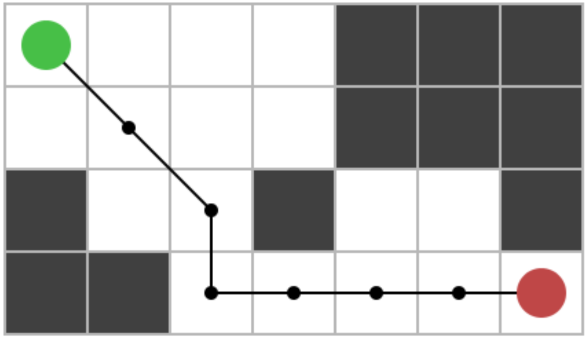
Informed Search Optimization

DV2557

Dr. Prashant Goswami
Assistant Professor, BTH (DIKR)
prashantgos.github.io

prashant.goswami@bth.se



INFORMED SEARCH

Informed Search

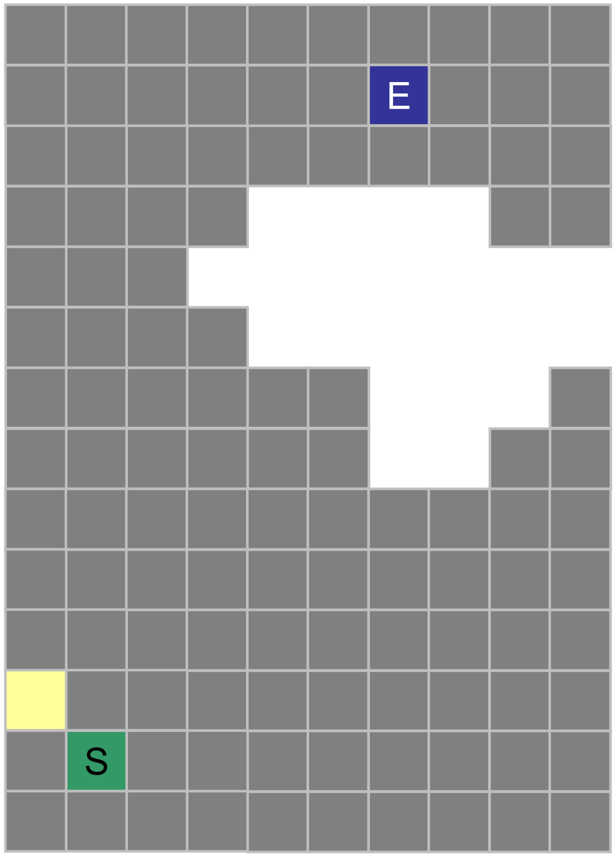
- The principle:
  - Dont just pick the first node from the open list (as in uninformed search).
  - Instead pick the most promising node
- Steer our search in the right direction.
- Sometimes called *best-first* or *guided search*.
- How do we know which node is the best?
- No exact answer, but we can make a qualified guess!
- Heuristic functions can be used to find a potentially good node.

Heuristic functions

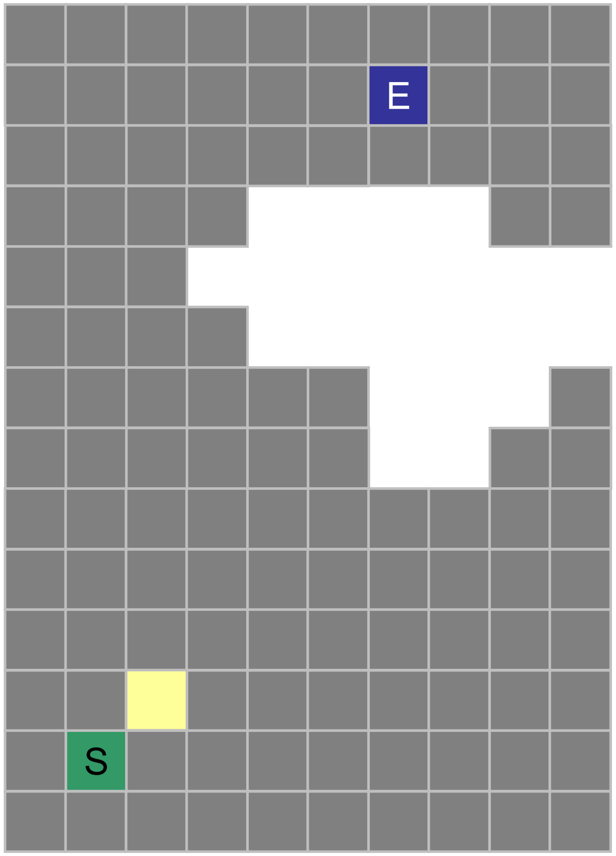
- Uninformed search (depth-first or breadth-first):
  - Pick the first node in the open list.
- Informed search:
  - Pick the most promising node in the open list.
- The most promising node is found by:
  - Use a heuristic function to calculate a promising value for each node in the open list.
  - Keep the open list sorted (implemented as a priority queue)
  - The most promising node first in the list, and evaluated first.
- A common heuristic function, Euclidean distance:
  - From current node to destination node.
  - ”Bird” path. Not perfect but good enough.

  - distance = $\sqrt{((x_{end} - x_{start})^2 + (y_{end} - y_{start})^2)}$

Informed Search

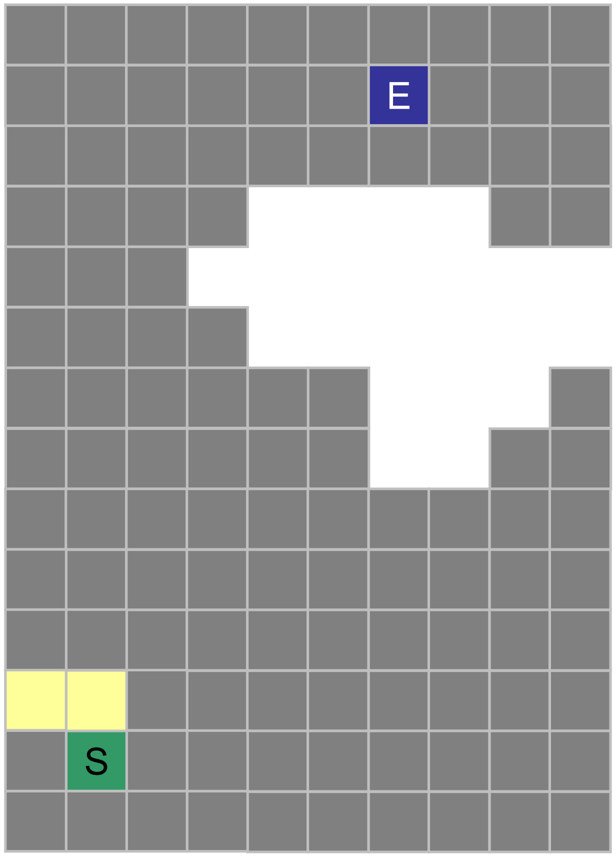


Uninformed search.

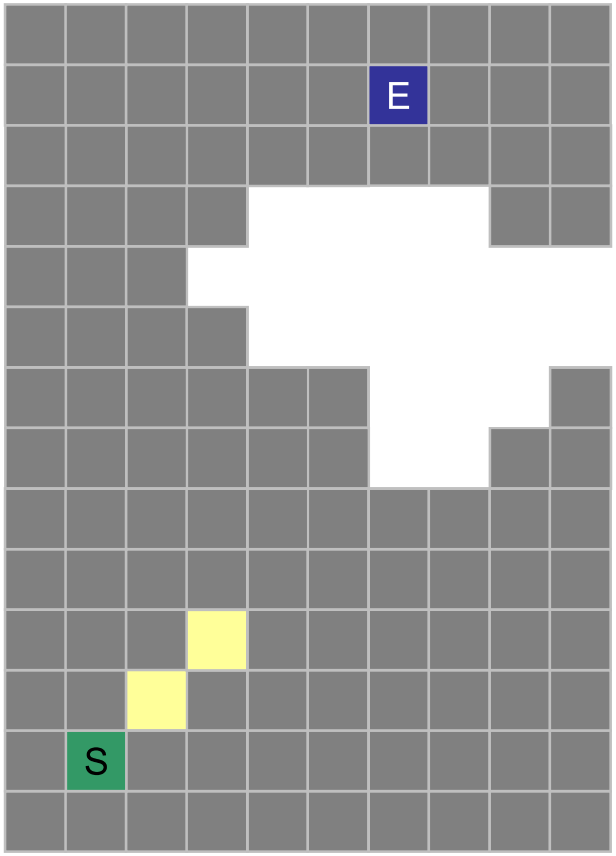


Informed search using Euclidean distance as heuristic.

Informed Search

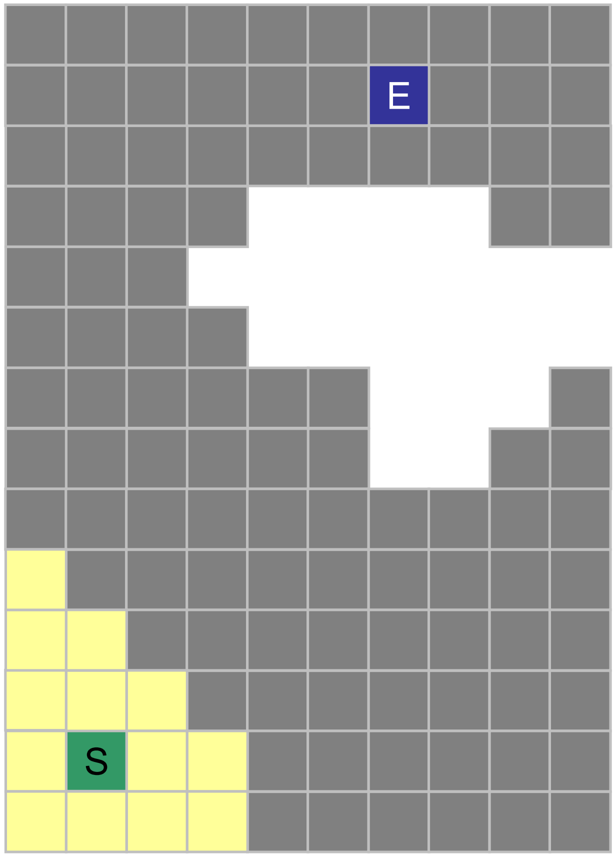


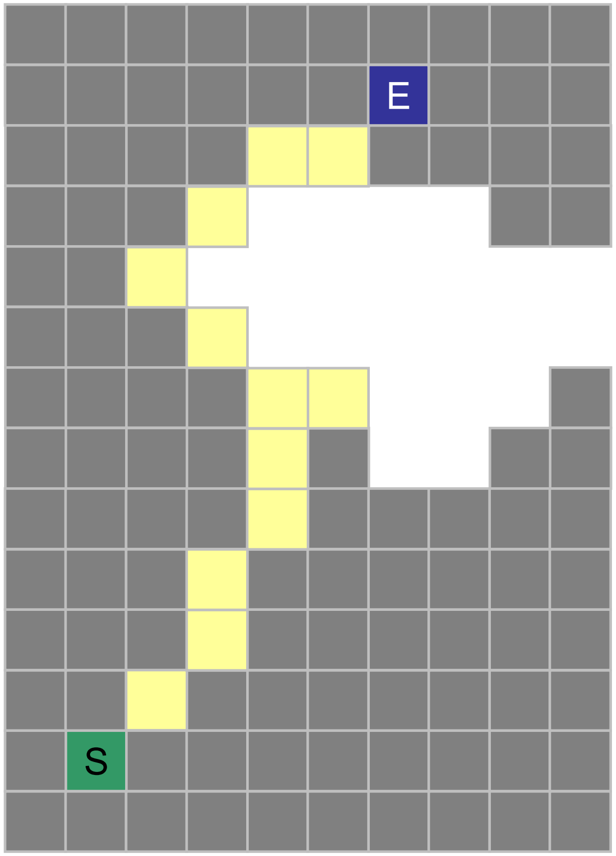
Uninformed



Informed

Informed Search



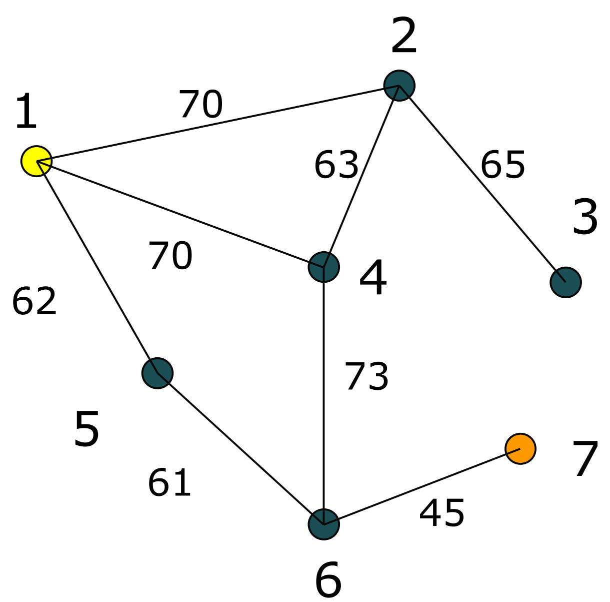


In both cases 13 nodes have been expanded!

Greedy Search

- Best node is decided from the estimated cost to goal node.
- Guaranteed to find a solution (complete).
- May find the optimal path, but it cannot be guaranteed.
- Evaluation function: $f(n) = h(n)$
    - $h(n)$ = estimated cost to goal state

Greedy search



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

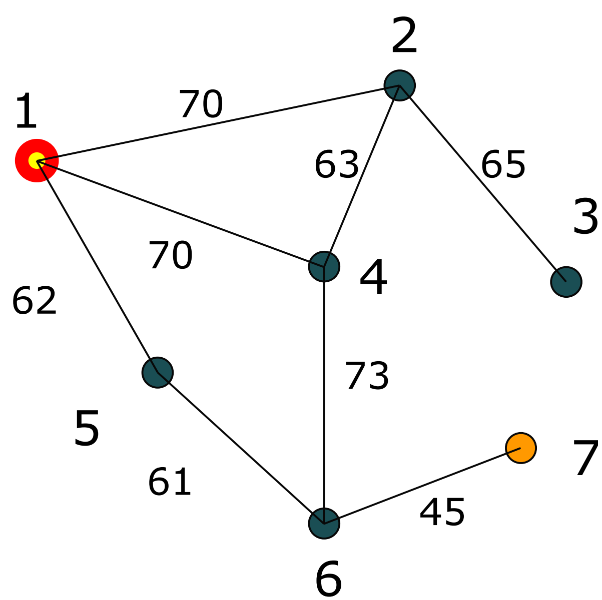
Goal:

Node 1 to 7

Open list:

Closed list:

Greedy search



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

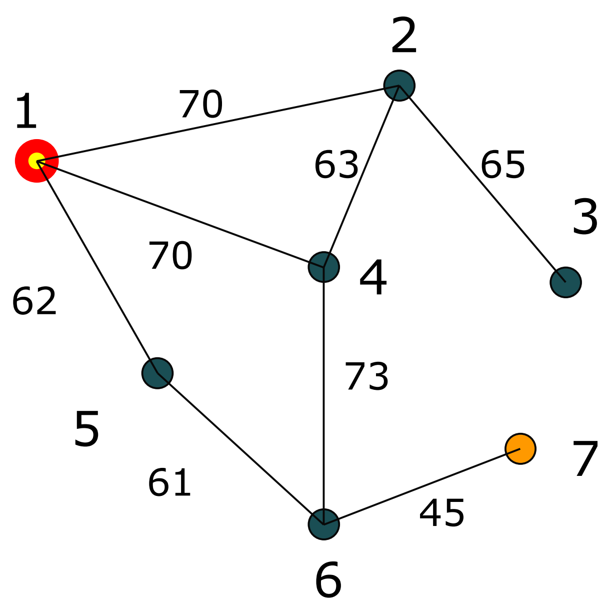
2 (110), 4 (65), 5 (68)

Closed list:

1 (120)

Sort the list by f(n) = h(n)

Greedy search



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

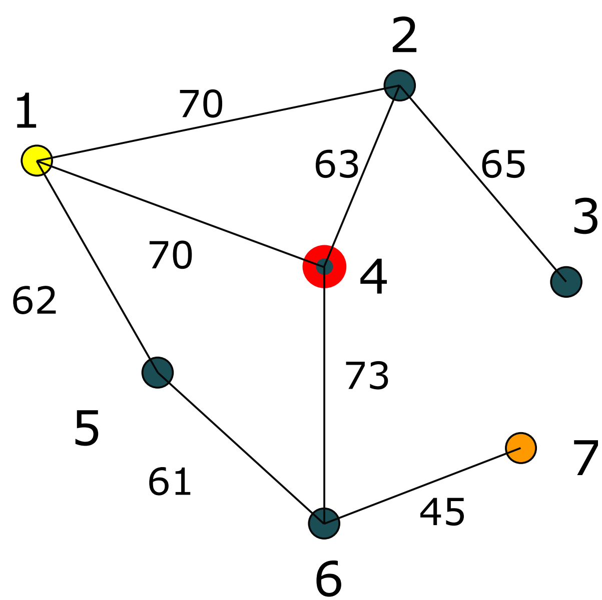
Goal:
Node 1 to 7

Open list:
4 (65), 5 (68), 2 (110)

Sorted open list!

Closed list:
1 (120)

Greedy search



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

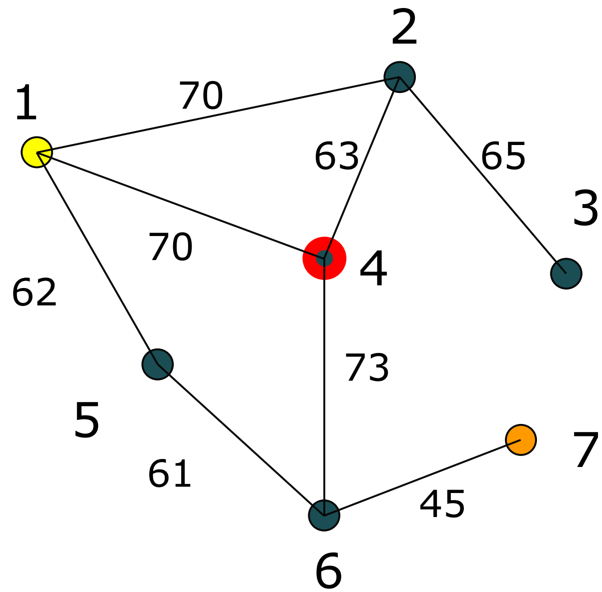
5 (68), 2 (110), 6 (45)

Sort the open list!

Closed list:

1 (120), 4 (65)

Greedy search



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

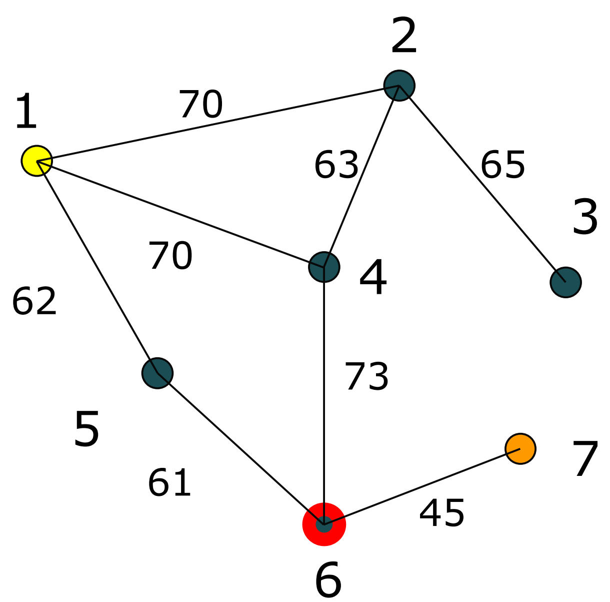
Goal:
Node 1 to 7

Open list:
6 (45), 5 (68), 2 (110)

Sorted open list!

Closed list:
1 (120), 4 (65)

Greedy search



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

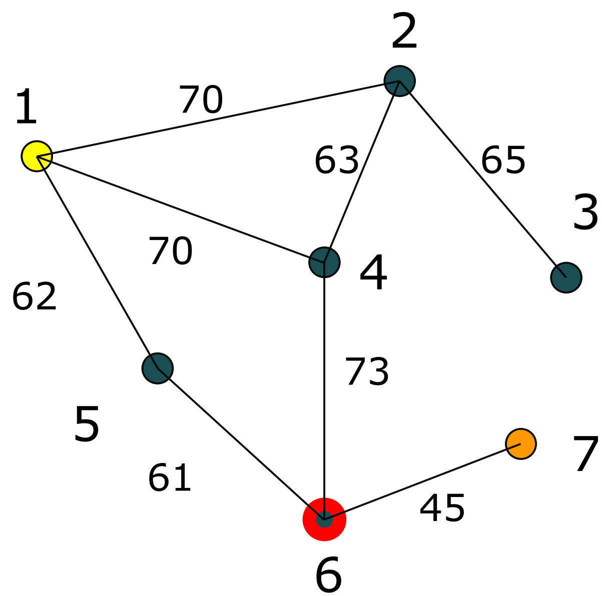
5 (68), 2 (110), 7 (0)

Sort the open list!

Closed list:

1 (120), 4 (65), 6 (45)

Greedy search



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

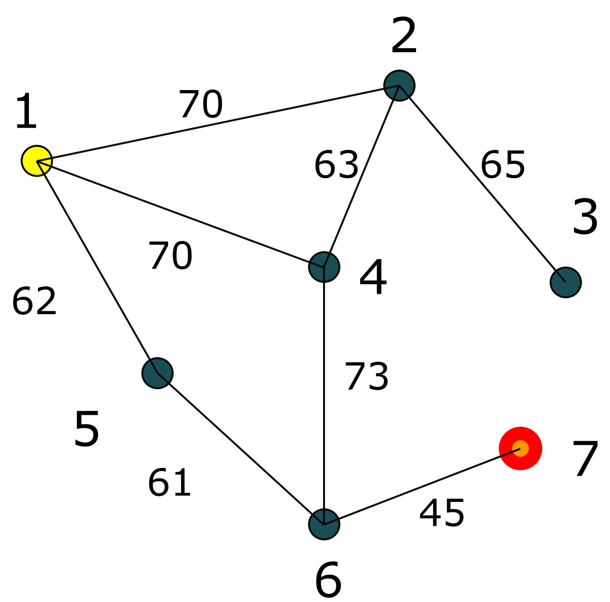
Goal:
Node 1 to 7

Open list:
7 (0), 5 (68), 2 (110)

Sorted open list!

Closed list:
1 (120), 4 (65), 6 (45)

Greedy search



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

Goal:
Node 1 to 7

Open list:
5 (68), 2 (110)

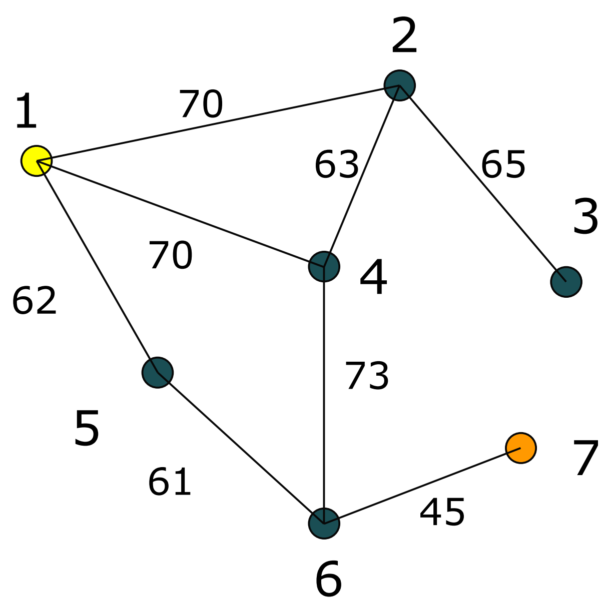
Closed list:
1 (120), 4 (65), 6 (45), 7 (0)

Distance 70 + 73 + 45 = 188 km
Is this the shortest path?

A*

- Smarter than greedy search.
- Best node *n* is decided from:
  - Estimated cost to goal node.
  - Cost of getting from start node to *n*.
    - $f(n) = h(n) + g(n)$
    - $h(n)$ = estimated cost to goal.
    - $g(n)$ = cost this far, from the start node to *n*.
- Each node *n* must hold its own *g(n)*, i.e. it must hold the shortest path from the start node to node *n*!
    - Memory consuming. In worst case it must hold an exponential number of nodes in the memory...

A*



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

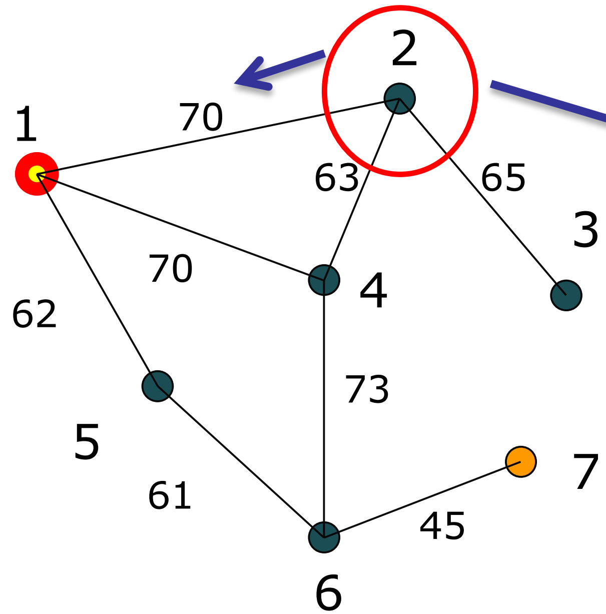
Goal:

Node 1 to 7

Open list:

Closed list:

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

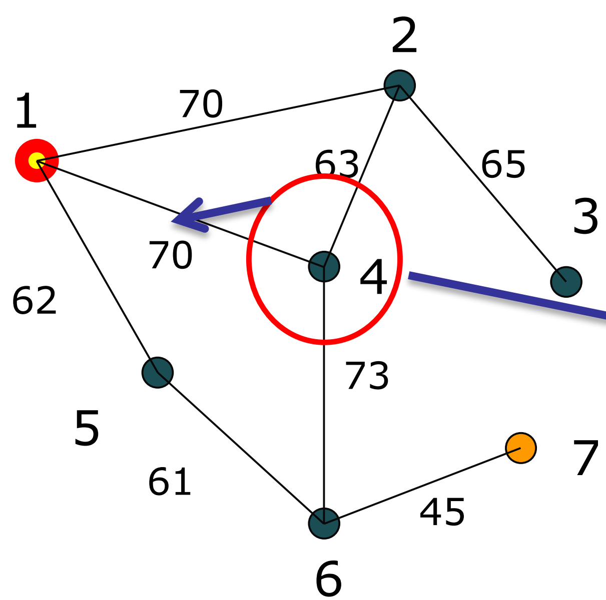
Goal:
Node 1 to 7

Open list:
2 (110+70), 4 (), 5 ()

$f(2) = h(2) + g(2) = 110 + 70$

Closed list:
1 (120+0)

A*



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

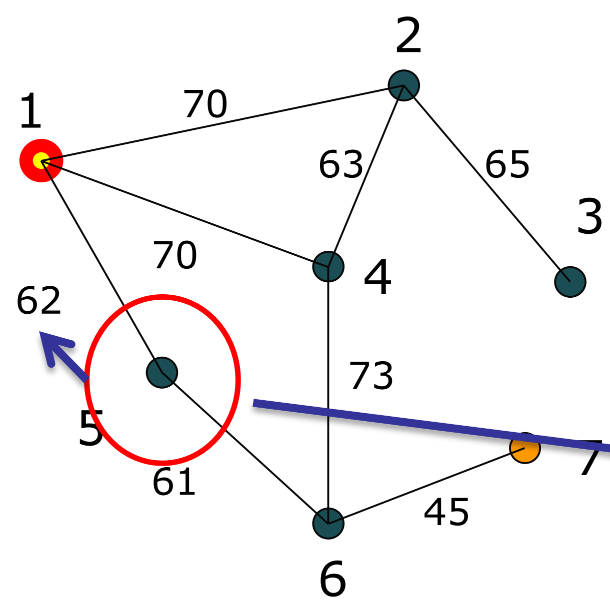
2 (110+70), 4 (65+70), 5 ()

Closed list:

1 (120+0)

f(4) = h(4) + g(4) = 65 + 70

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

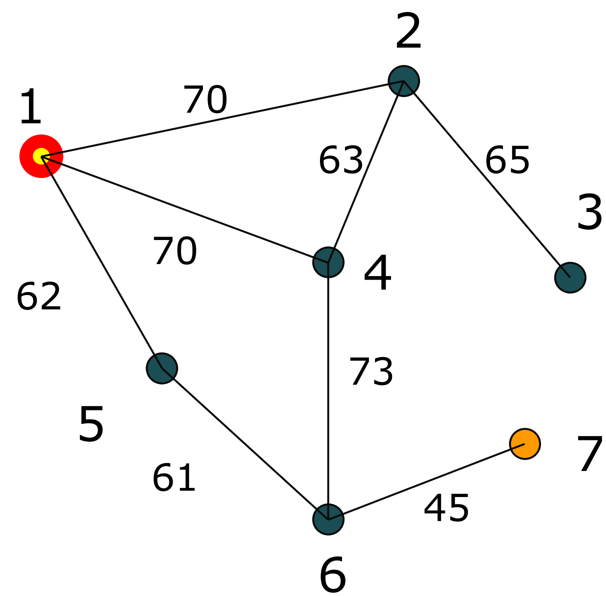
Goal:
Node 1 to 7

Open list:
2 (110+70), 4 (65+70), 5 (68+62)

Closed list:
1 (120+0)

$f(5) = h(5) + g(5) = 68 + 62$

A*



<u>Heuristic h(n)</u>

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

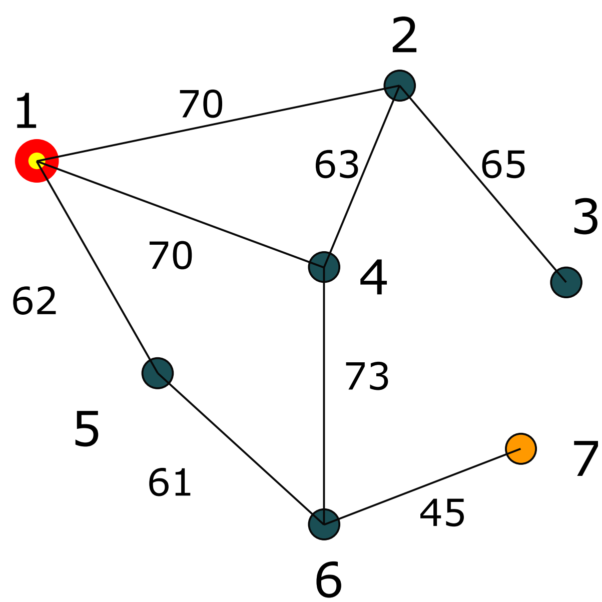
Goal:
Node 1 to 7

Open list:
2 (110+70), 4 (65+70), 5 (68+62)

Sort the open list!

Closed list:
1 (120+0)

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

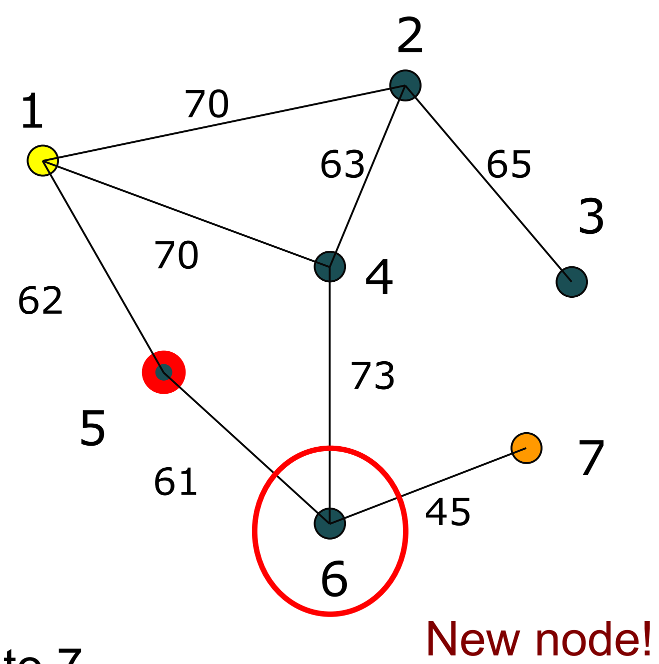
Goal:
Node 1 to 7

Open list:
5 (68+62), 4 (65+70), 2 (110+70)

Sorted open list!

Closed list:
1 (120+0)

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

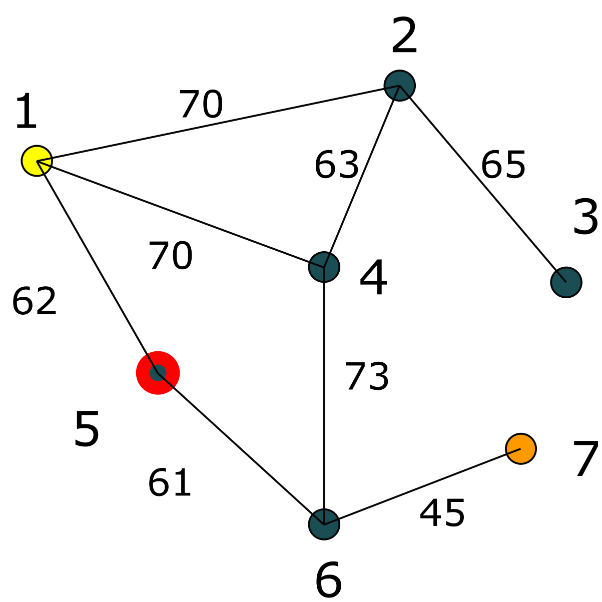
Goal:
Node 1 to 7

Open list:
4 (65+70), 2 (110+70), 6 (45+123)

Closed list:
1 (120+0) 5 (68+62)

$f(6) = h(6) + g(6) = 45 + 61 + 62$
$= 45 + 123$

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

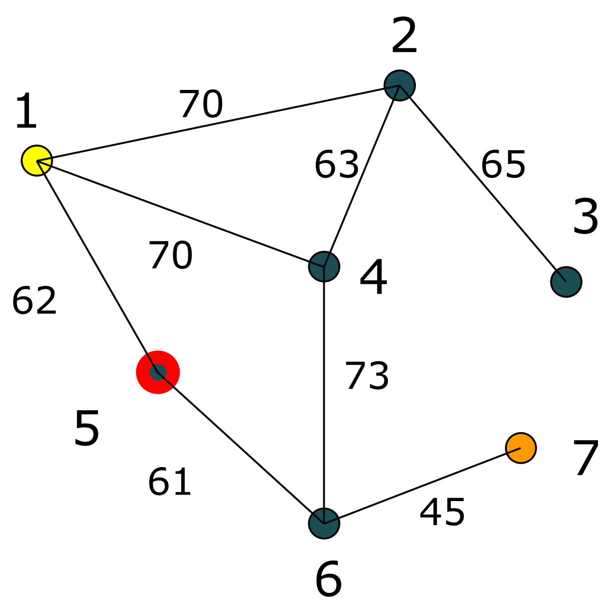
Goal:
Node 1 to 7

Open list:
4 (65+70), 2 (110+70), 6 (45+123)

Sort the open list!

Closed list:
1 (120+0) 5 (68+62)

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

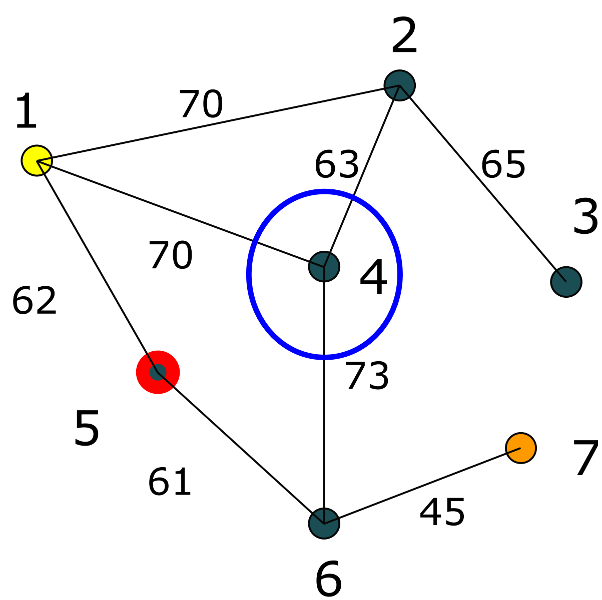
Goal:
Node 1 to 7

Open list:
4 (65+70), 6 (45+123), 2 (110+70)

Sorted open list!

Closed list:
1 (120+0) 5 (68+62)

A*



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

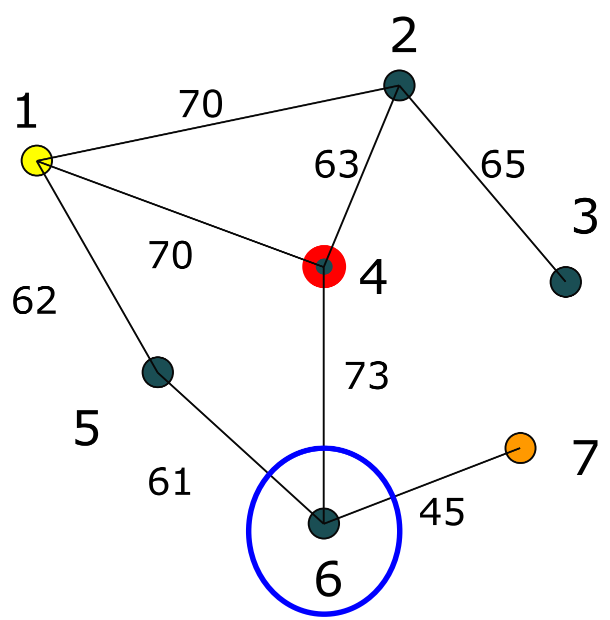
4 (65+70), 6 (45+123), 2 (110+70)

Closed list:

1 (120+0) 5 (68+62)

Node 4 is next to visit!

A*



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

6 (45+123), 2 (110+70)

Closed list:

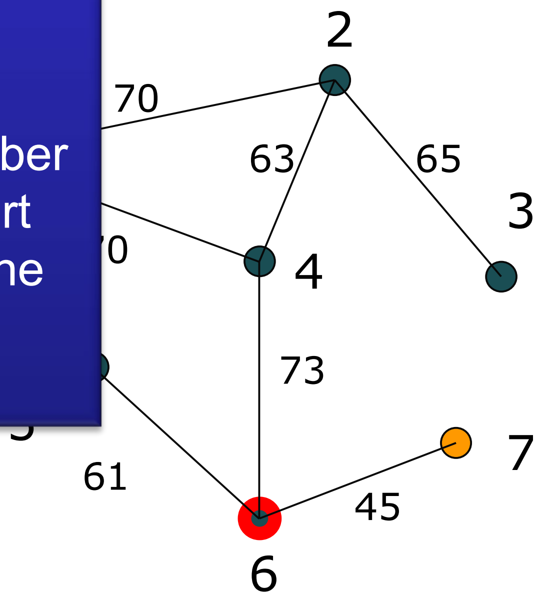
1 (120+0) 5 (68+62), 4 (65+70)

Open list already sorted!
Node 6 is next.

A*

Now we have found two possible paths to node 6: 1-4-6 and 1-5-6

Node 6 has to remember the best path from start node to node 6, and the total cost of it g(6):
1-5-6   123km



Heuristic h(n)

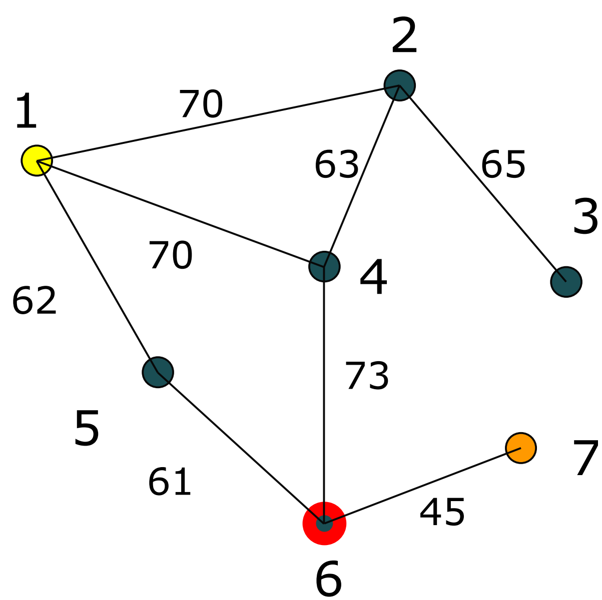
1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

Goal:
Node 1 to 7

Open list:
2 (110+70), 7 (0+168)

Closed list:
1 (120+0) 5 (68+62), 4 (65+70), 6 (45+123)

A*



Heuristic h(n)

1 120km

2 110km

3 40km

4 65km

5 68km

6 45km

Goal:

Node 1 to 7

Open list:

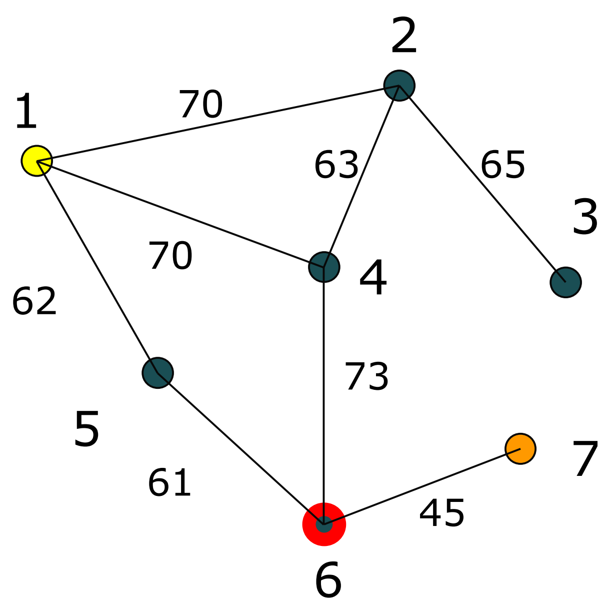
2 (110+70), 7 (0+168)

Sort the open list!

Closed list:

1 (120+0) 5 (68+62), 4 (65+70), 6 (45+123)

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

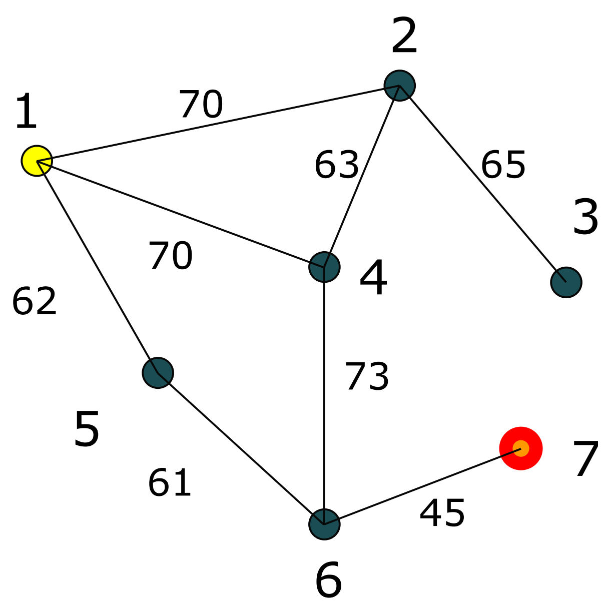
Goal:
Node 1 to 7

Open list:
7 (0+168), 2 (110+70)

Sorted open list!

Closed list:
1 (120+0) 5 (68+62), 4 (65+70), 6 (45+123)

A*



Heuristic h(n)

1 120km
2 110km
3 40km
4 65km
5 68km
6 45km

Goal:
Node 1 to 7

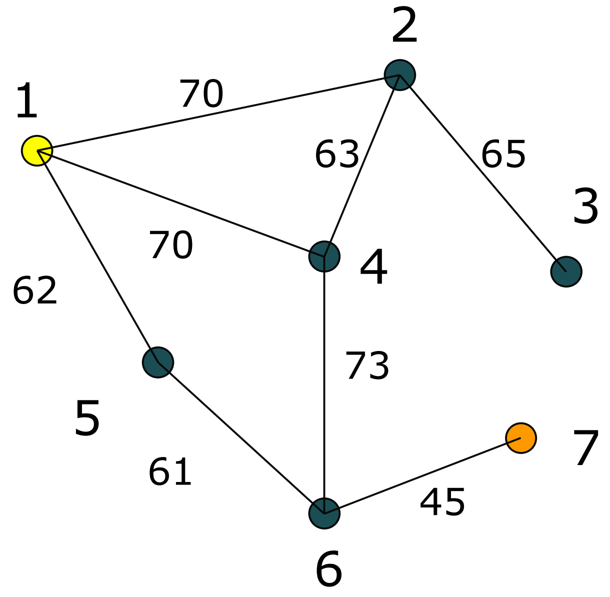
Open list:
2 (110+70)

Distance 62 + 61 + 45 = 168 km
We have found the best path!

Closed list:
1 (120+0) 5 (68+62), 4 (65+70), 6 (45+123), 7 (0+168)

Why did Greedy Search fail?

Imperfect heuristic



Heuristic h(n)

1	120km	168	-29%
2	110km	181	-39%
3	40km	246	-83%
4	65km	118	-45%
5	68km	106	-36%
6	45km	45	0%

A* is…

- Optimal and complete if:
  - The heuristic function never overestimates the cost of getting to the goal node.
  - If it never overestimates the cost of getting from one node to the next.
- What about overestimation?
  - Cannot guarantee optimality.
  - A small controlled overestimation can actually speed up A* search.
  - Often used where speed is more important than optimality, like in real-time games.
- Efficient!
  - No other algorithm is known to expand less nodes for a given heuristic function.

OPTIMIZATION

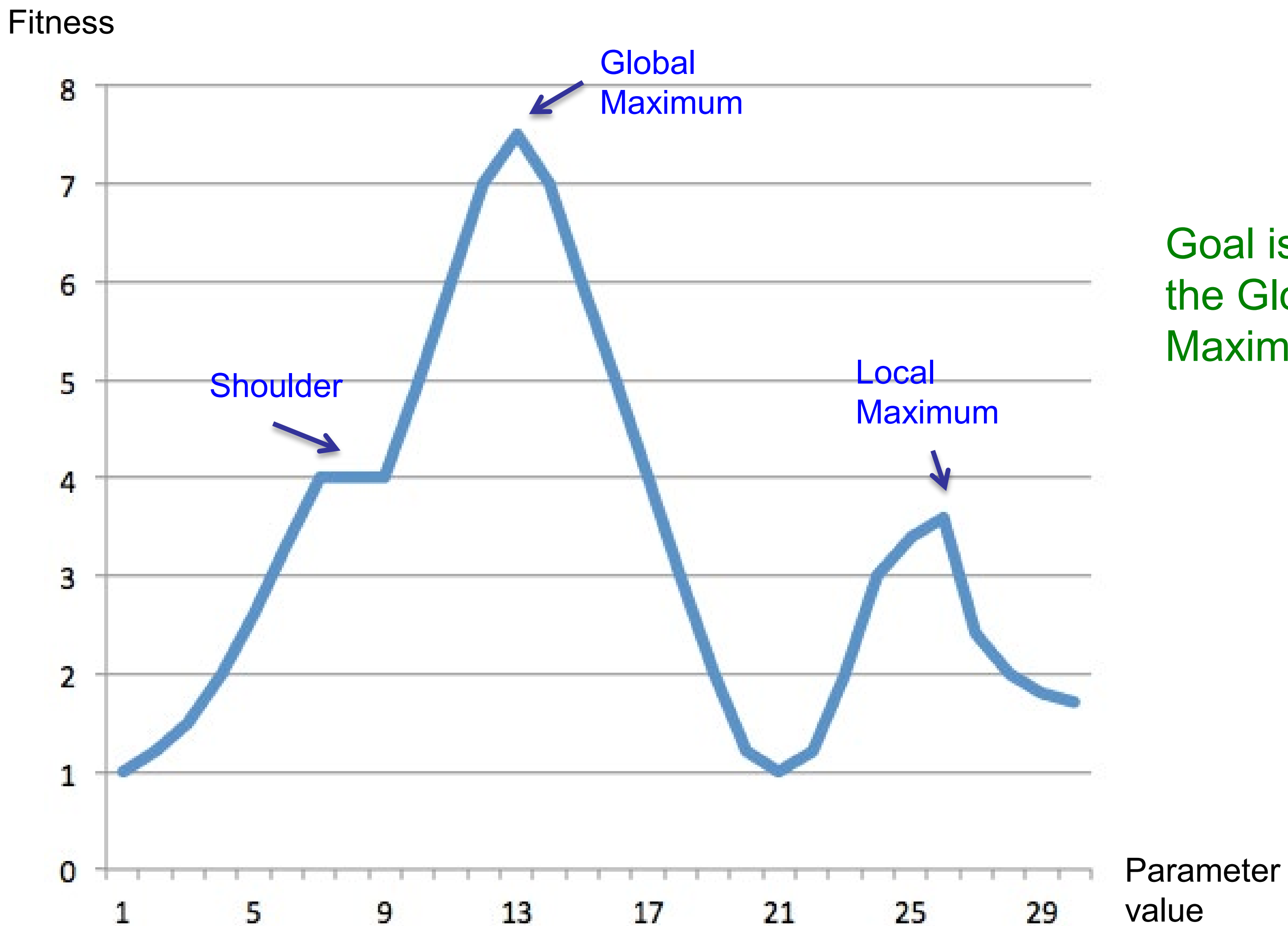
Optimization

- A common problem is to find the optimal value to parameter(s).
- We can do this by brute-force search, but that is often too time consuming.
- A better approach is optimization algorithms:
  - Hill Climbing
  - Genetic Algorithms

Optimization

- For any optimization to work we need a *fitness function*.
- It gives a numerical value to how well a parameter setting works.
- It could be number of wins for a game, total computation time for an algorithm, percentage correctly classified for a machine learning algorithm, …

Optimization

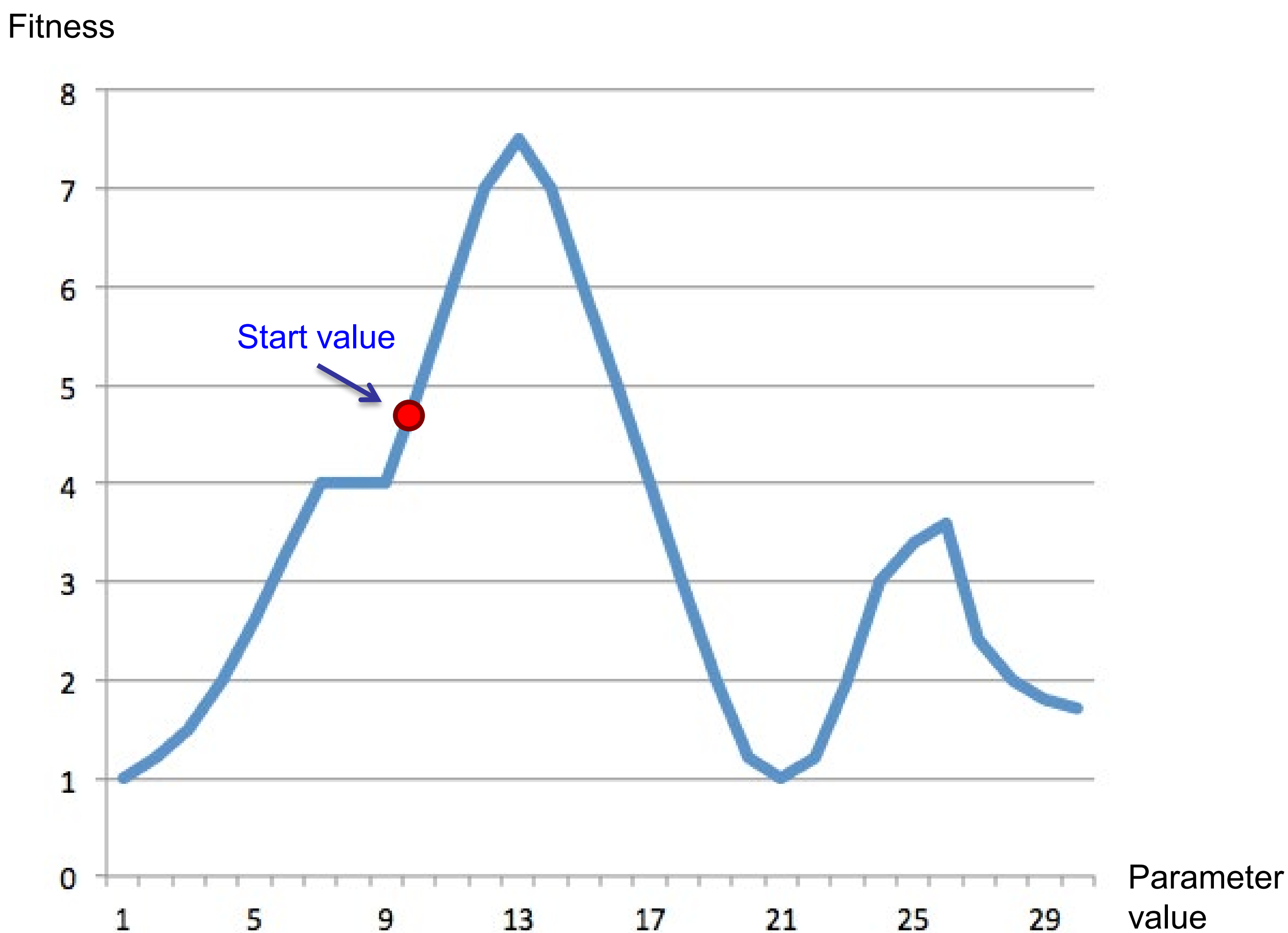


Goal is to find the Global Maximum!

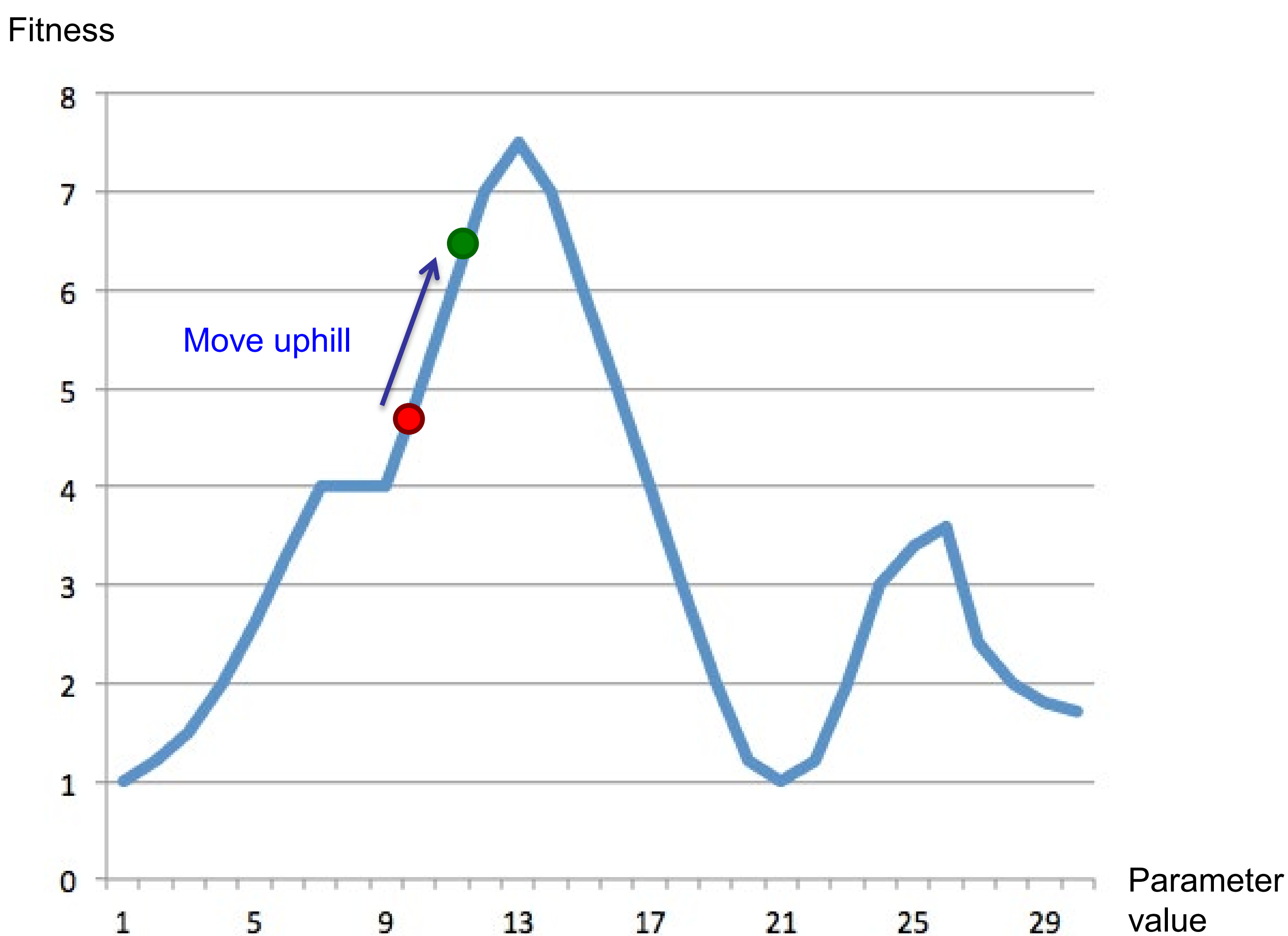
Hill Climbing

- Randomly select a start value between min and max values for a parameter.
- Find fitness for the start value and its neighbors.
- Move in the direction of increasing fitness values – uphill.
- Stop when a peak is reached.

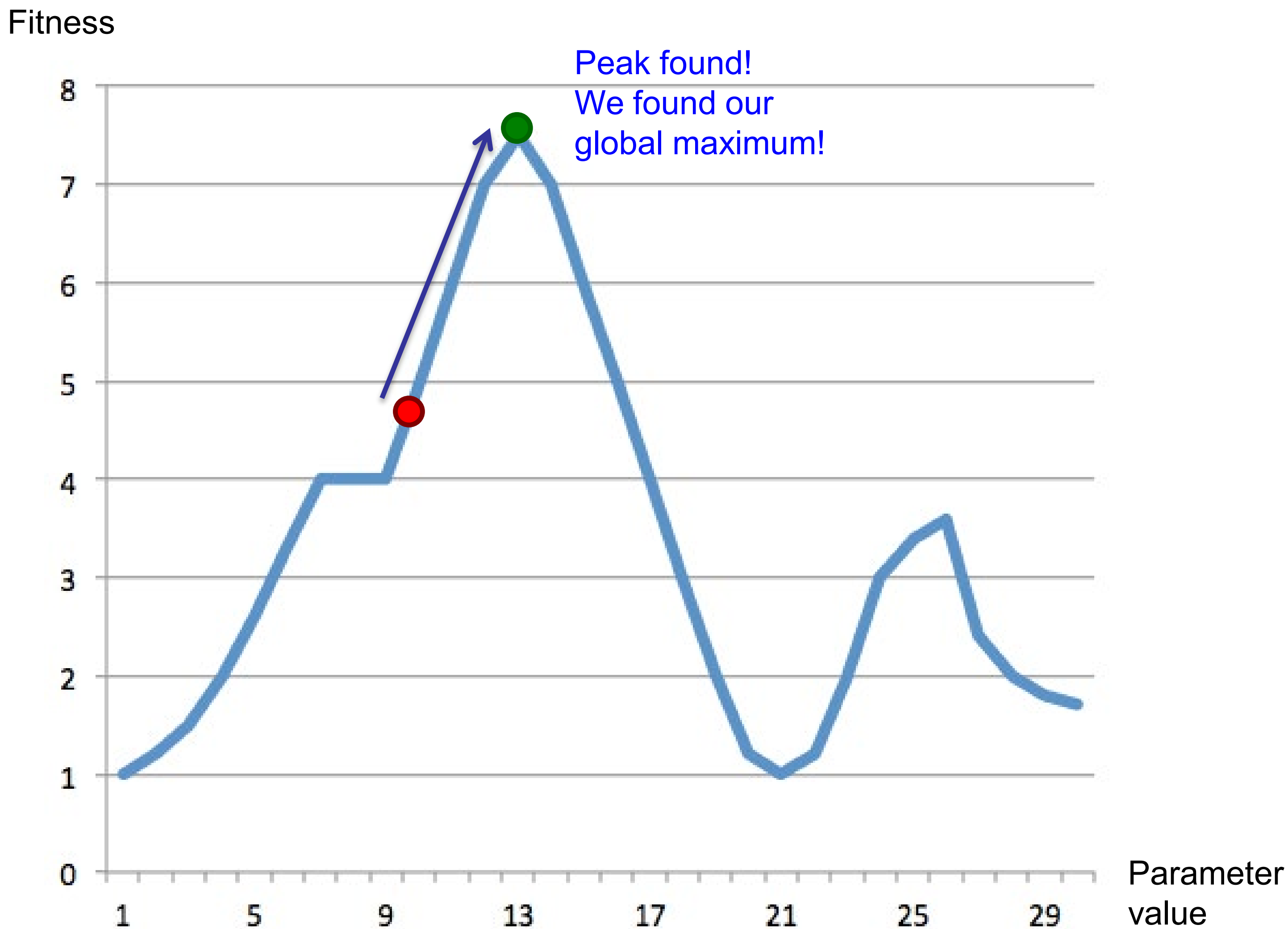
Hill Climbing



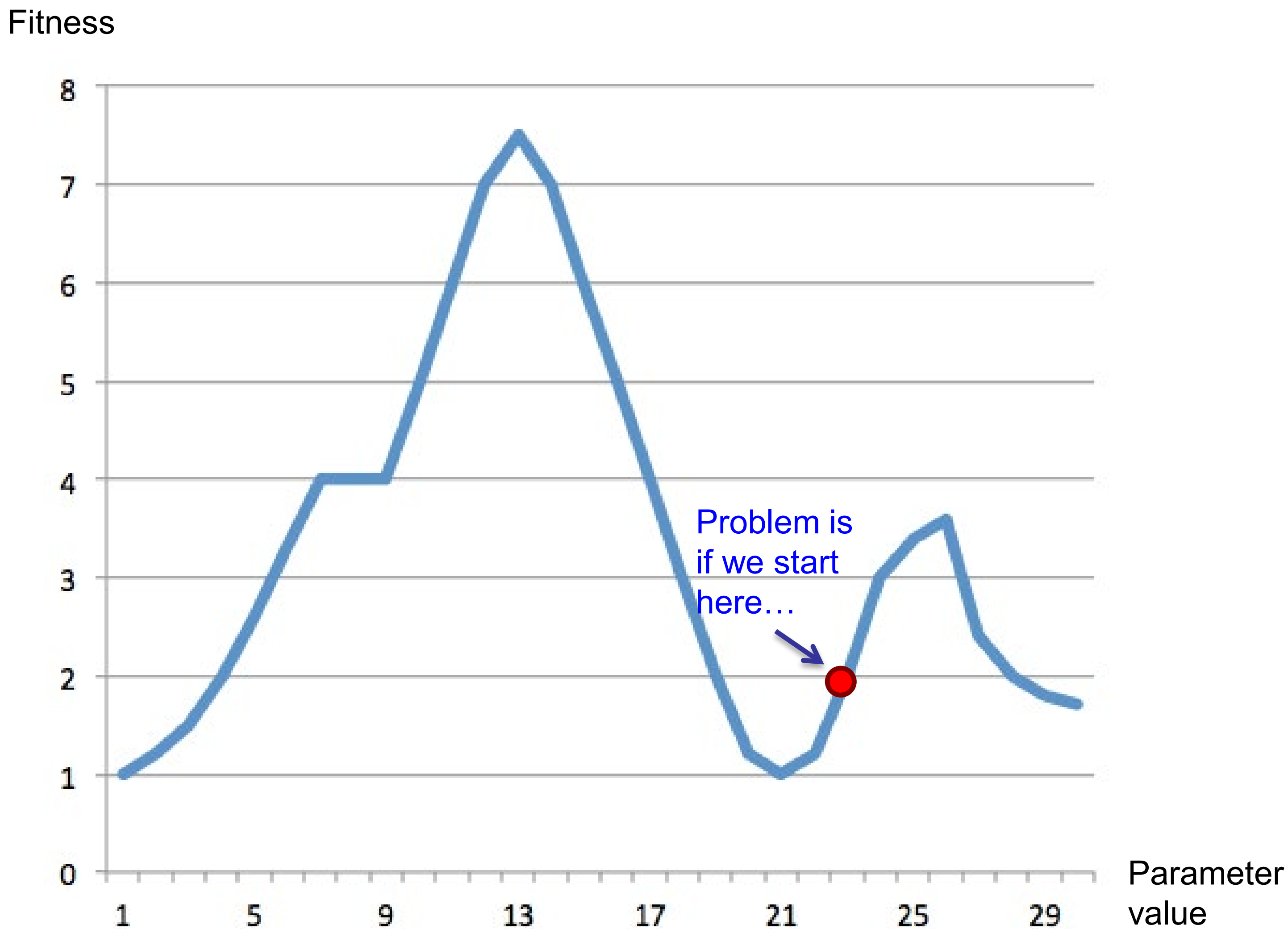
Hill Climbing



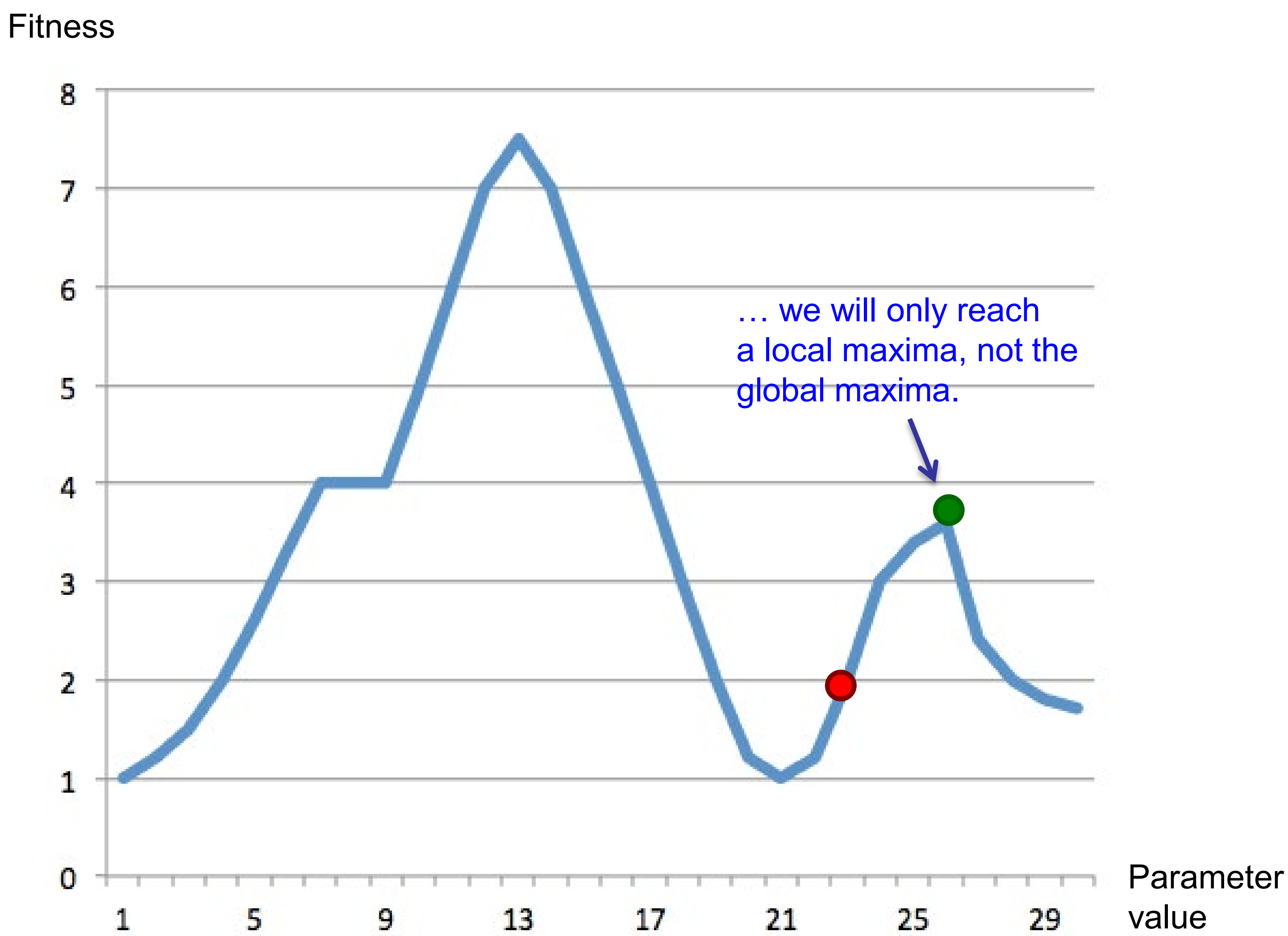
Hill Climbing



Hill Climbing



Hill Climbing



Simulated Annealing

- Combine Hill Climbing with random walk to reduce the possibility of getting stuck in local optima.
- The idea is:
    - Randomly select a new position.
    - If the new position is better (uphill), it is always accepted.
    - If it is worse (downhill), it is accepted with some probability less than 1.

Simulated Annealing

- The probability for selecting a worse situation depends on:
  - The difference in utility between the previous and the new state. The probability decreases exponentially with the ”badness” ΔE.
  - A *”temperature”* T starting at some value and for each iteration gradually decreases to zero.
  - Probability decreases with T.
  - No worse situation will be accepted if T is zero.

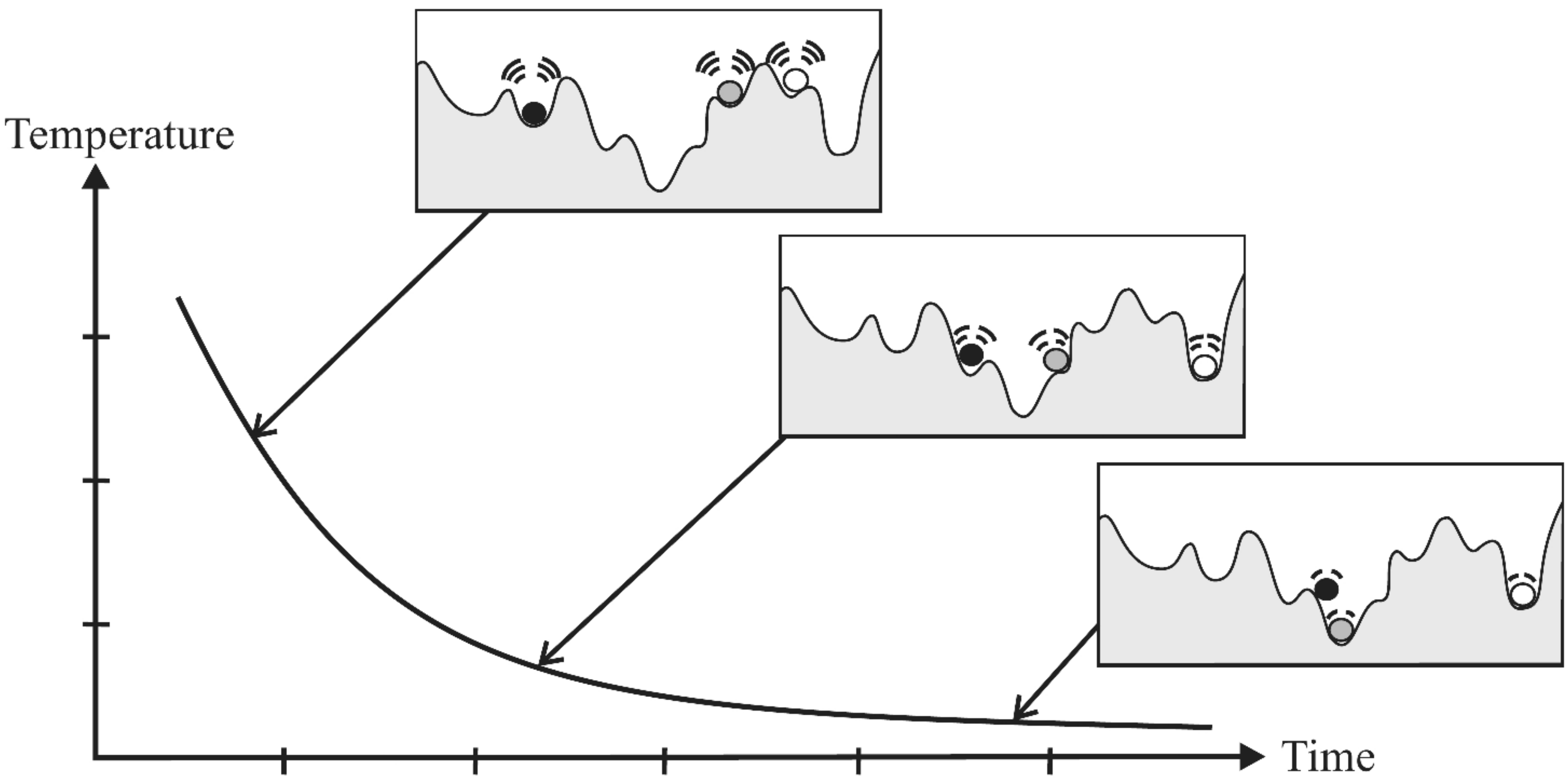
$$p \propto exp(\frac{-\Delta E}{T})$$

Simulated Annealing

- If T is decreasing slowly, the global optima will be found with probability close to 1.
- Only problem is slow decreasing of T leads to increased computation time.

- More likely to find global optima than Hill Climbing, but requires more computation time.

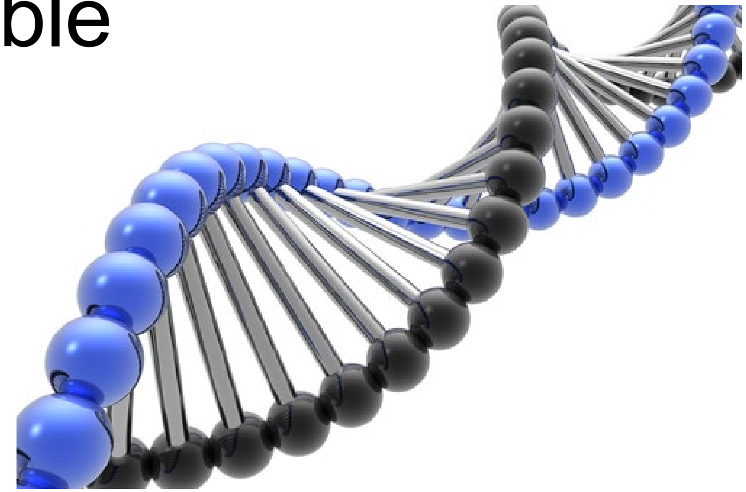
Simulated Annealing



Courtesty: University of Guanajuato, Department of Computer Engineering, Salamanca, Mexico

Genetic Algorithms

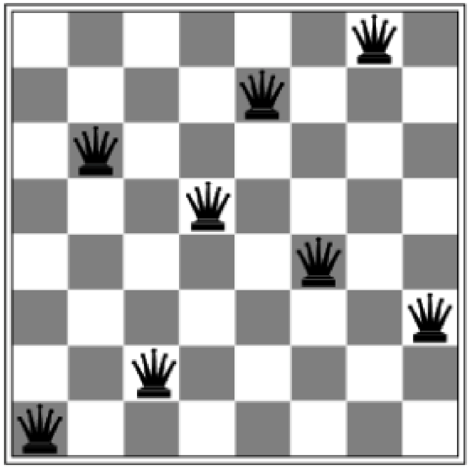
- GA is a technique with some similarities of biological evolution.
- The idea is evolving from already good solutions, to find even better ones.
- The first step is to represent the problem in a way that is usable by GAs.



Problem Representation

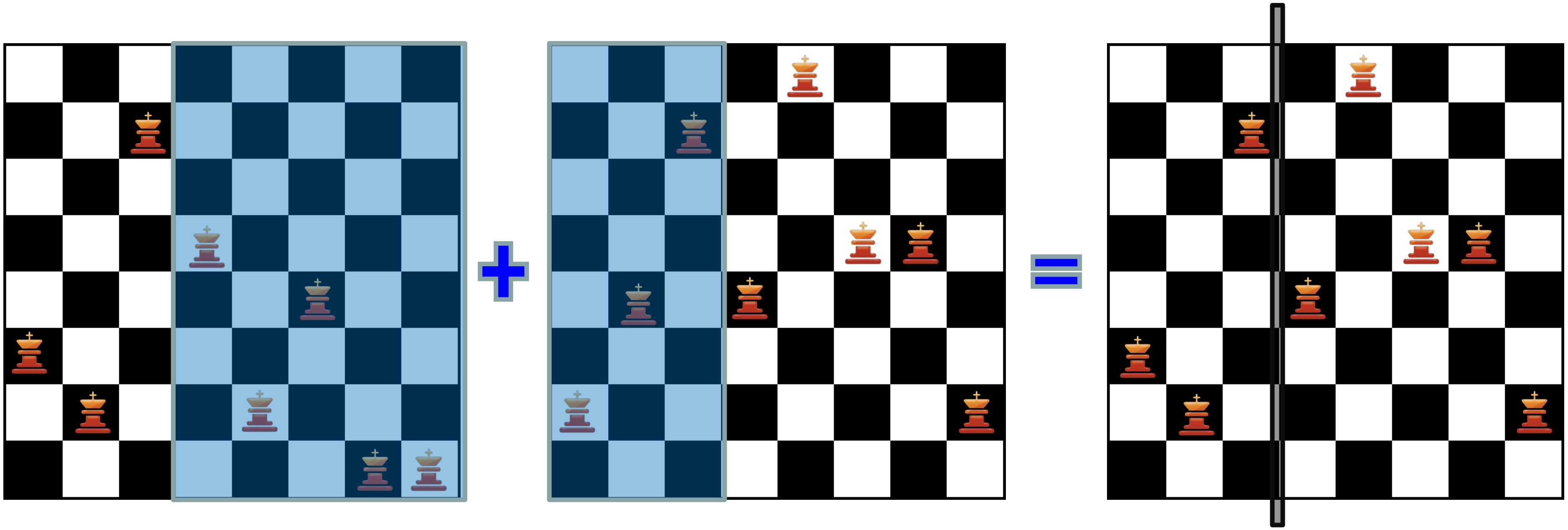
States: assume each queen has its own column, represent a state by listing a row where the queen is in each column (digits 1 to 8)

for example, the state below will be represented as 16257483



We need N $\log_2$ N bits for N queens

Genetic Algorithm
8-Queen Problem



8-Queen Problem

2 4 7 4 8 5 5 2

3 2 7 5 2 4 1 1

2 4 4 1 5 1 2 4

3 2 5 4 3 2 1 3

Represent States

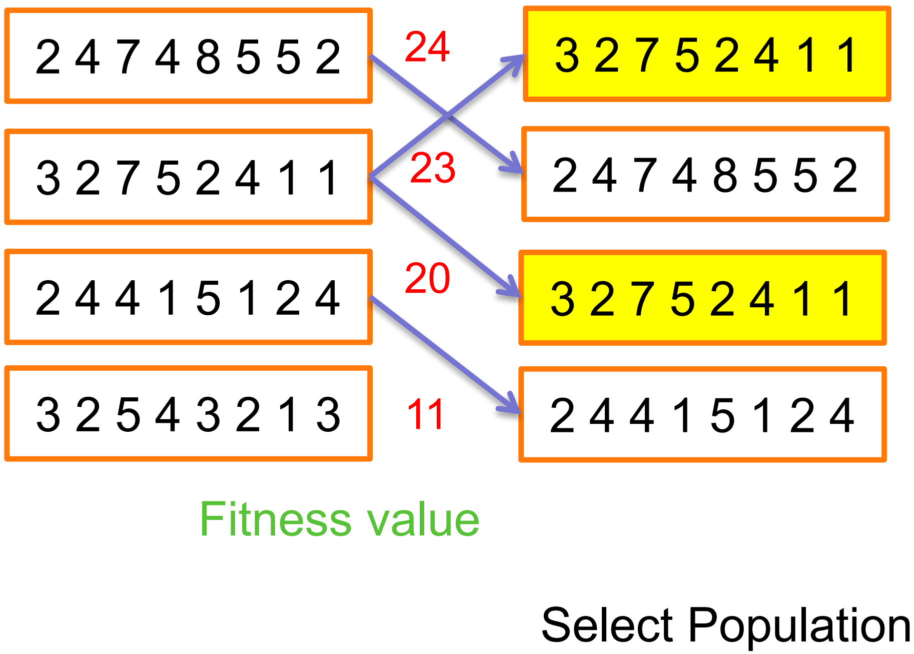
8-Queen Problem

2 4 7 4 8 5 5 2	24
3 2 7 5 2 4 1 1	23
2 4 4 1 5 1 2 4	20
3 2 5 4 3 2 1 3	11

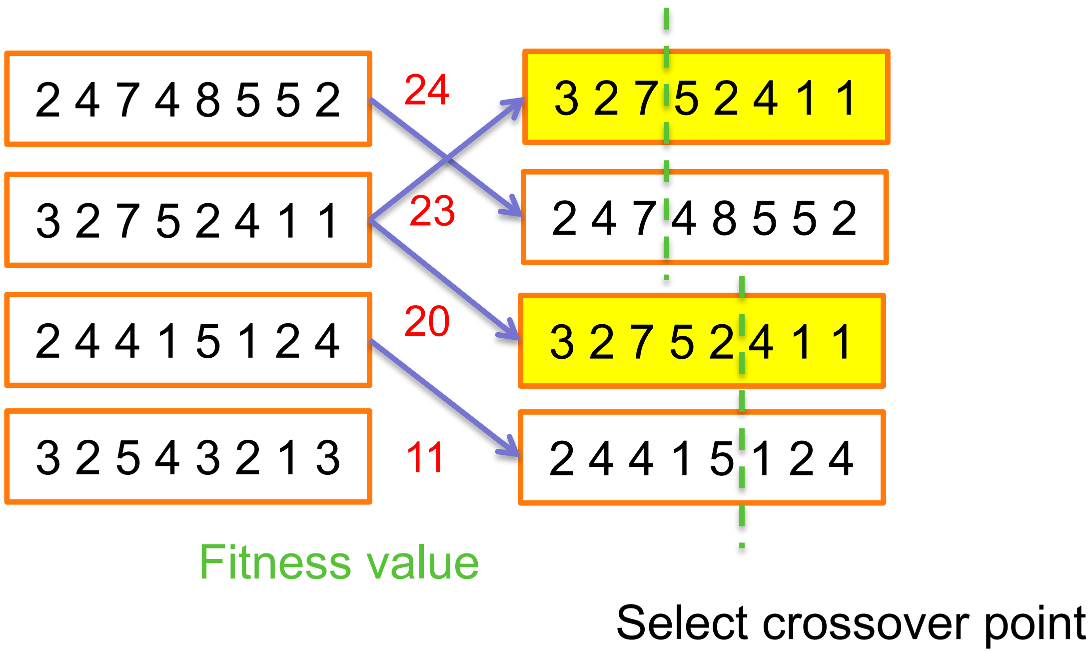
Fitness value

Fitness Evaluation

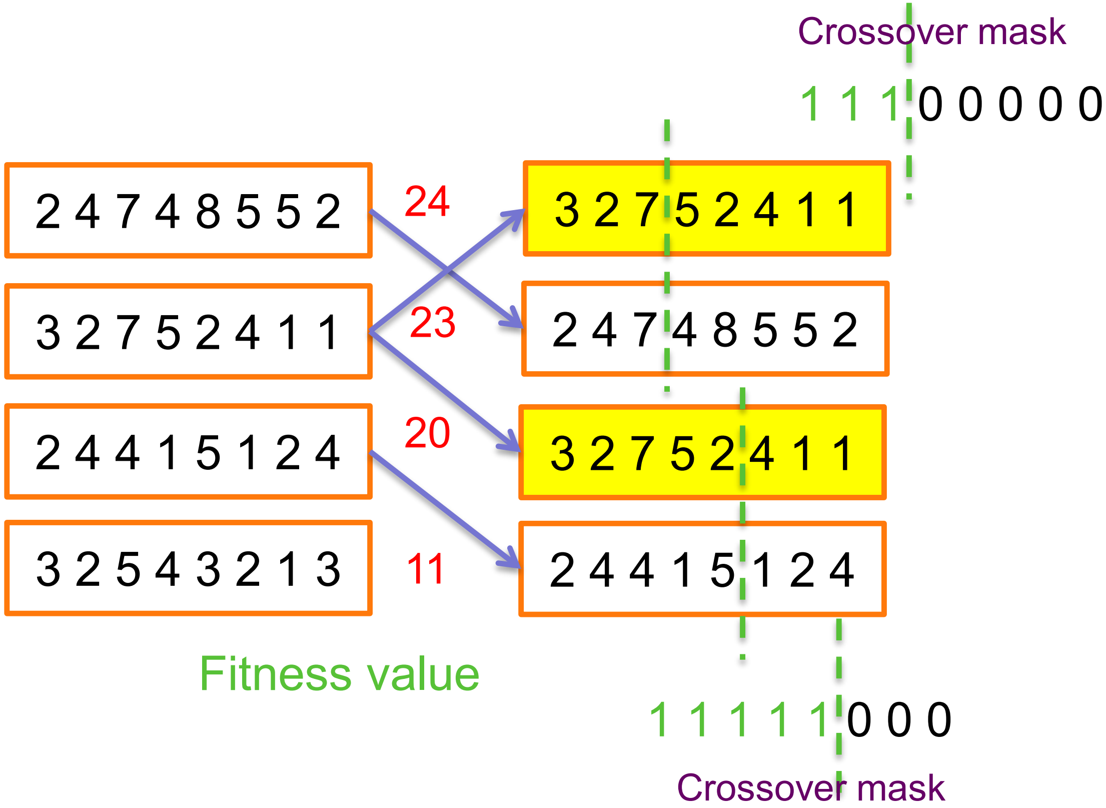
8-Queen Problem



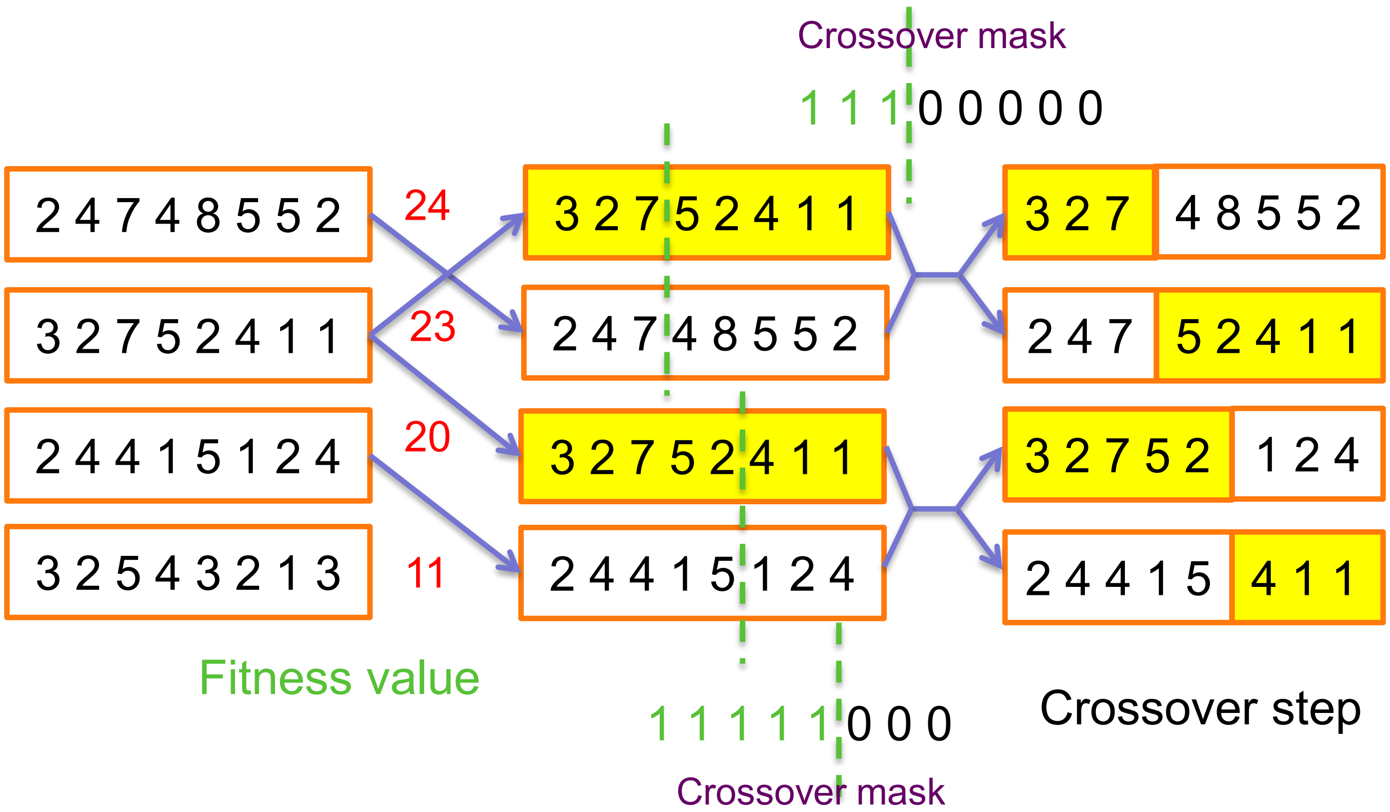
8-Queen Problem



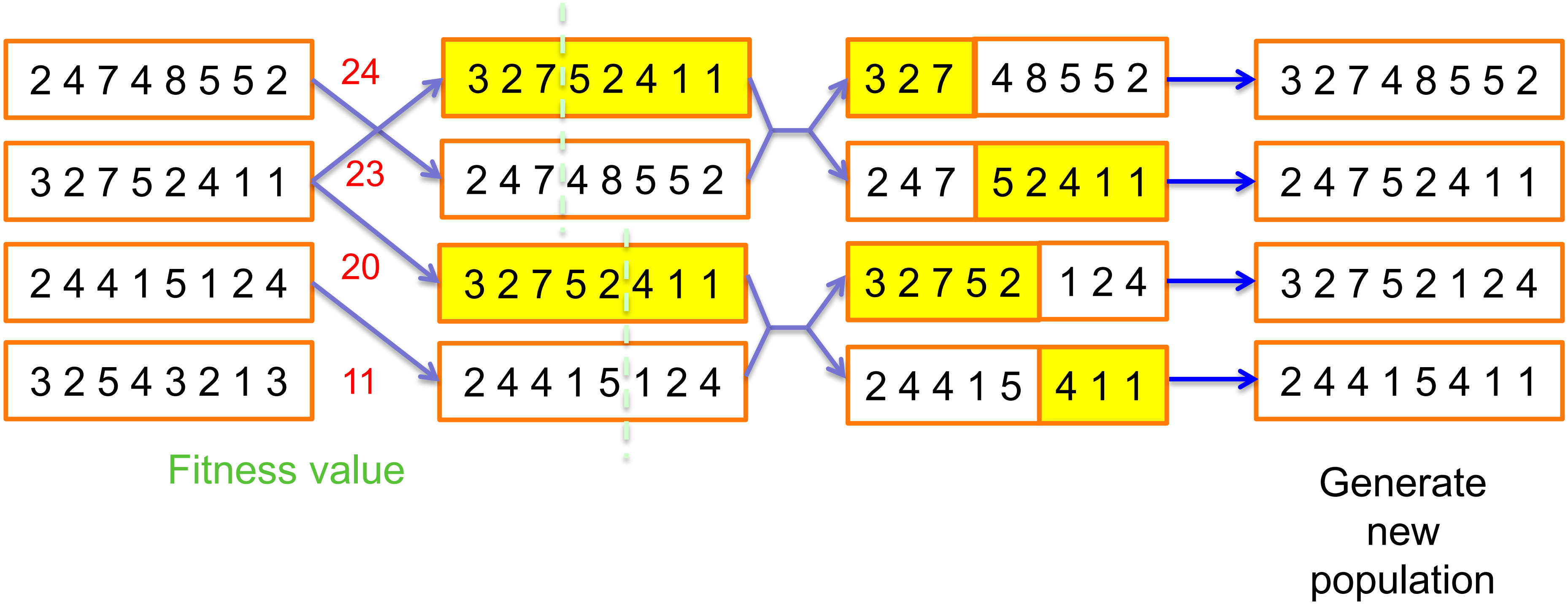
8-Queen Problem



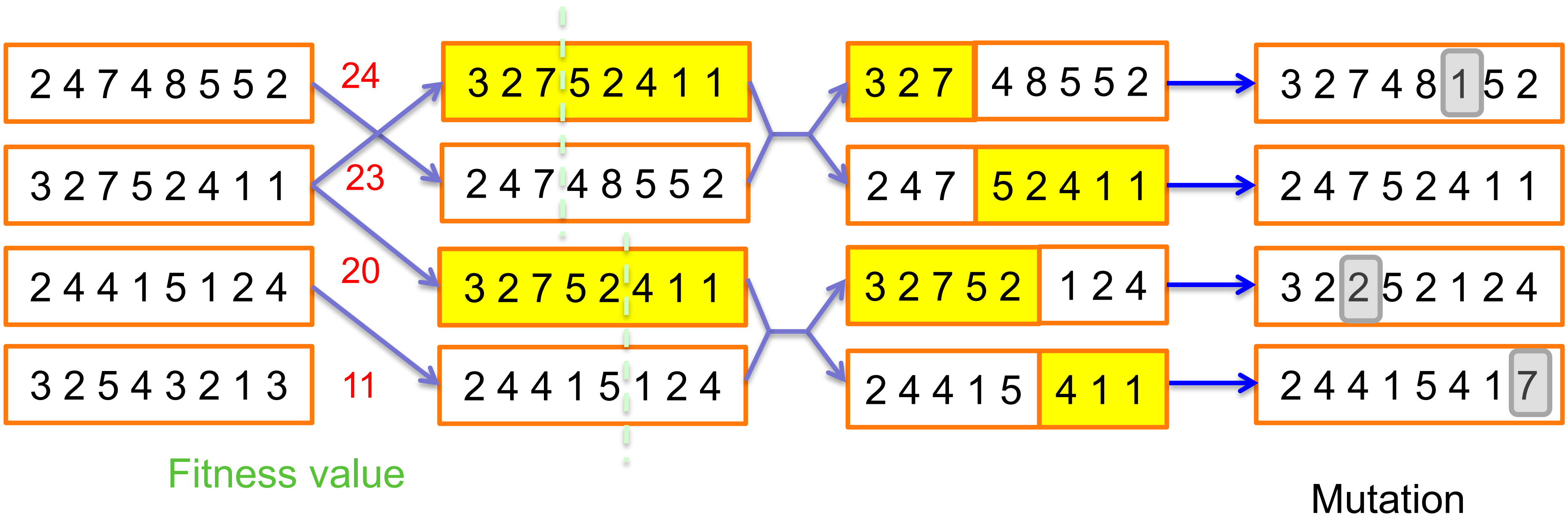
8-Queen Problem



8-Queen Problem



8-Queen Problem



Problem Representation

NoProcesses: [1:4]
Iterations: [1:20]
UseTechA: [true:false]
UseTechB: [true:false]

Optimization Parameters

Bitstring representation

100 10100 1 0

4 20 t f

The parameters to optimize must be represented in a bitstring!

Genetic Algorithm

```
START
Generate the initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP
```

(From newly generated population)

When do we stop?

1. After a number of iterations.

2. Termination criteria.

$$\boxed{\operatorname*{Max}_{x=1}^{p}[\text{Fitness}(h_x)] \geq \text{Threshold}_{\text{fitness}}}$$

Problems

- Time: If the evaluation of one hypothesis is time consuming, it might take ages to reach the termination criteria.
- Randomness in the system: If there are random factors that affect the result from the system, we might not reach the termination criteria at all.

That was all for this lecture

Vote link for the poll
http://etc.ch/btAi

Acknowledgements

Dr. Johan Hagelbäck
Linnæus University
johan.hagelback@lnu.se
http://aiguy.org