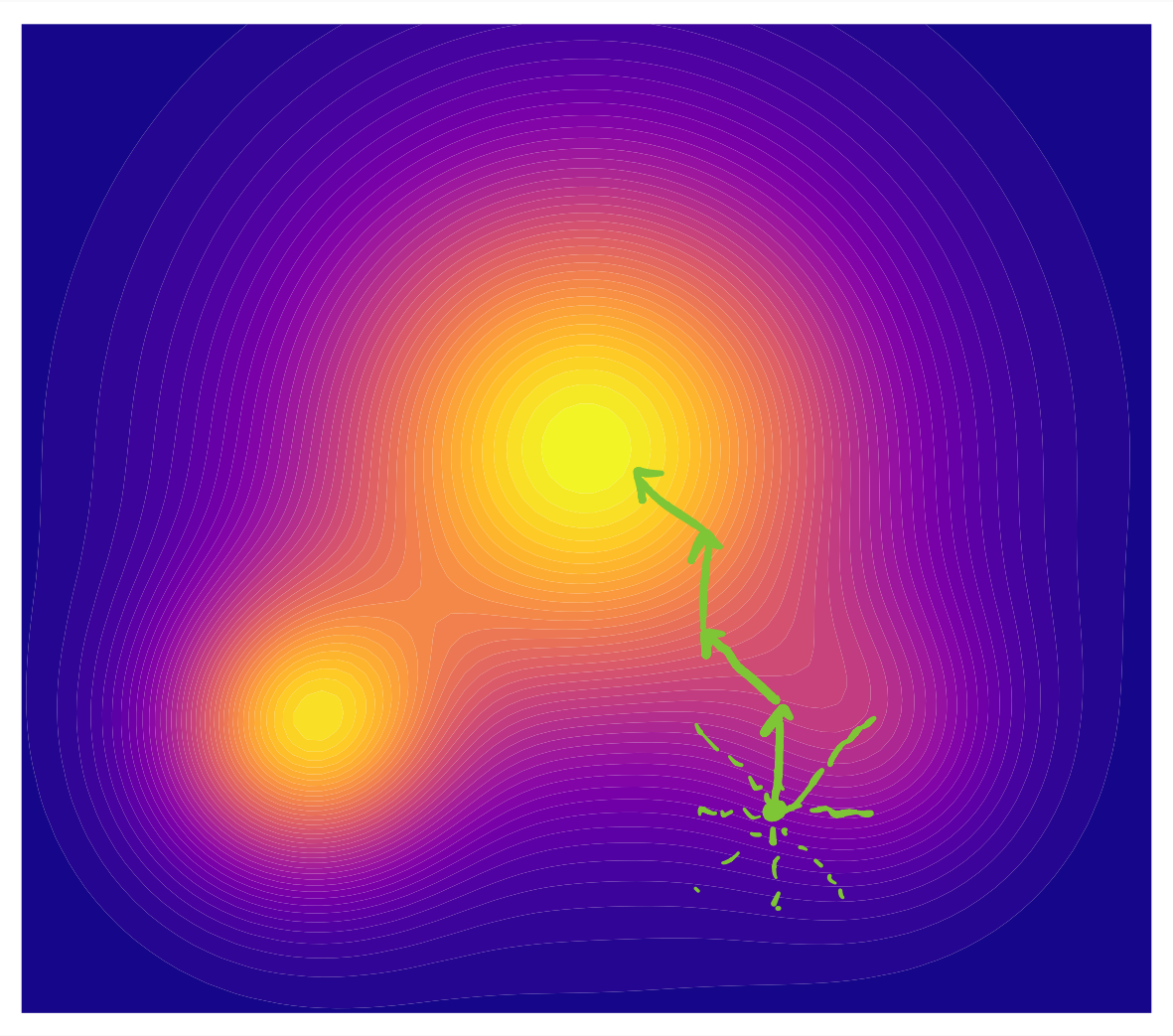
Example 1: Minimizing a Function

Operator 1

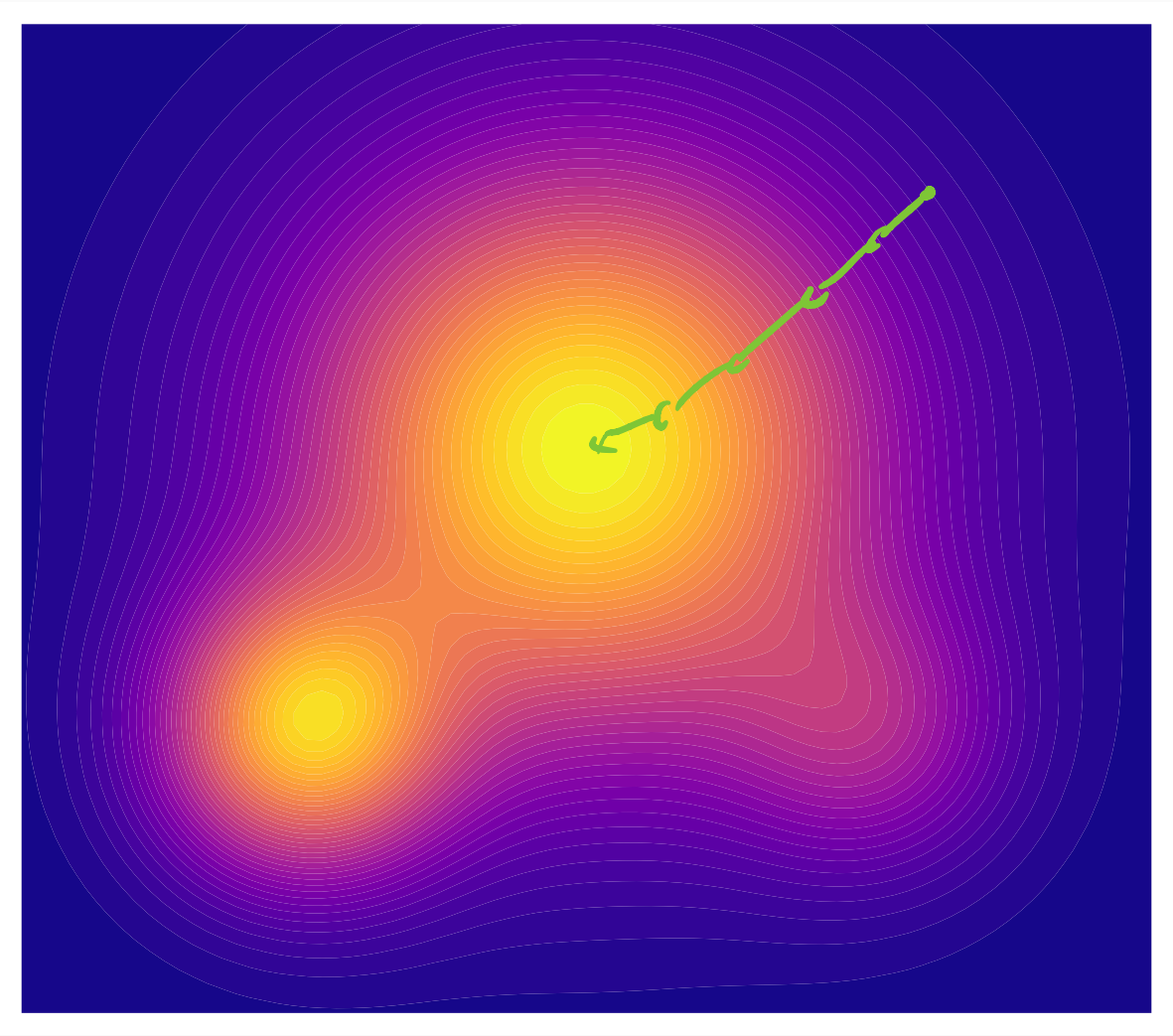


Example 1: Minimizing a Function

Operator 2



Example 1: Minimizing a Function



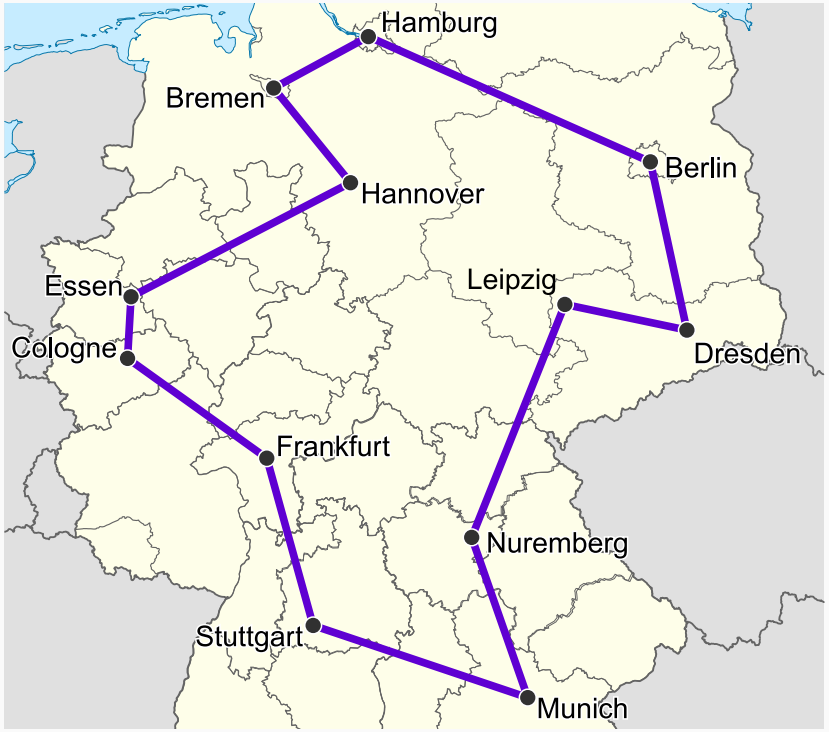
Example 2: Traveling Salesperson Problem

Input:

- Complete graph G
- Edge lengths w (with Δ-inequality)

Goal: Find a cycle

- through each vertex
- with minimum length



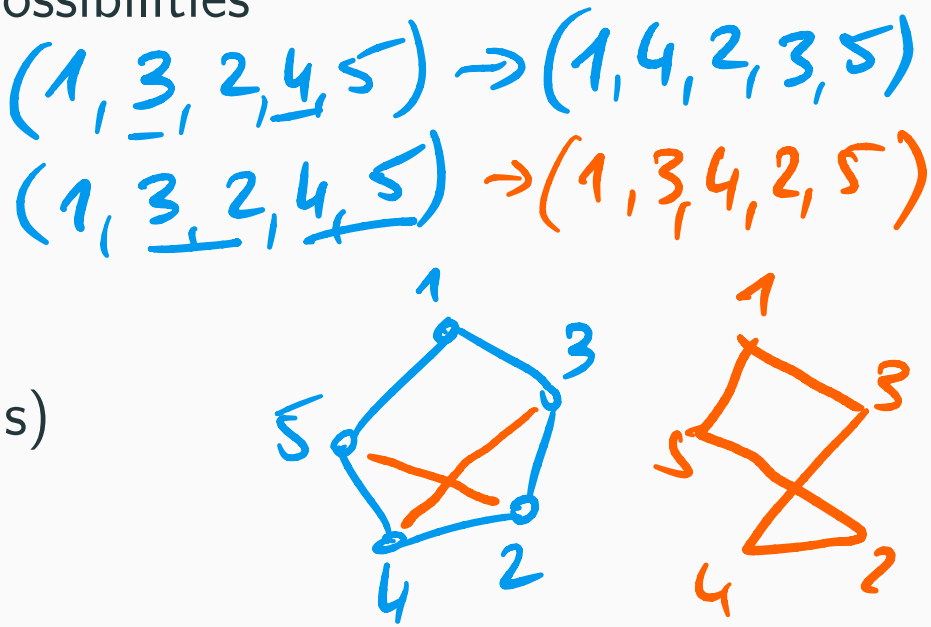
Example 2: Traveling Salesperson Problem

Solution:

- Sequence of vertices (permutation)

Neighborhood operators: many possibilities

1. Switch two vertices
2. Exchange two edges (2-opt)
3. Exchange k edges (k-opt)
4. Lin-Kernighan (alternating trails)



Example 2: Traveling Salesperson Problem

Solution:

- Sequence of vertices (permutation)

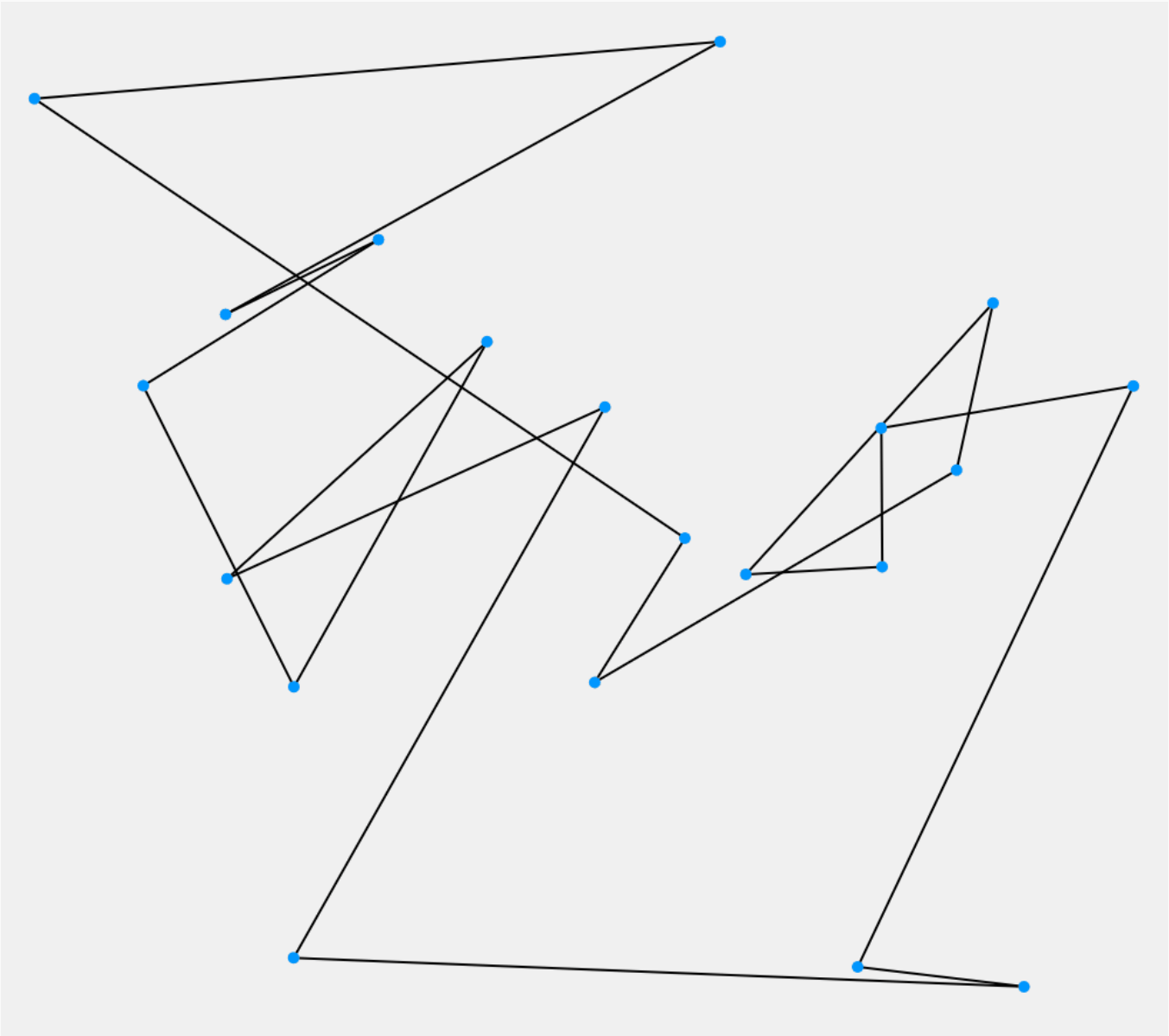
Neighborhood operators:

- E.g. switch two vertices
- Implies a neighborhood

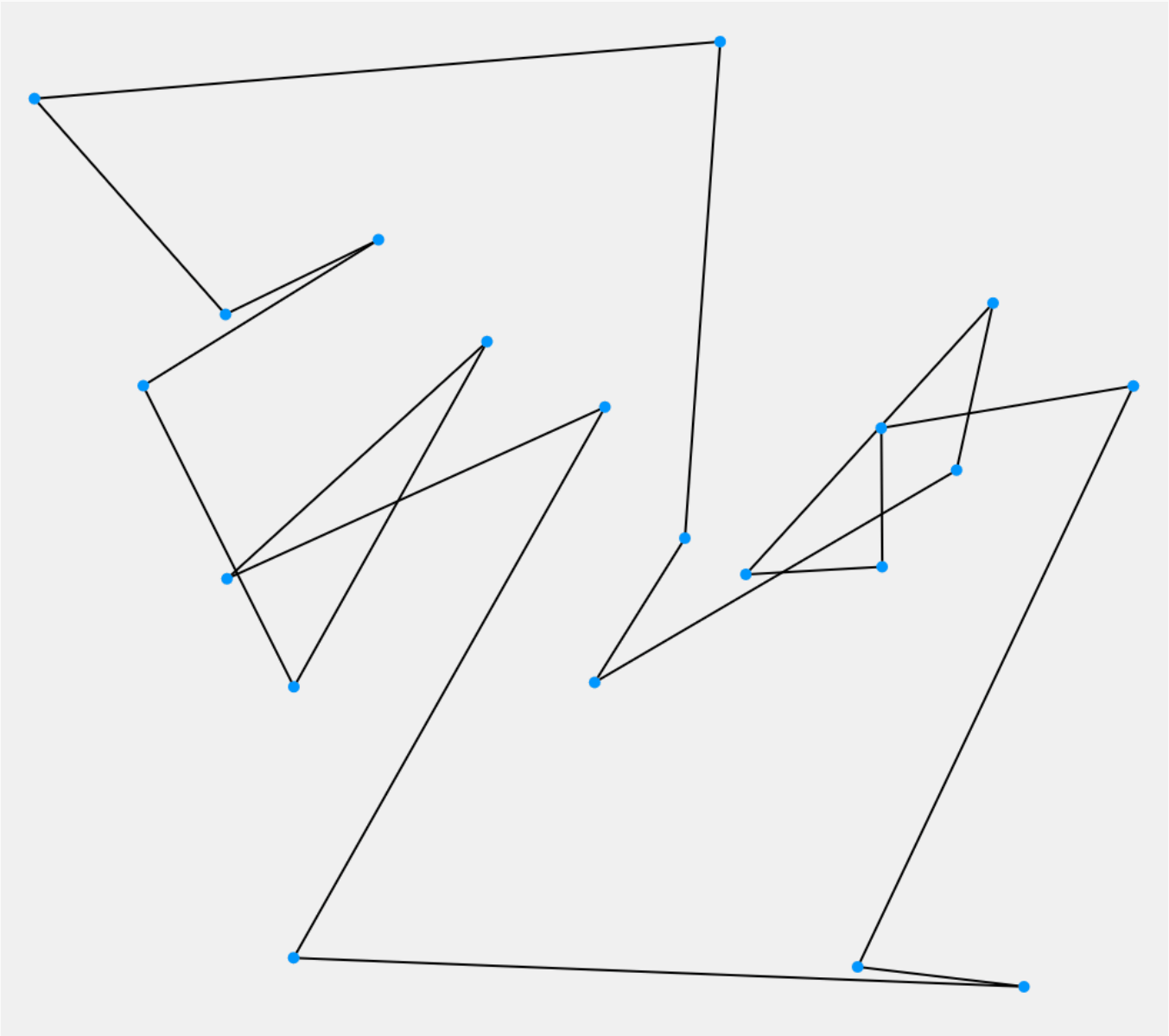
Complexity:

- Each iteration, look at all possible switches
- How many? $O(n^2)$
- What about 2-opt and k-opt? $O(n^2)$ $O(n^k)$

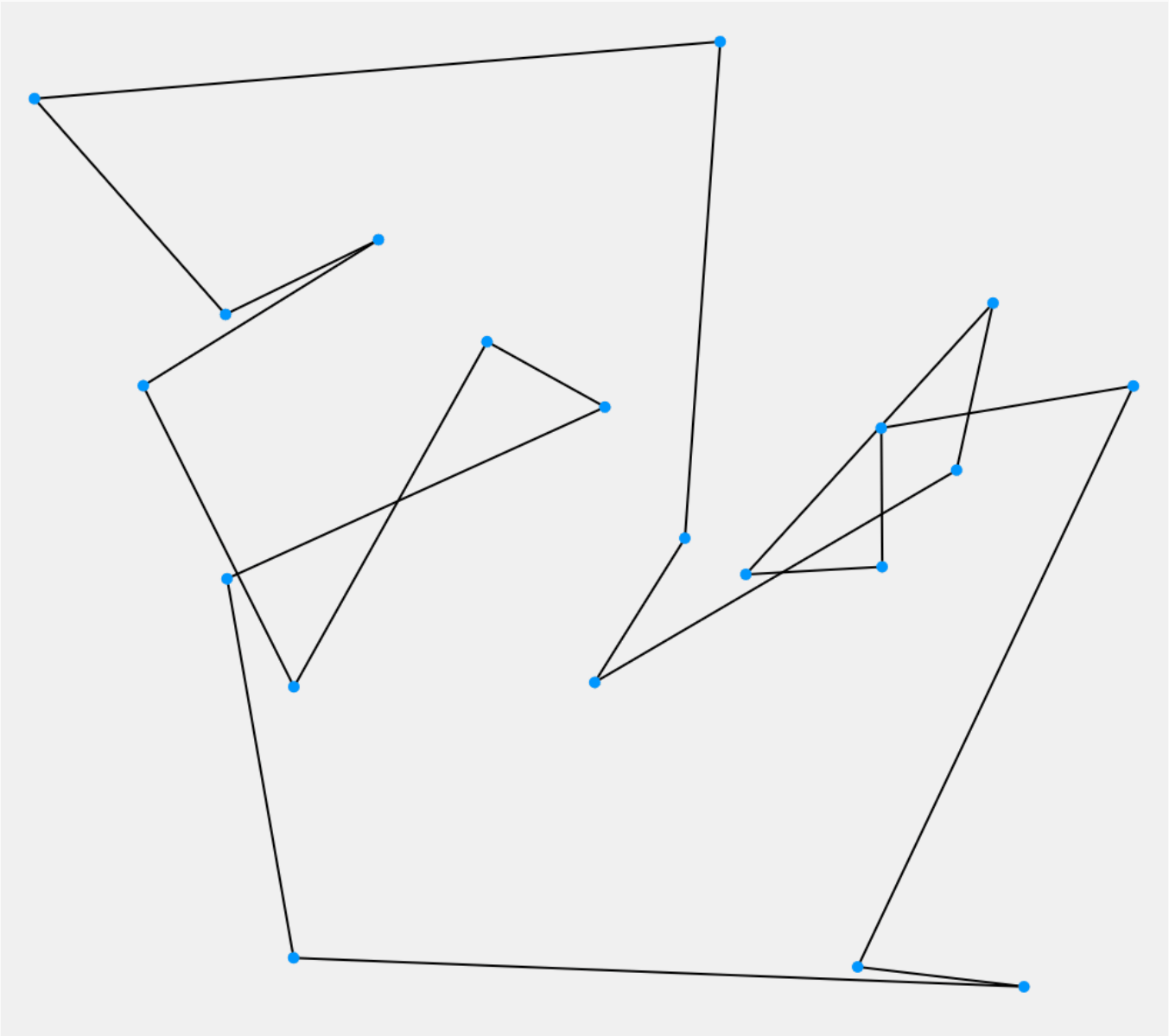
Example 2: Traveling Salesperson Problem



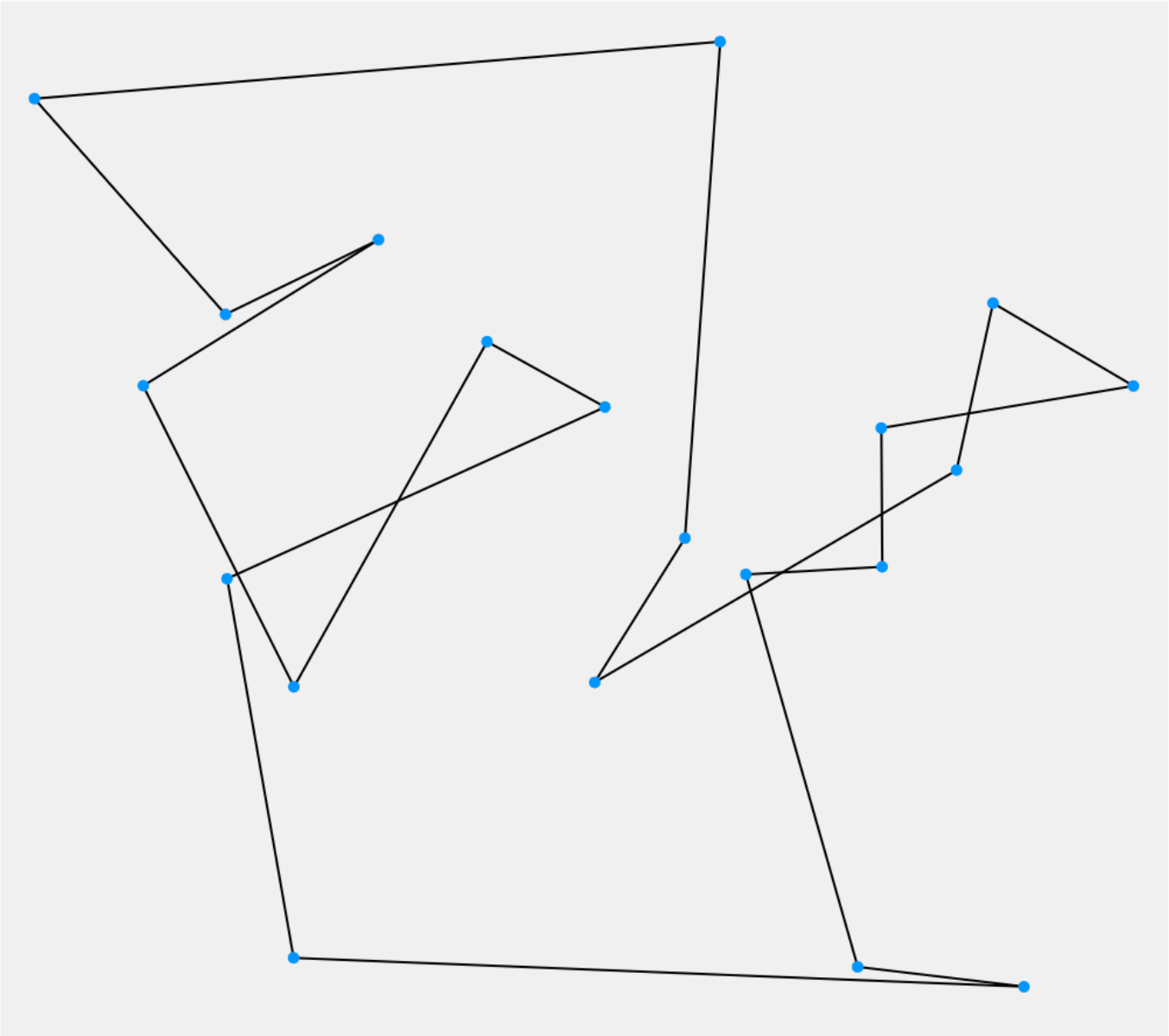
Example 2: Traveling Salesperson Problem



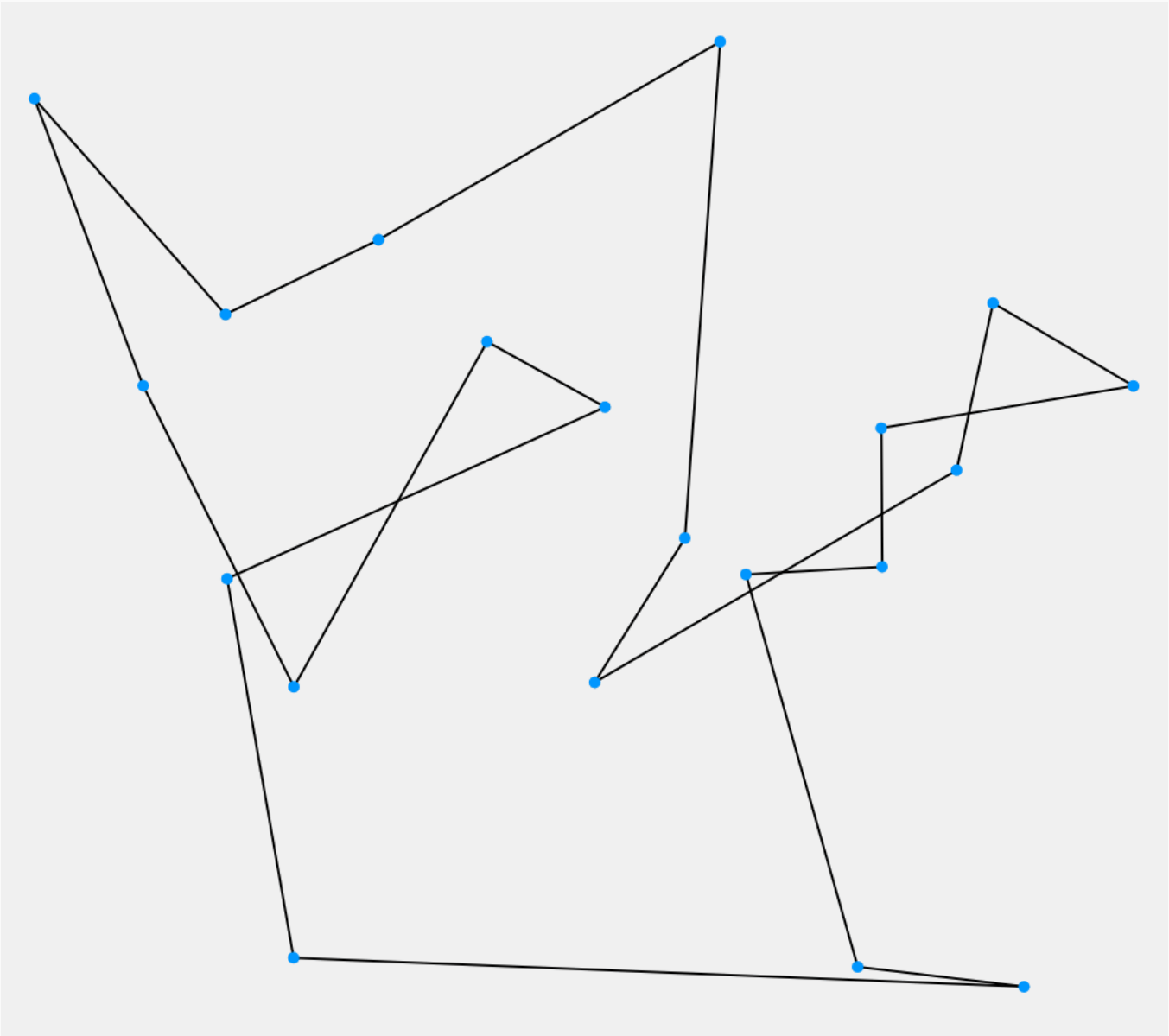
Example 2: Traveling Salesperson Problem



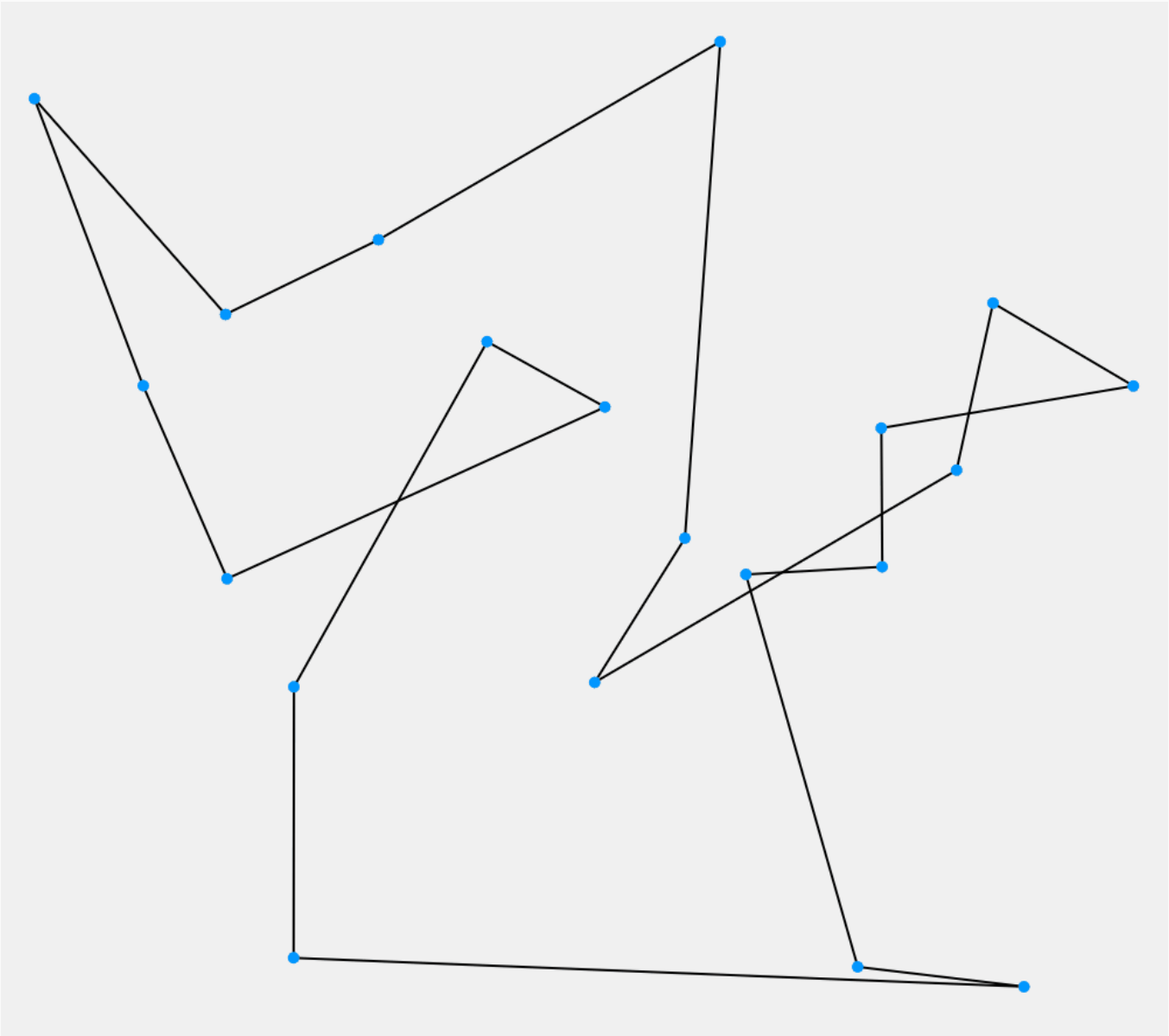
Example 2: Traveling Salesperson Problem



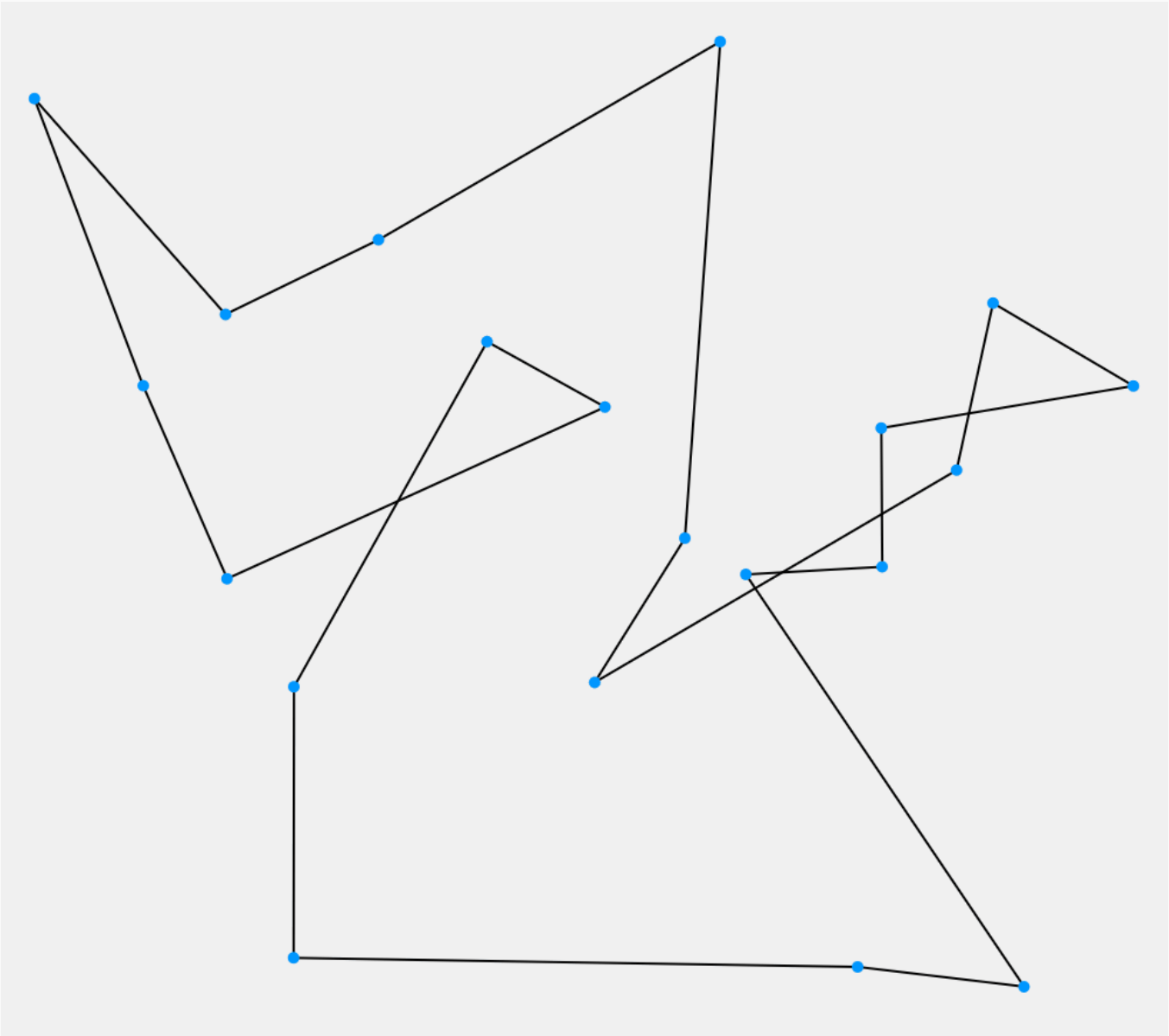
Example 2: Traveling Salesperson Problem



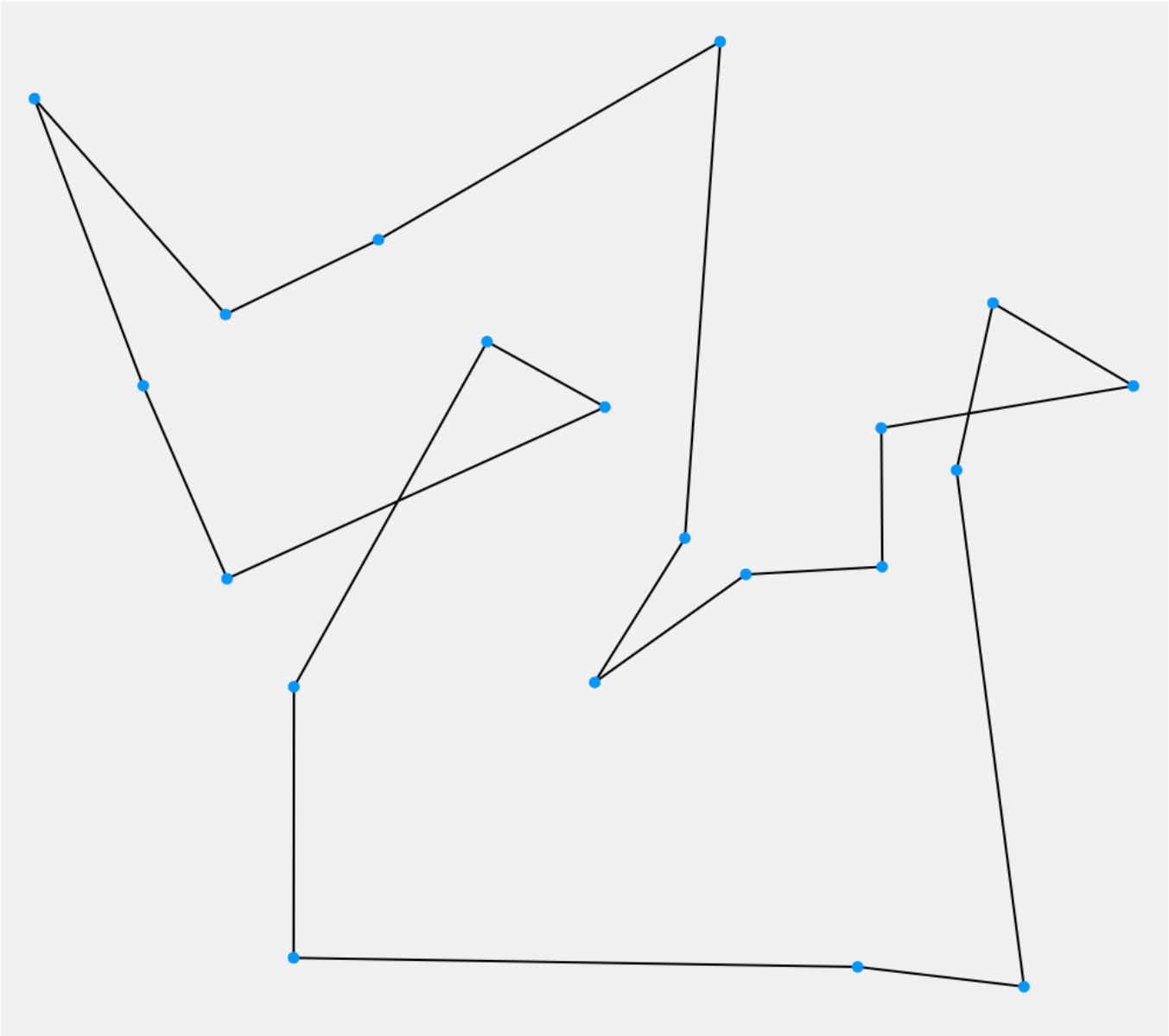
Example 2: Traveling Salesperson Problem



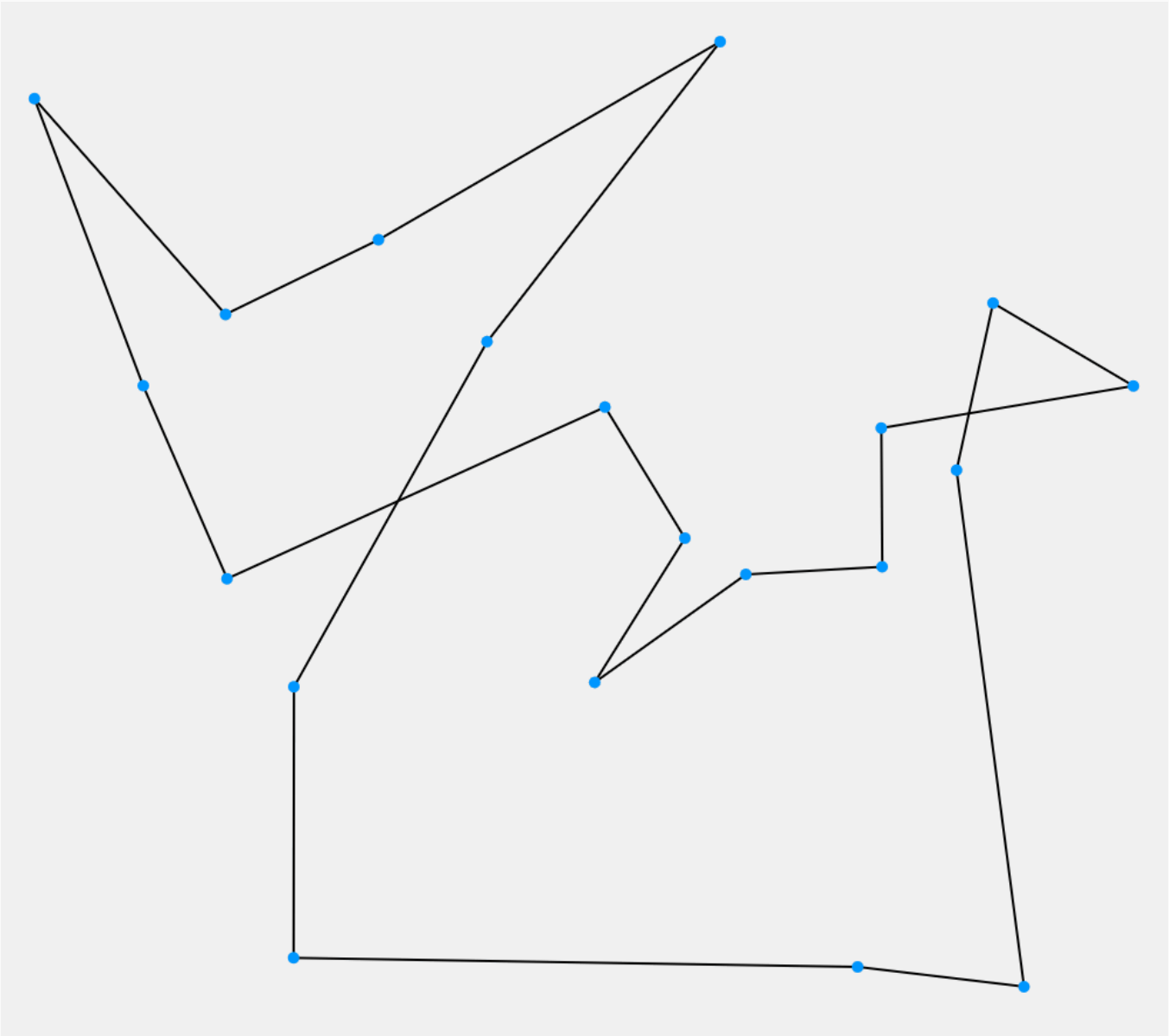
Example 2: Traveling Salesperson Problem



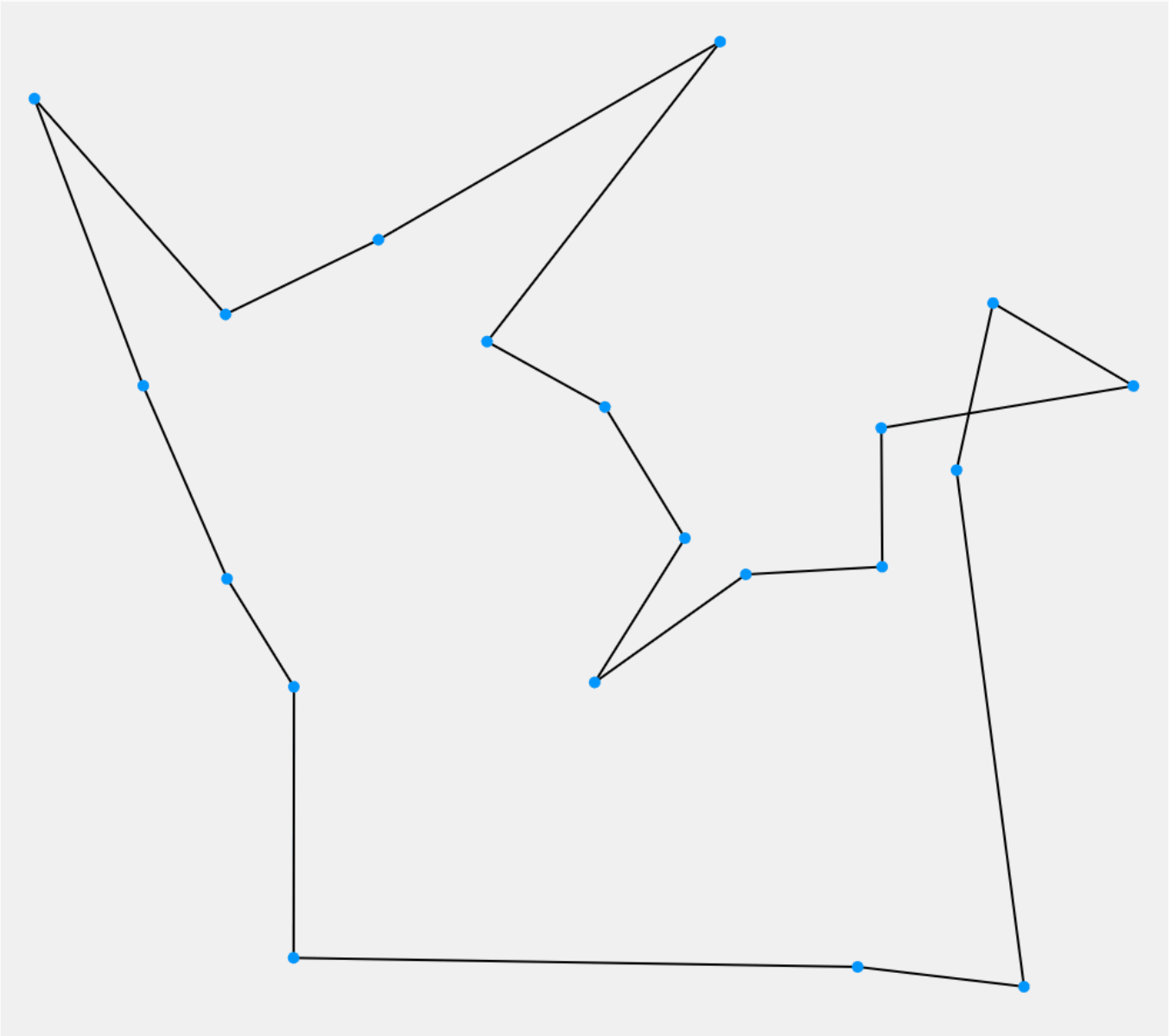
Example 2: Traveling Salesperson Problem



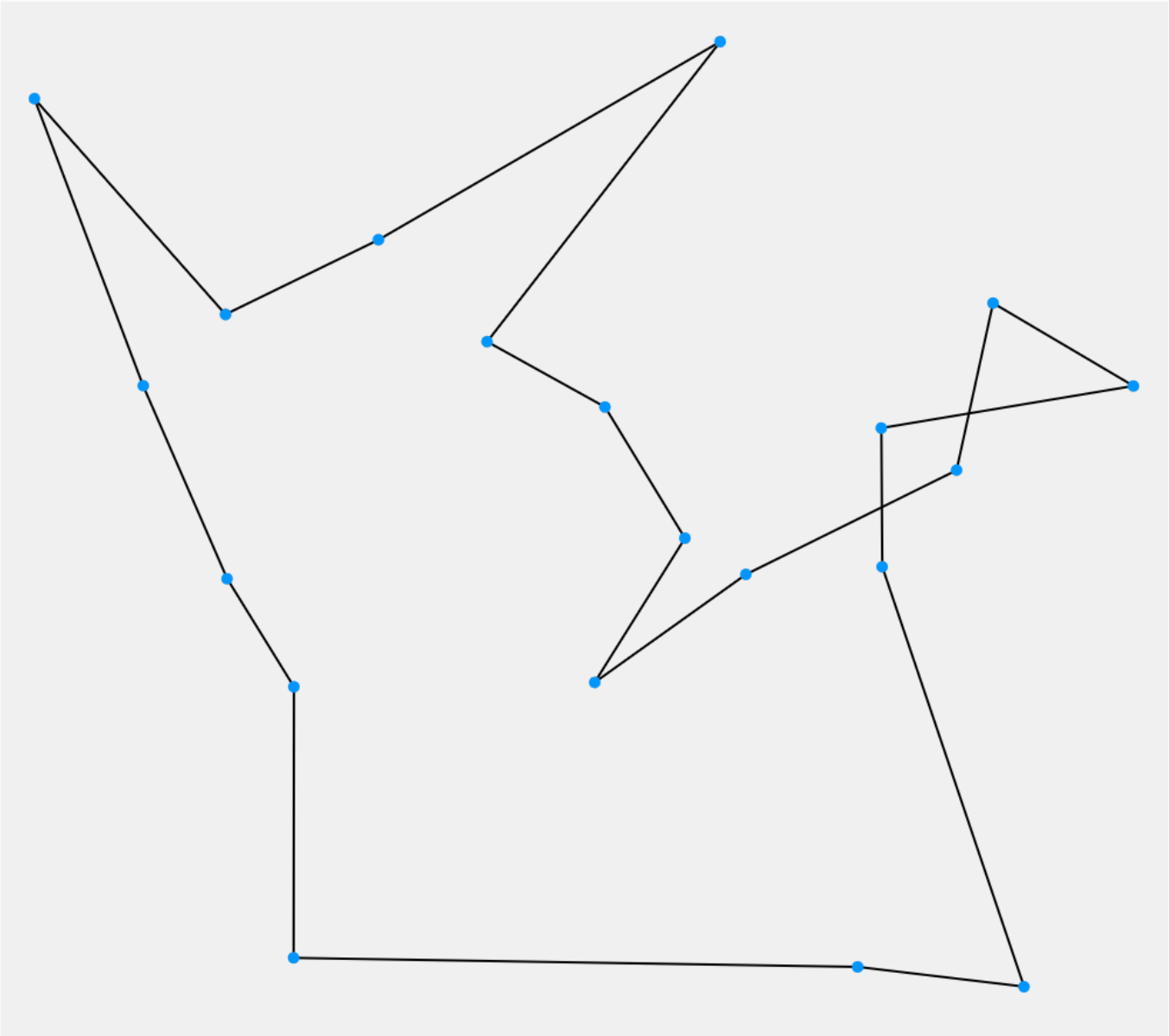
Example 2: Traveling Salesperson Problem



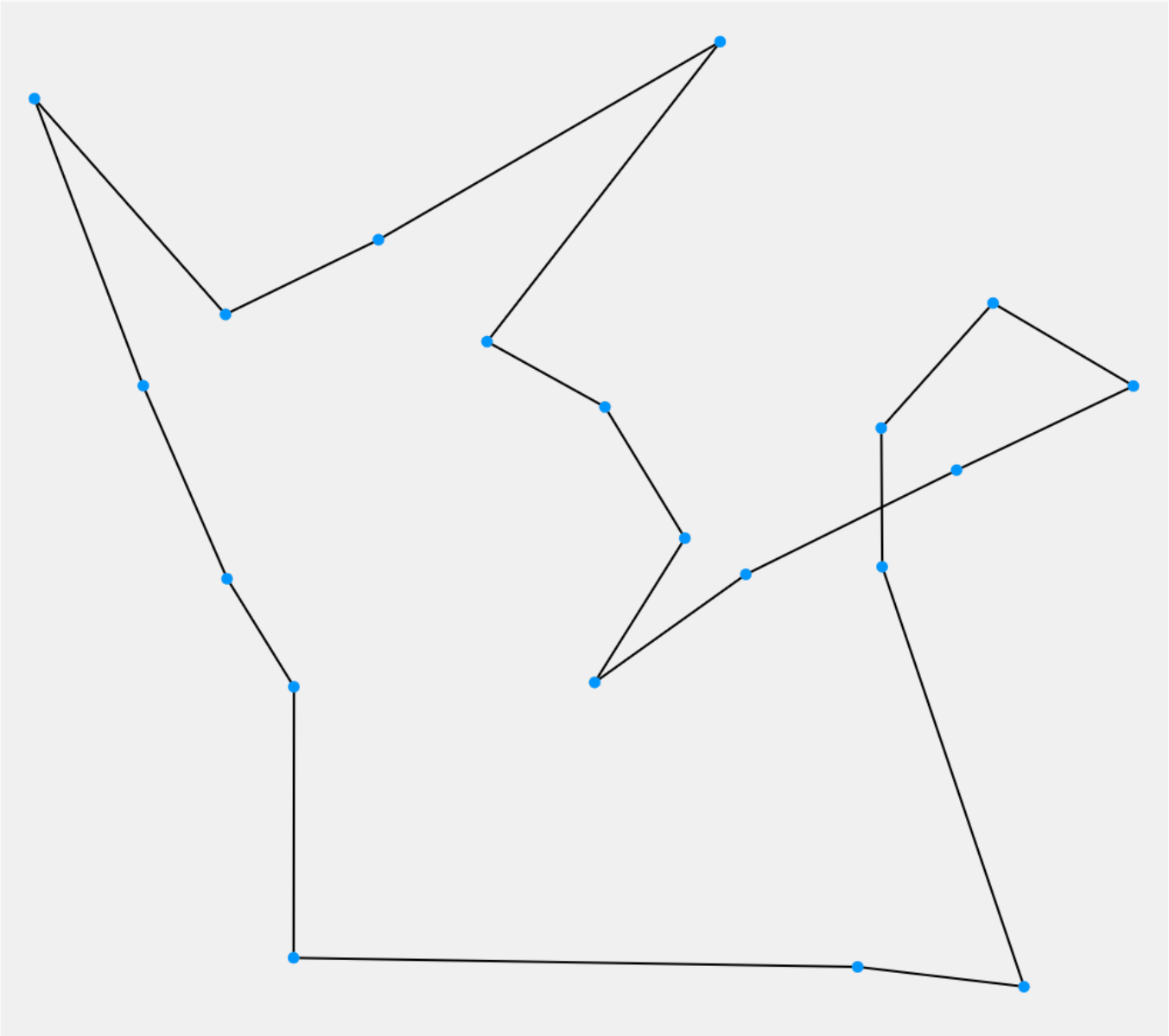
Example 2: Traveling Salesperson Problem



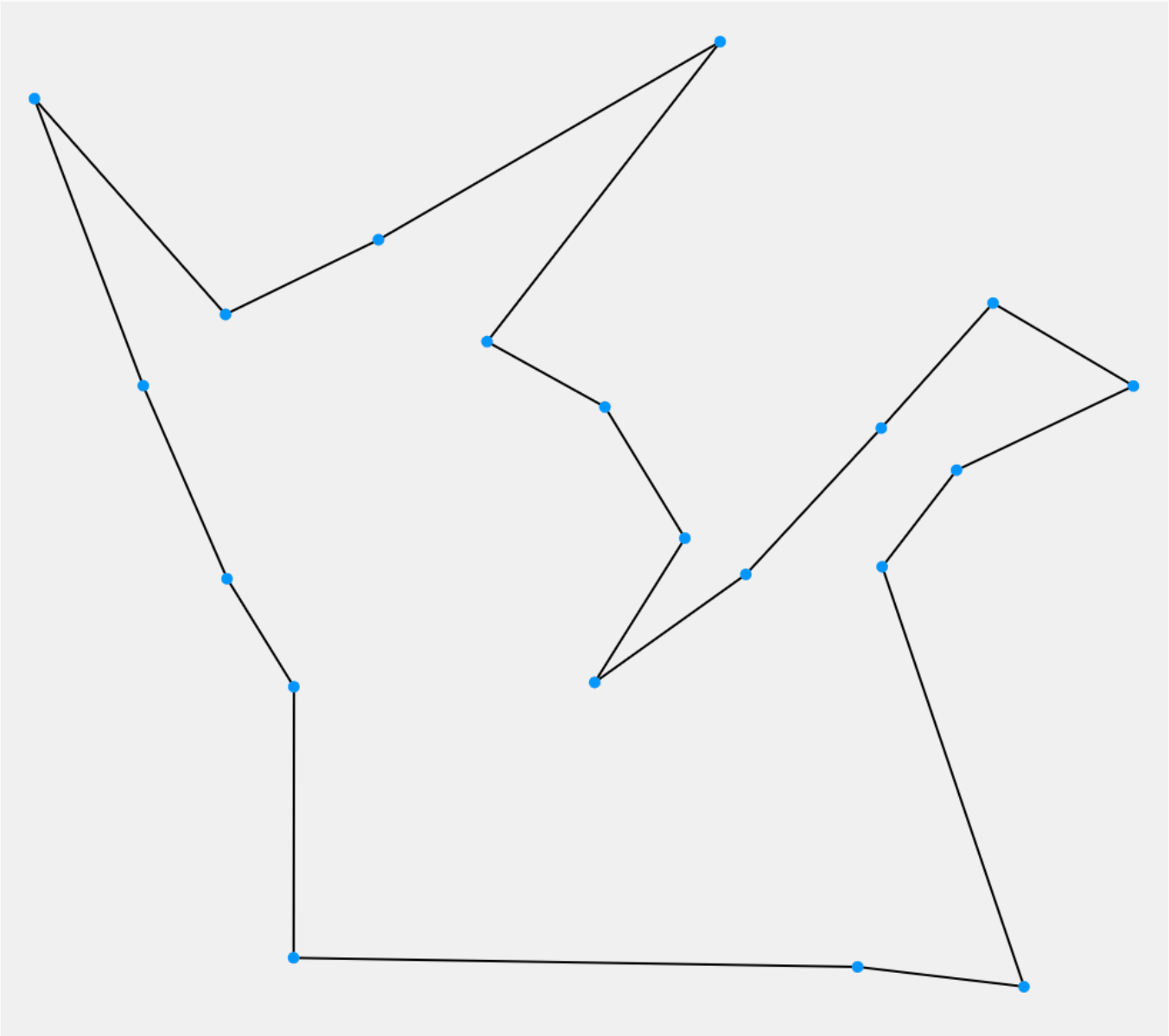
Example 2: Traveling Salesperson Problem



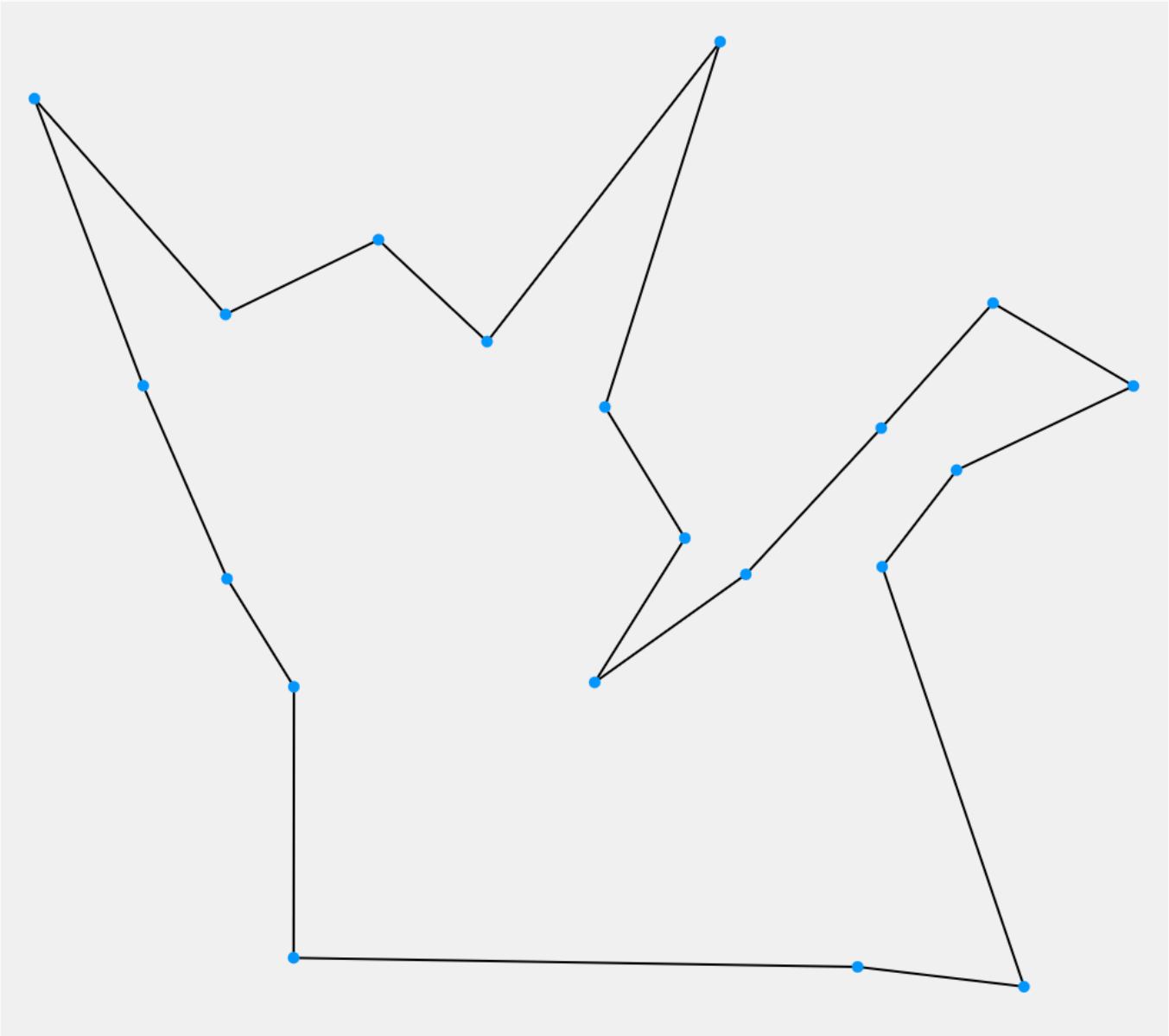
Example 2: Traveling Salesperson Problem



Example 2: Traveling Salesperson Problem



Example 2: Traveling Salesperson Problem



Example 3: Group work scheduling

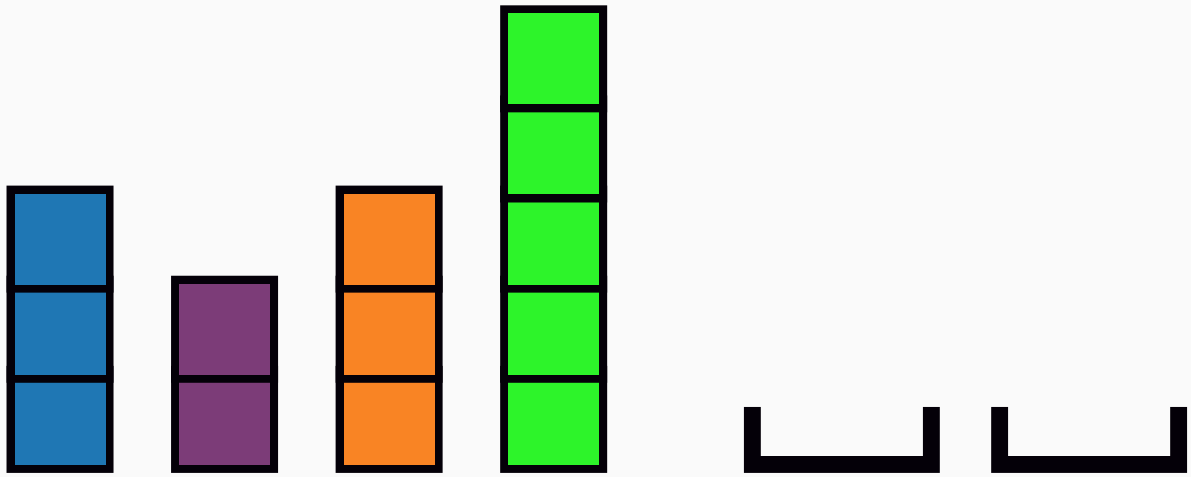
Input:

- n tasks
- k students
- task i takes t_i hours

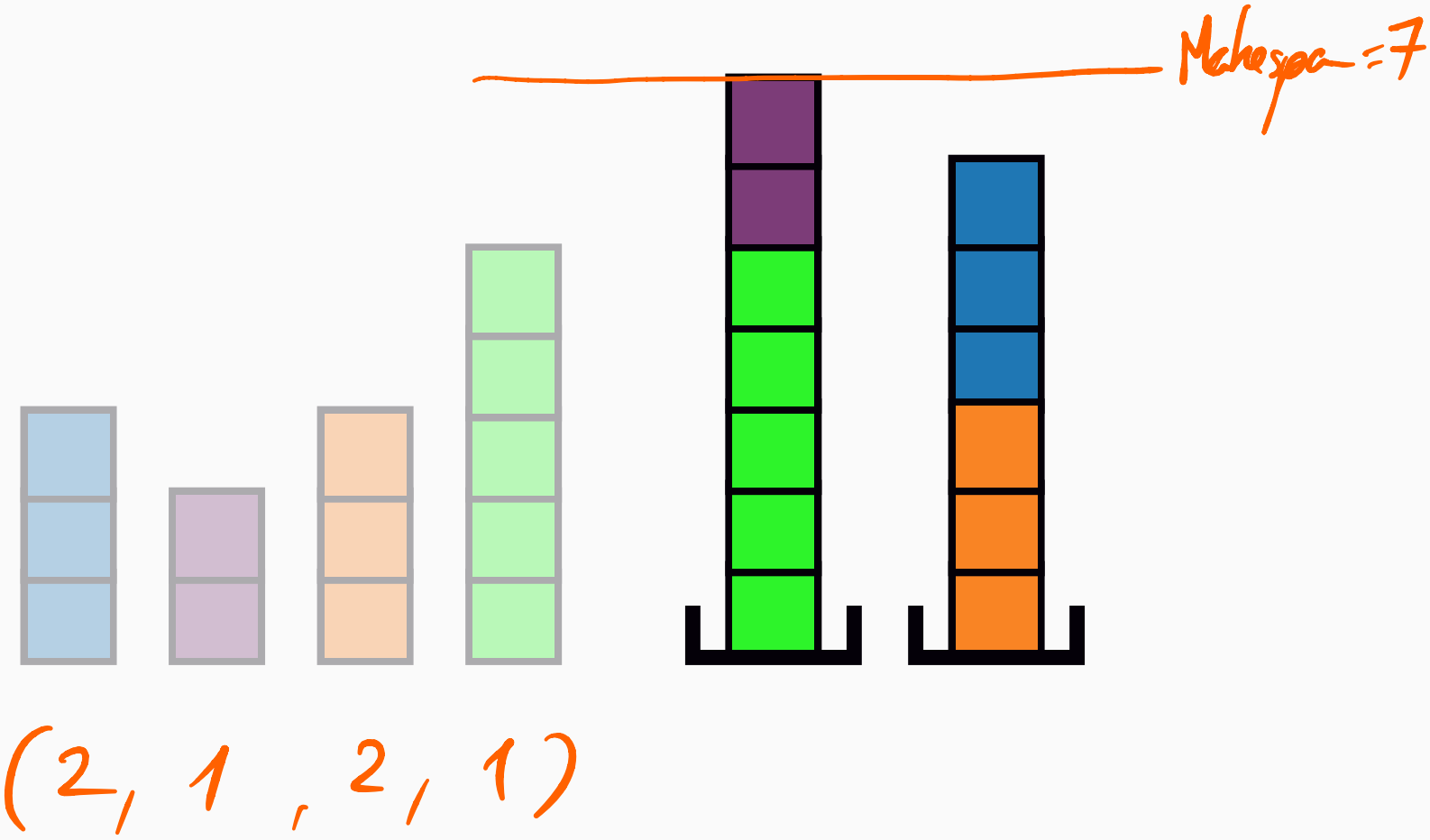
Goal: Find an assignment of tasks to students

- That completes as soon as possible
- Without splitting tasks
- Makespan: Maximum work of any student

Example 3: Group work scheduling



Example 3: Group work scheduling



Example 3: Group work scheduling

Solution:

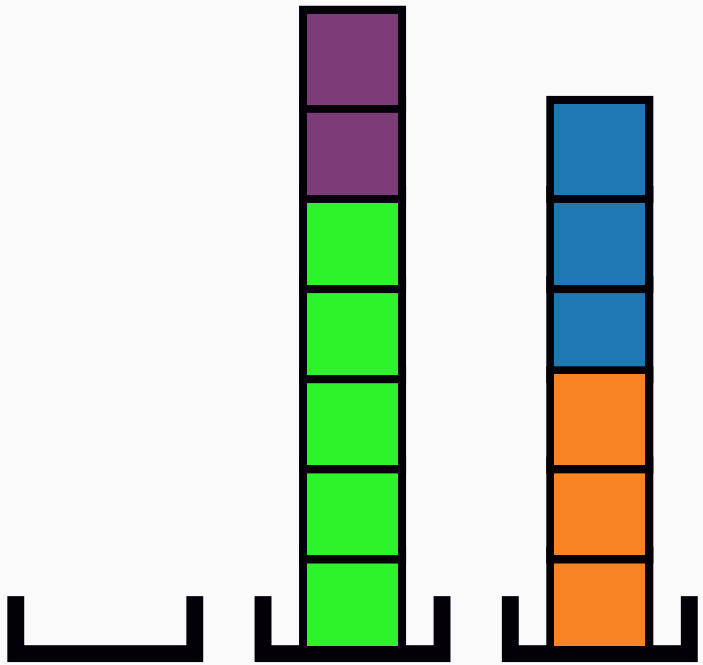
- Assignment of tasks to students
- Vector v where student v_i does task i

Neighborhood operators:

1. Move a task
2. Exchange two tasks

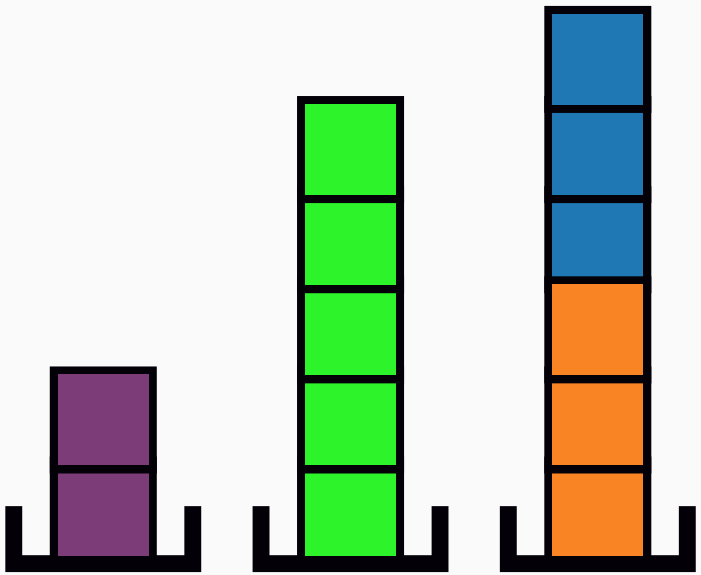
Example 3: Group work scheduling

Operator: move a task



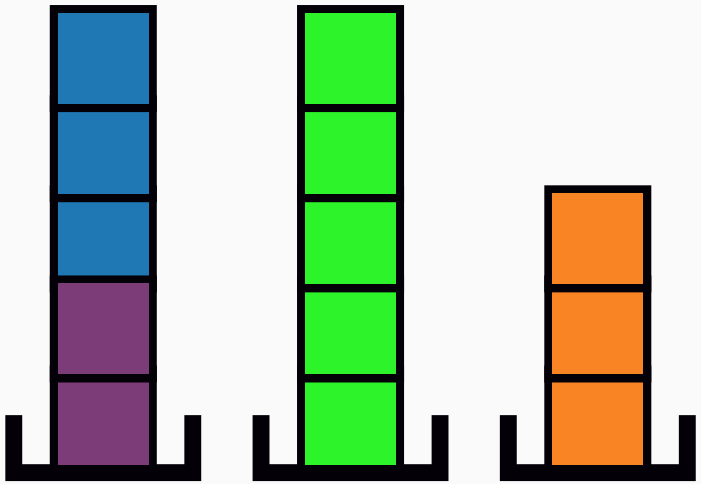
Example 3: Group work scheduling

Operator: move a task



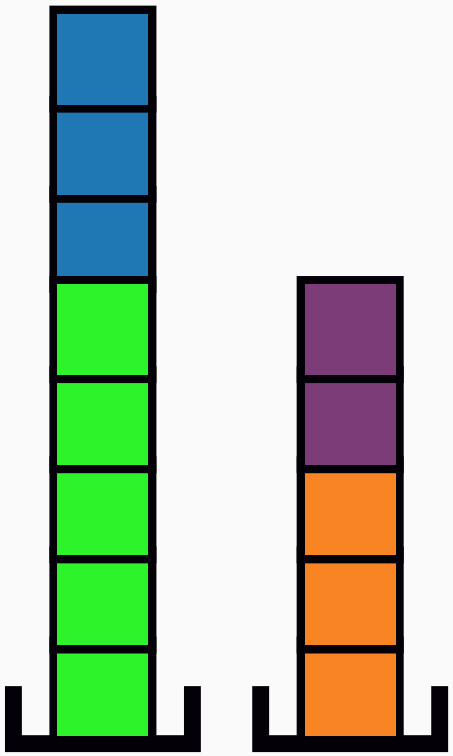
Example 3: Group work scheduling

Operator: move a task



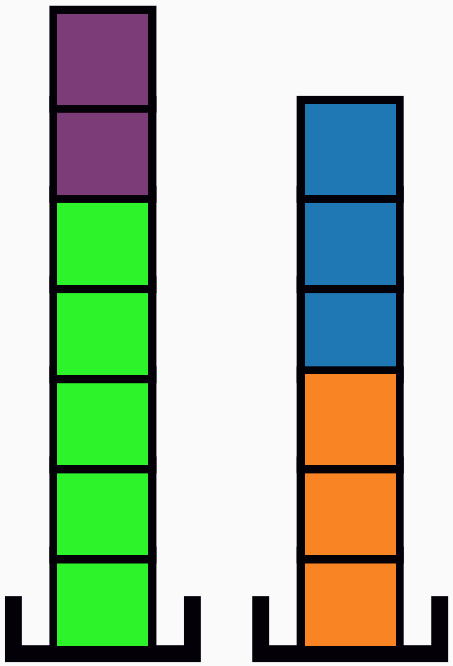
Example 3: Group work scheduling

Operator: exchange two tasks



Example 3: Group work scheduling

Operator: exchange two tasks



Example 3: Group work scheduling

Problem:

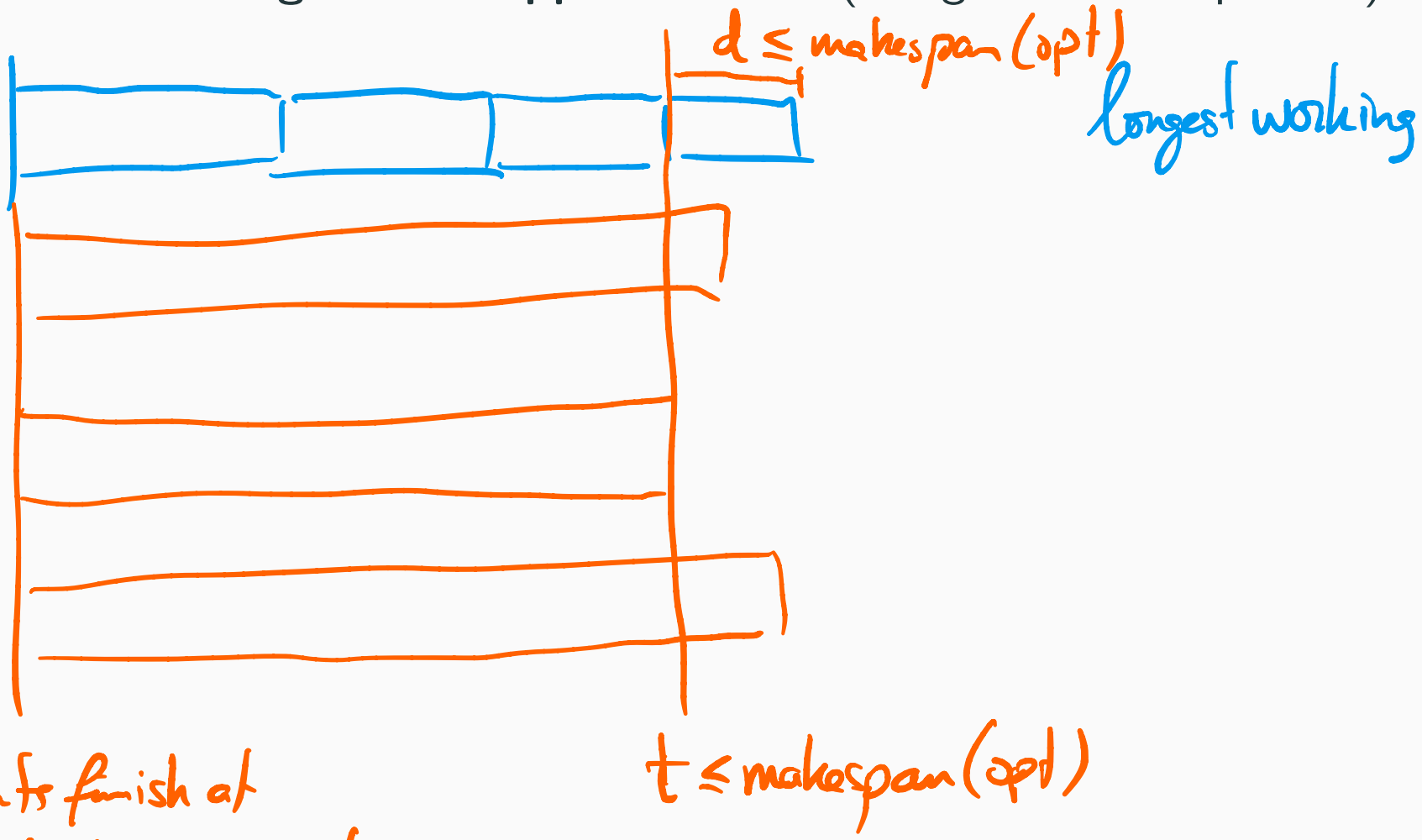
- Simplest version of a scheduling problem
- Scheduling problems are an important class

Local Search:

- Always at most $2\times$ the optimum makespan
- It is a 2-approximation algorithm

Example 3: Group work scheduling

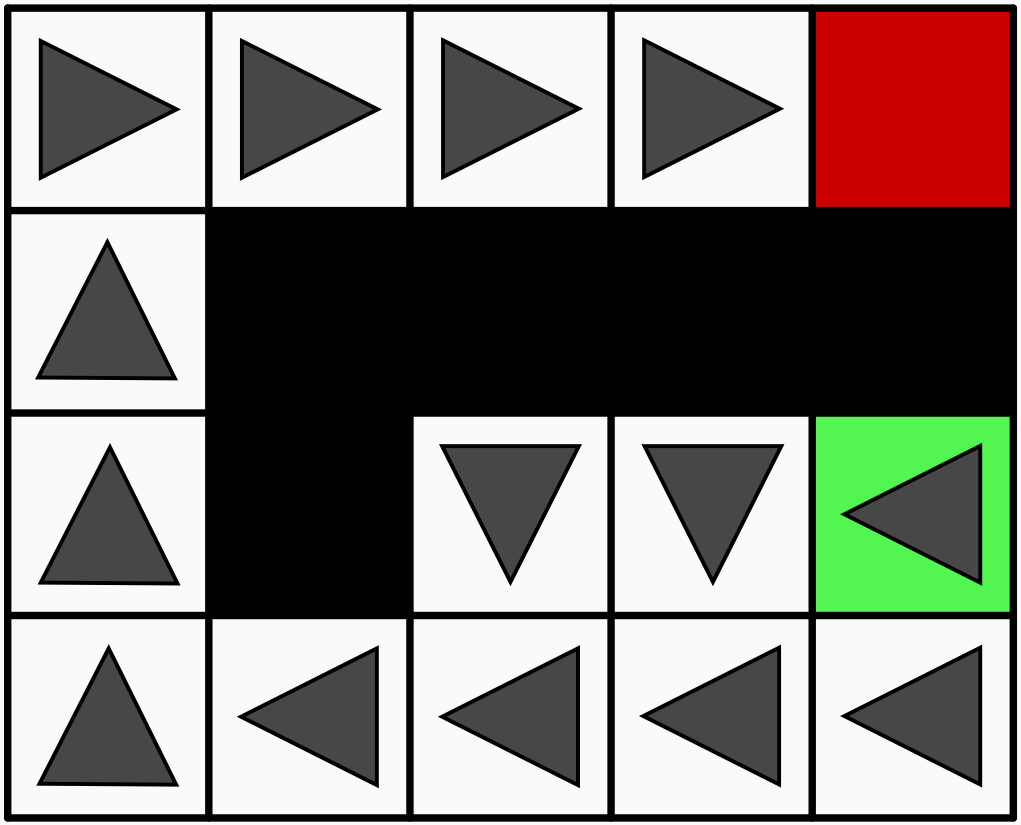
Local Search gives a 2-approximation (using the move operator)



Example 4: Maze Solving

Problem:

- Given a labyrinth, place directions on each tile
- A robot should reach the exit following the arrows



Example 4: Maze Solving

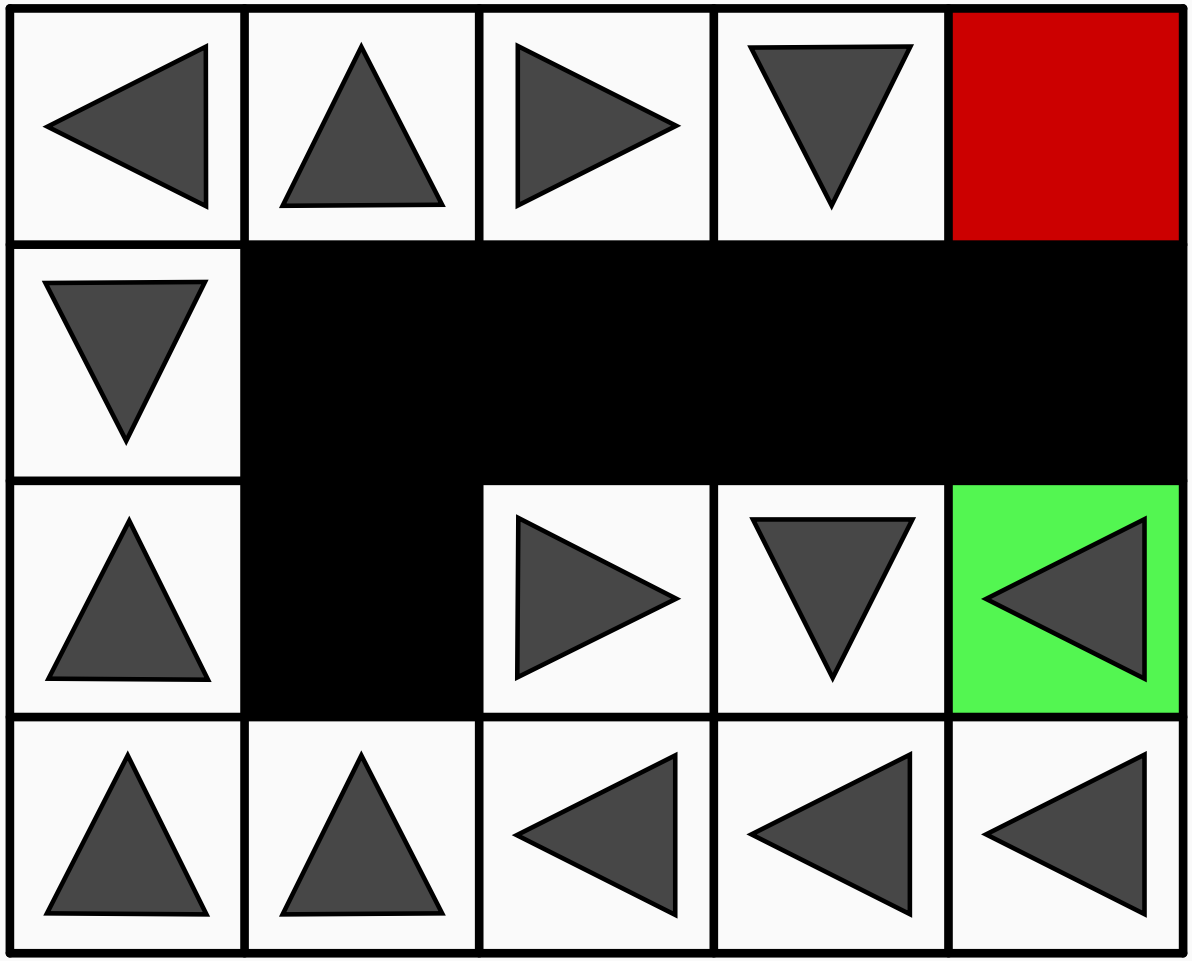
Local Search:

- Solution: One of $\{\uparrow, \rightarrow, \downarrow, \leftarrow\}$ on each cell
- Operator: Change one arrow
- Fitness: Number of steps

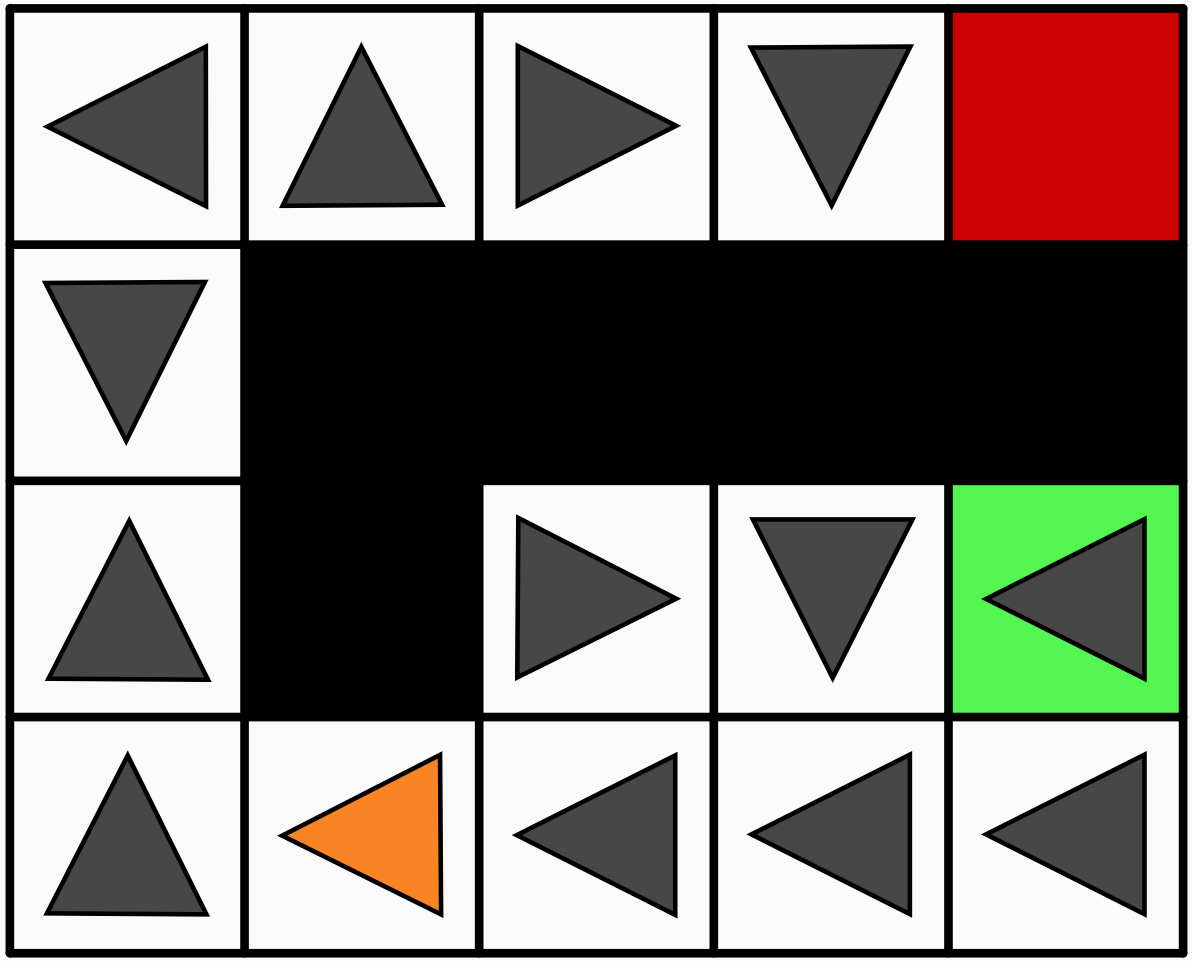
Problems:

- What if the solution is infeasible?
- And if multiple neighbors have similar fitness?

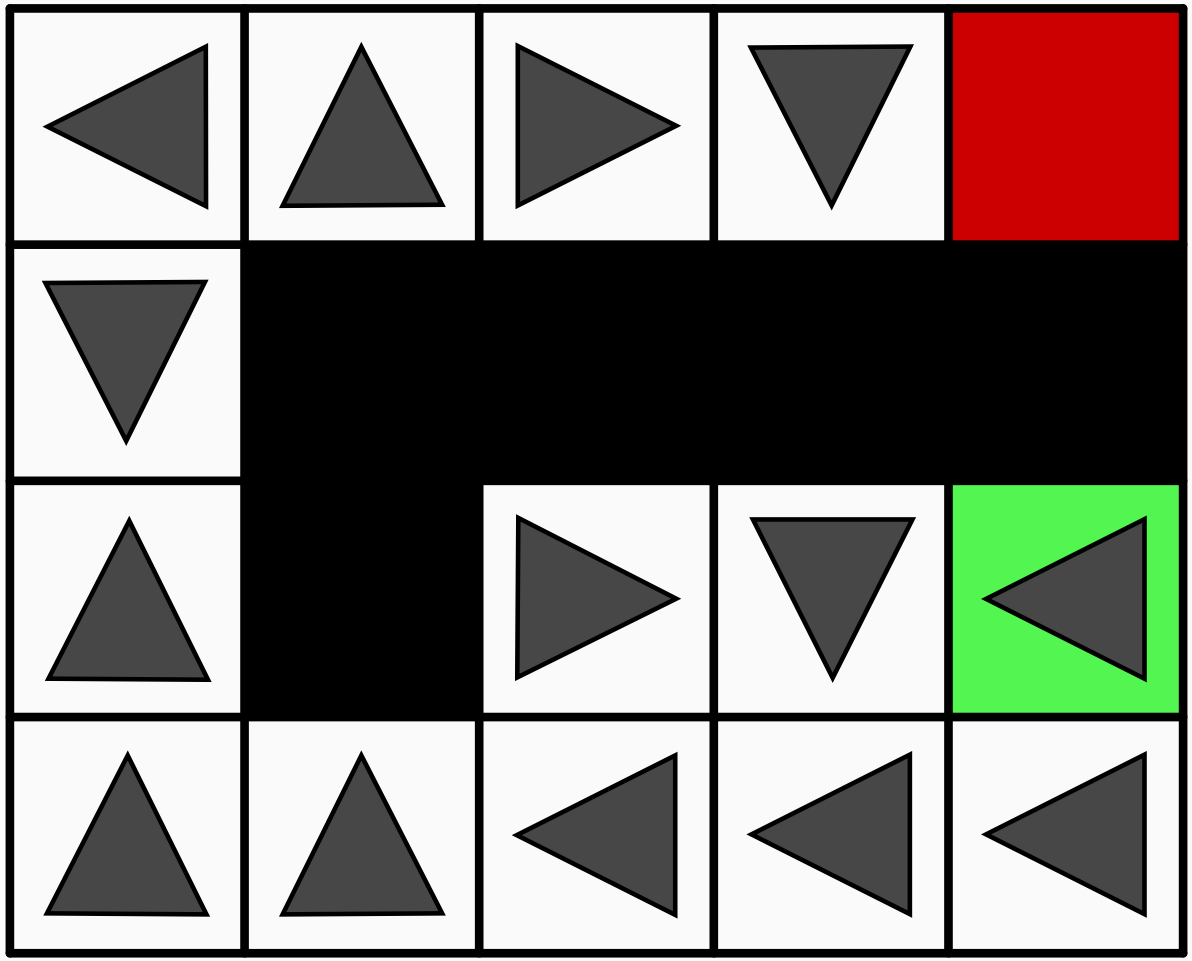
Example 4: Maze Solving



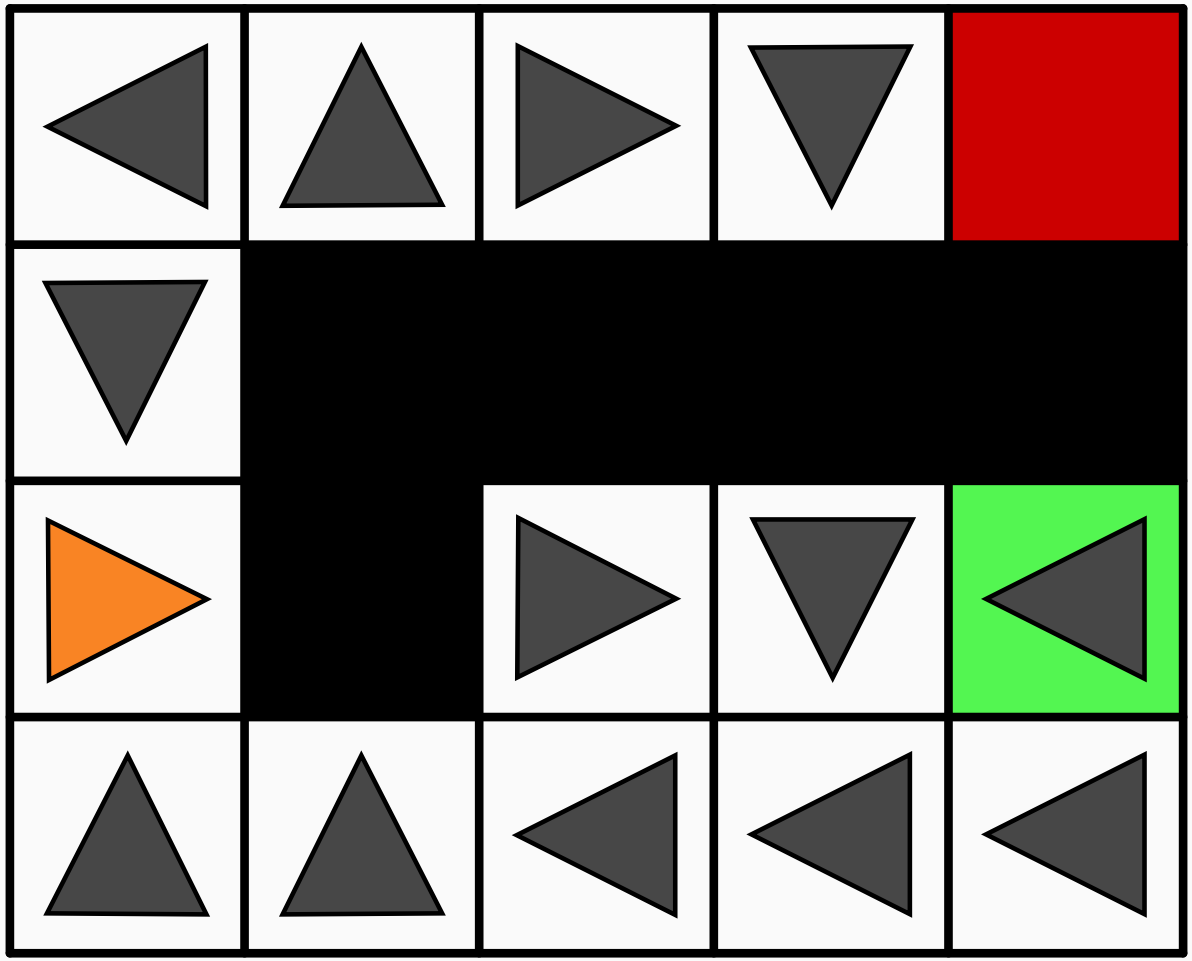
Example 4: Maze Solving



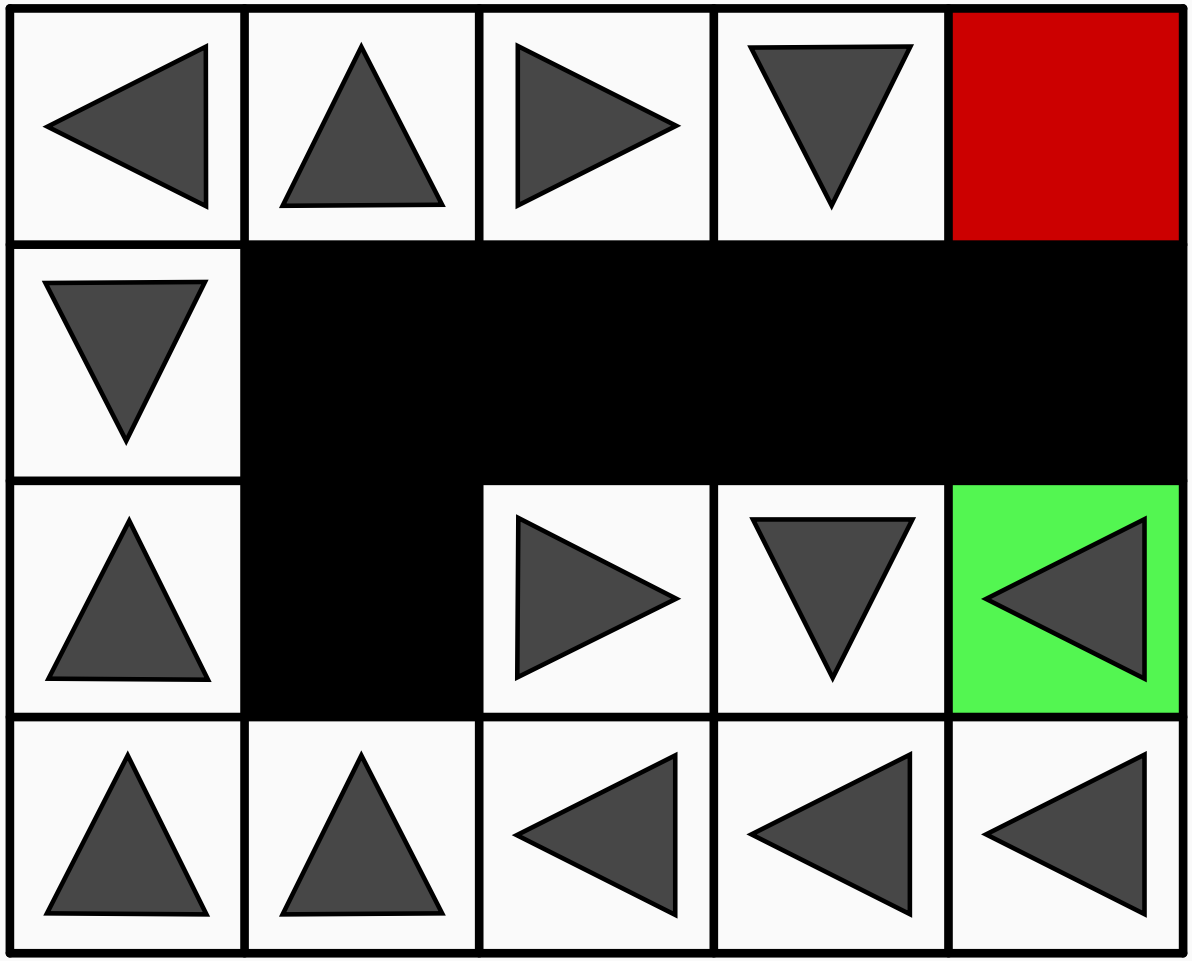
Example 4: Maze Solving



Example 4: Maze Solving



Example 4: Maze Solving



Example 4: Maze Solving

Guided search:

- Incorporate infeasibility into fitness
  - If infeasible, give higher fitness
- Specialize the operators
  - Only allow changes on tiles visited by the robot

Another option: allow fitness to increase

- Topic for later

Local Search

Improving Local Search

Variations

Choice of neighbor:

- Best possible
  - Very inefficient
- A random improving neighbor
  - Random behavior not always desired
  - Multiple tries?
  - When to stop?
- First improving move
  - Biased towards first neighbors
  - Inefficient in the worst case
  - Threshold?

Running time optimization

Can we improve the running time?

Fitness evaluation

- Repeated many times
  - Even small improvements count
- Idea: Compute difference between solutions
  - TSP: length of changed edges
  - Often $O(1)$ instead of $O(n)$

Neighbor set

- Running time is proportional to its size
- If possible, discard bad neighbors
  - If sure that there is no improvement
- Example: TSP in $\mathbb{R}^2$ $d(u,v) = \sqrt{(u_1-v_1)^2+(u_2-v_2)^2}$
  - Always uncross edges first

Local Minima

Local minima: big problem for local search

- Change the fitness
- Change the neighborhood
- Change the strategy

Strategies: increase exploration

- Random restart: once stuck, generate new random solution
- Simulated annealing: accept bad moves "sometimes"
- Tabu search: change the neighborhood dynamically

Summary

Local search:

- Repeated local improvement
- Need solution representation and neighborhood
- Focused on exploitation, little or no exploration

Greedy local search:

- Choose neighbor that minimizes fitness
- Inefficient, but very general
- Gets trapped in local minima
- Examples: minimization, TSP, scheduling, maze solving
- Variations, optimization, and improvements