CSC242: Introduction to Artificial Intelligence

Lecture 1.5

Please put away all electronic devices

Announcements

- Unit 1 Exam: One week from today
- Project 1 due that day 1159PM

Local Search

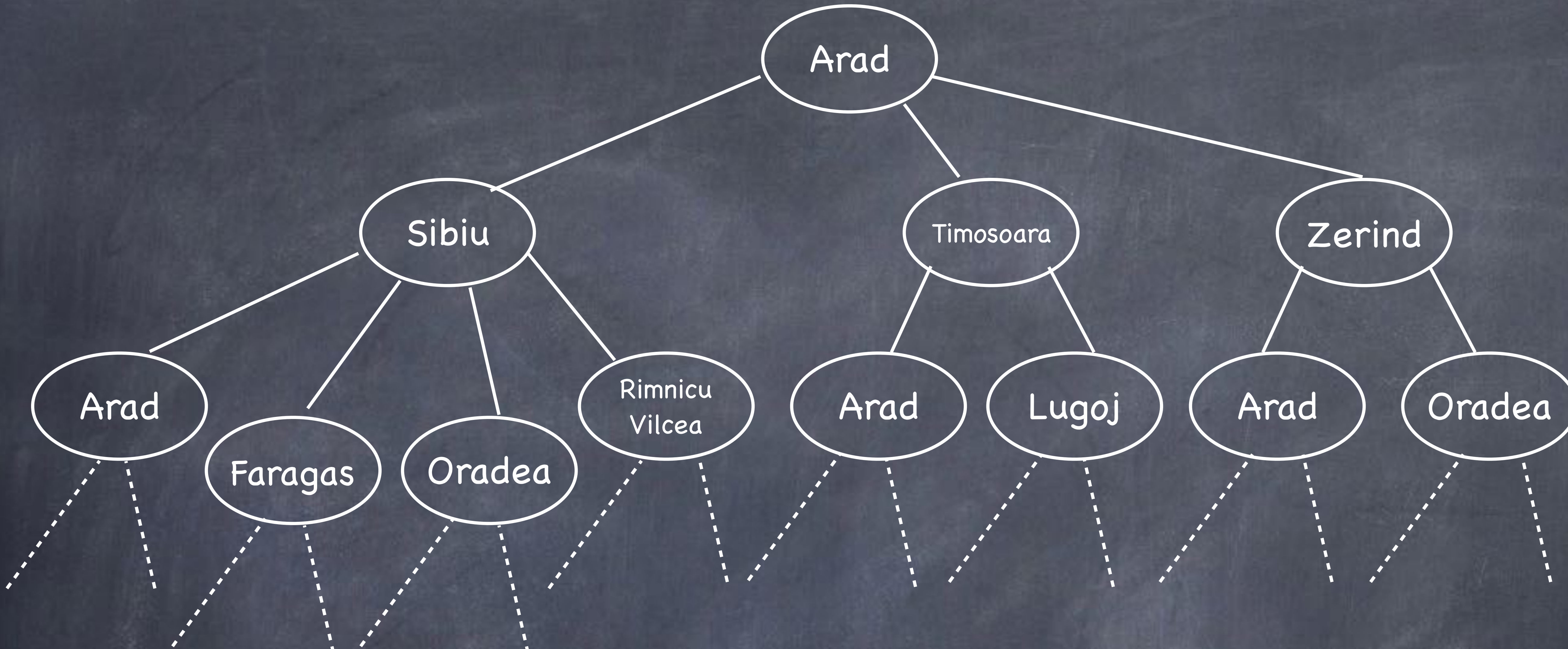
State Space

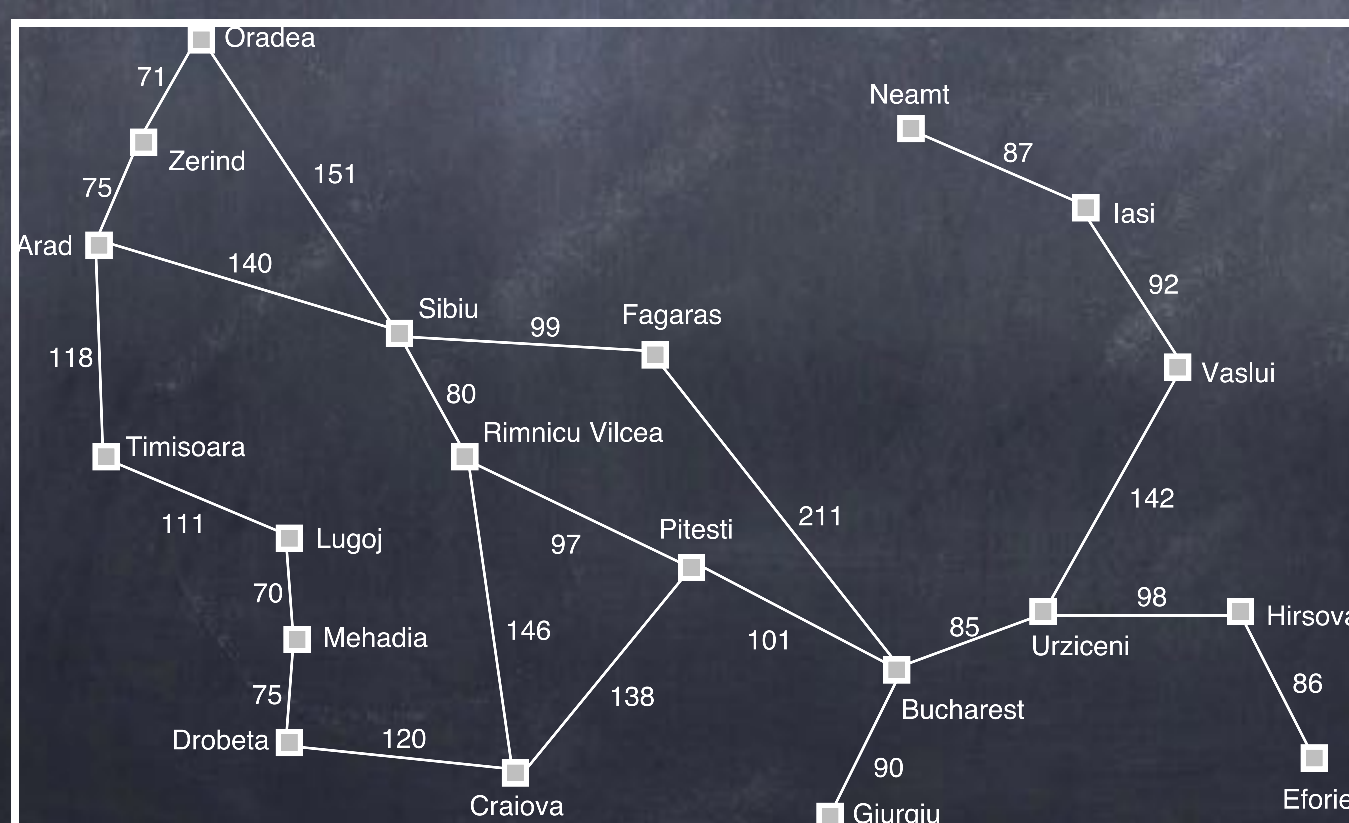
- Directed graph of states reachable from the initial state by some sequence of actions

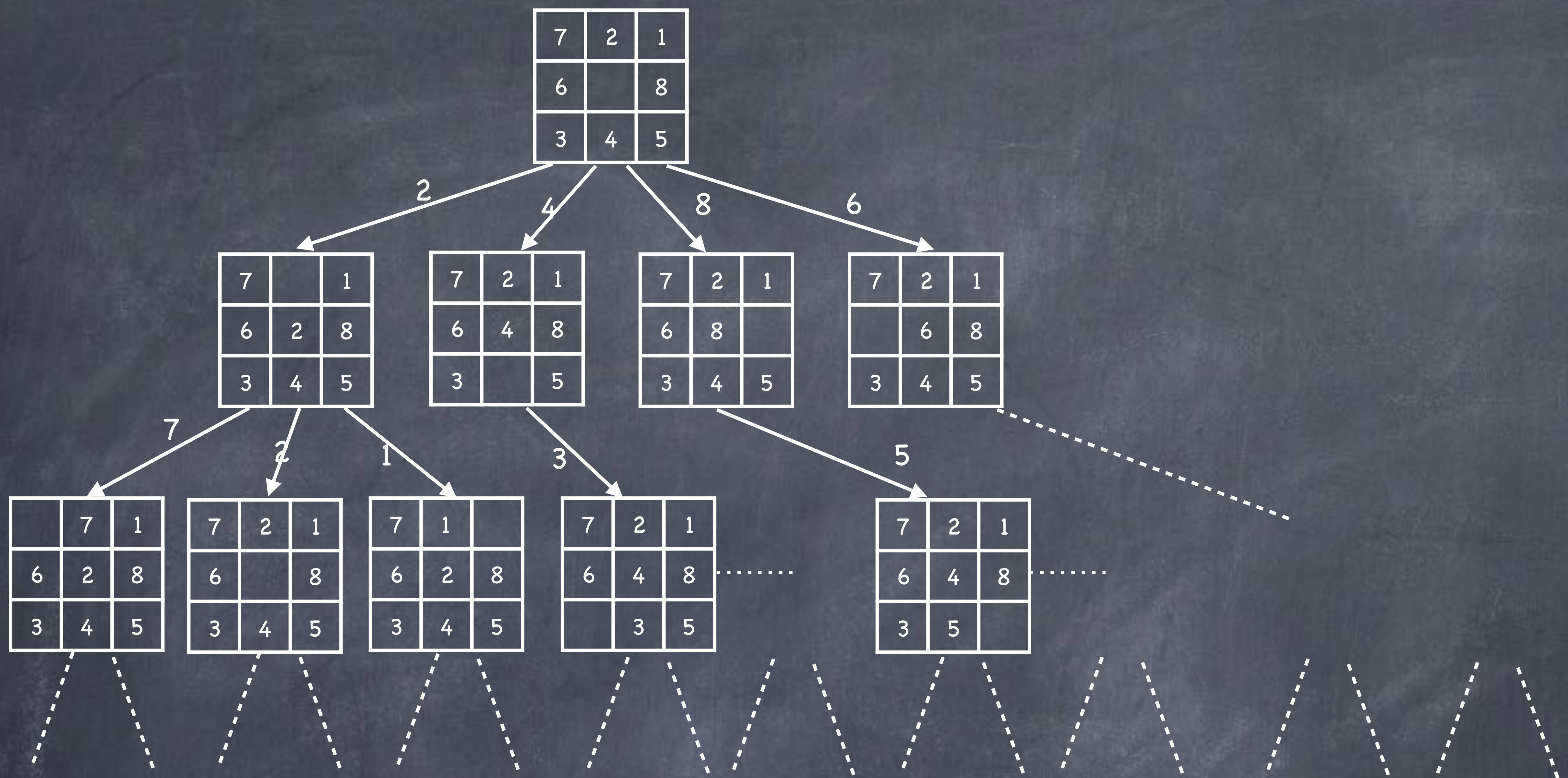
$$\langle V, E \rangle :$$

$$V = \{v_i \mid s_i \in \mathcal{S}\}$$

$$E = \{\langle v_i, v_j, a \rangle \mid s_j = \textsc{Result}(s_i, a)\}$$


Arad
Sibiu
Timosoara
Zerind
Arad
Faragas
Oradea
Rimnicu
Vilcea
Arad
Lugoj
Arad
Oradea


Oradea
71
Neamt
Zerind
87
151
Iasi
75
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Lugoj
97
Pitesti
70
98
85
Hirsova
146
101
Mehadia
Urziceni
86
75
138
Bucharest
Drobeta
120
90
Craiova
Eforie
Giurgiu



7 2 1
6 8
3 4 5
2
4
8
6
7 1
6 2 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 8
3 4 5
7 2 1
6 8
3 4 5
7
2
1
3
5
7 1
6 2 8
3 4 5
7 2 1
6 8
3 4 5
7 1
6 2 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 4 8
3 5

Coolest Program Ever

Initialize the frontier to just I

While the frontier is not empty:

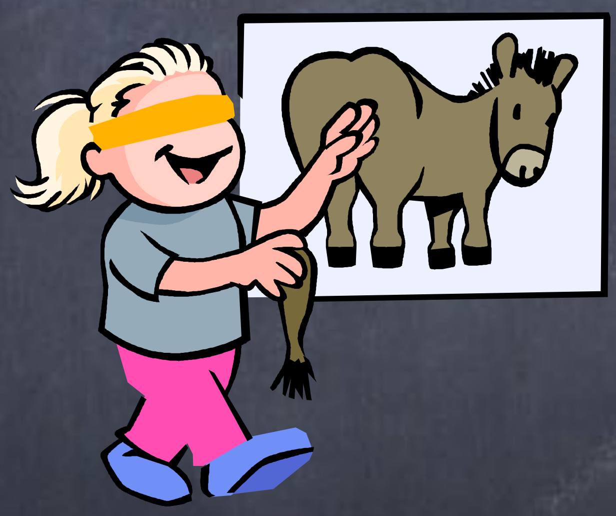
- Remove a state s from the frontier
- If $s \in G$:
    - Return solution to s
- else:
    - Add $successors(s)$ to the frontier

NO CODE REQUIRED!

Search Strategies

Uninformed

Informed (Heuristic)



No additional information about states

Can identify "promising" states

Search Strategies

Uninformed | Informed

	BFS	DFS (tree)	IDS	Greedy	A*	IDA*
Complete?	✓	✗	✓	✗	✓†	✓†
Optimal?	✓*	✗	✓*	✗	✓†	✓*†
Time	$O(b^d)$	$O(b^m)$	$O(b^d)$	$O(b^m)$	$O(b^{\epsilon d})$	$O(b^{\epsilon d})$
Space	$O(b^d)$	$O(bm)$	$O(bd)$	$O(b^m)$	$O(b^d)$	$O(bd)$

* If step costs are identical

† With an admissible heuristic

Adversarial Search

- DFS for adversarial problems
- MINIMAX and H-MINIMAX
- Back up utility values through alternating MIN and MAX (zero-sum game)
- Pruning search trees (e.g., α/β)
- Expectation for stochastic and partially observable environments (games)

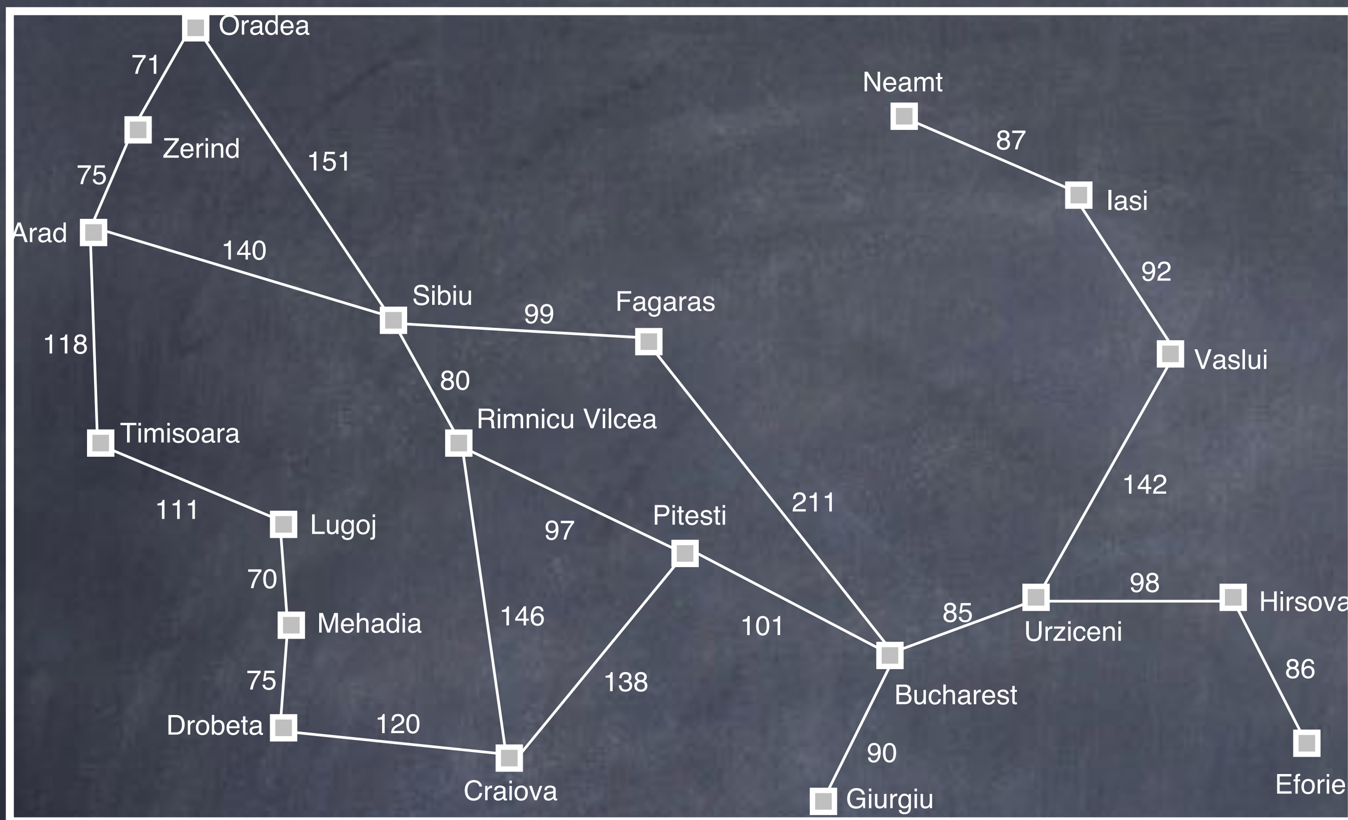
Systematic Search

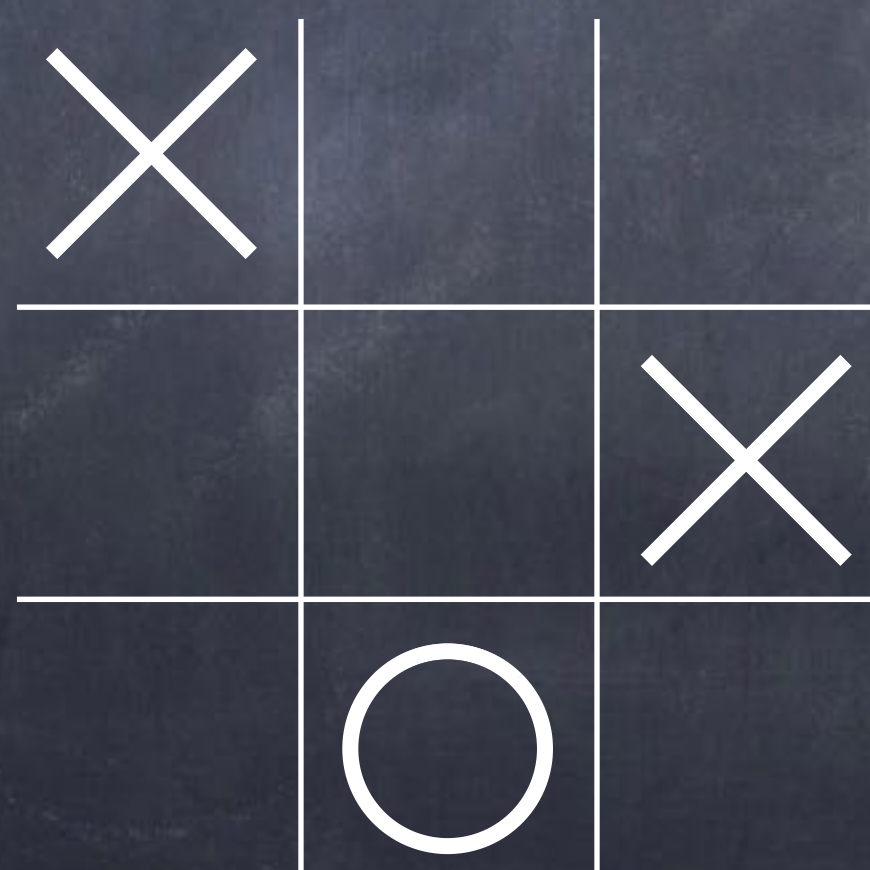
- Enumerates paths from initial state
- Records what alternatives have been explored at each point in the path

Good: Systematic $\rightarrow$ Exhaustive

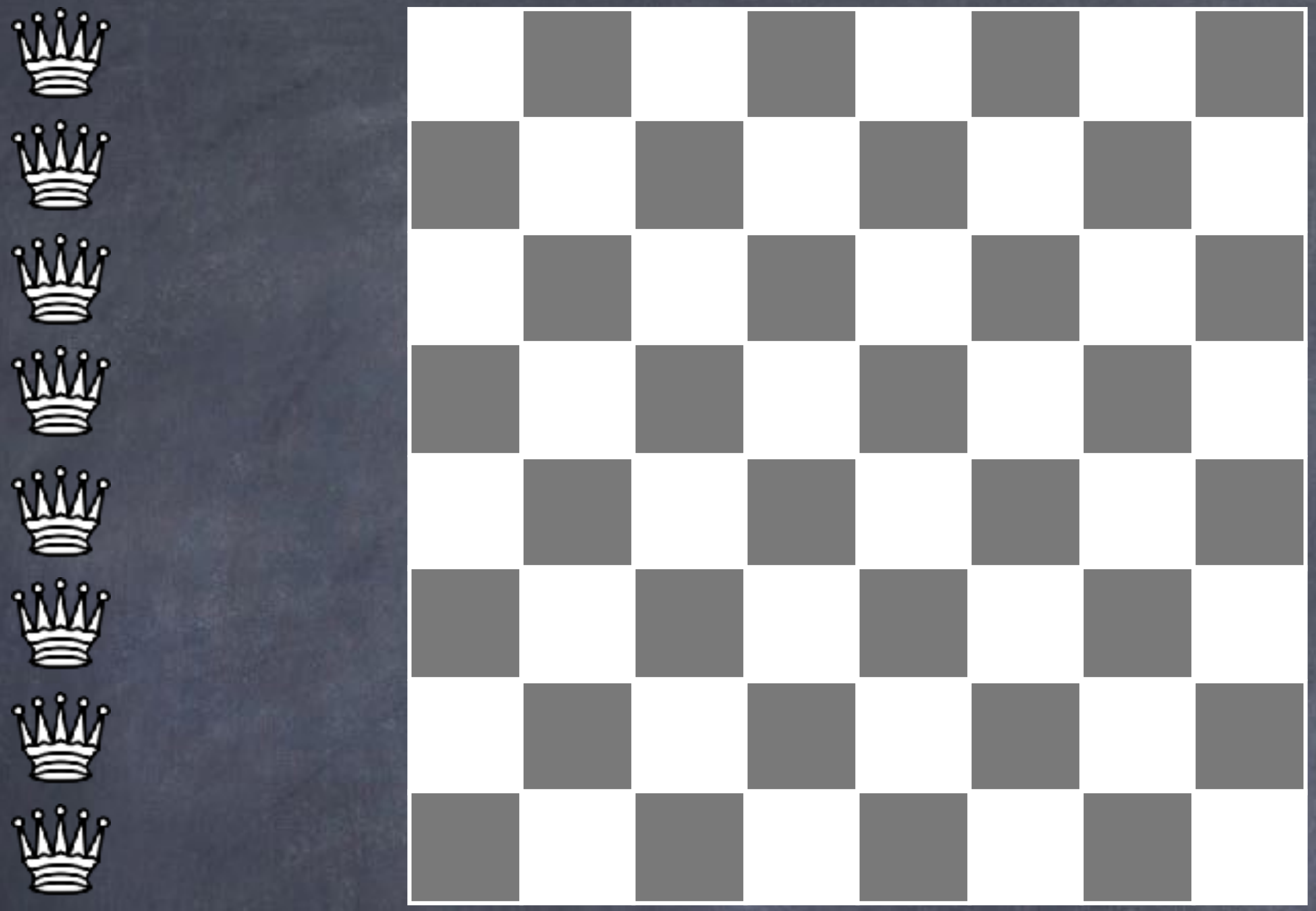
Bad: Exponential time and/or space

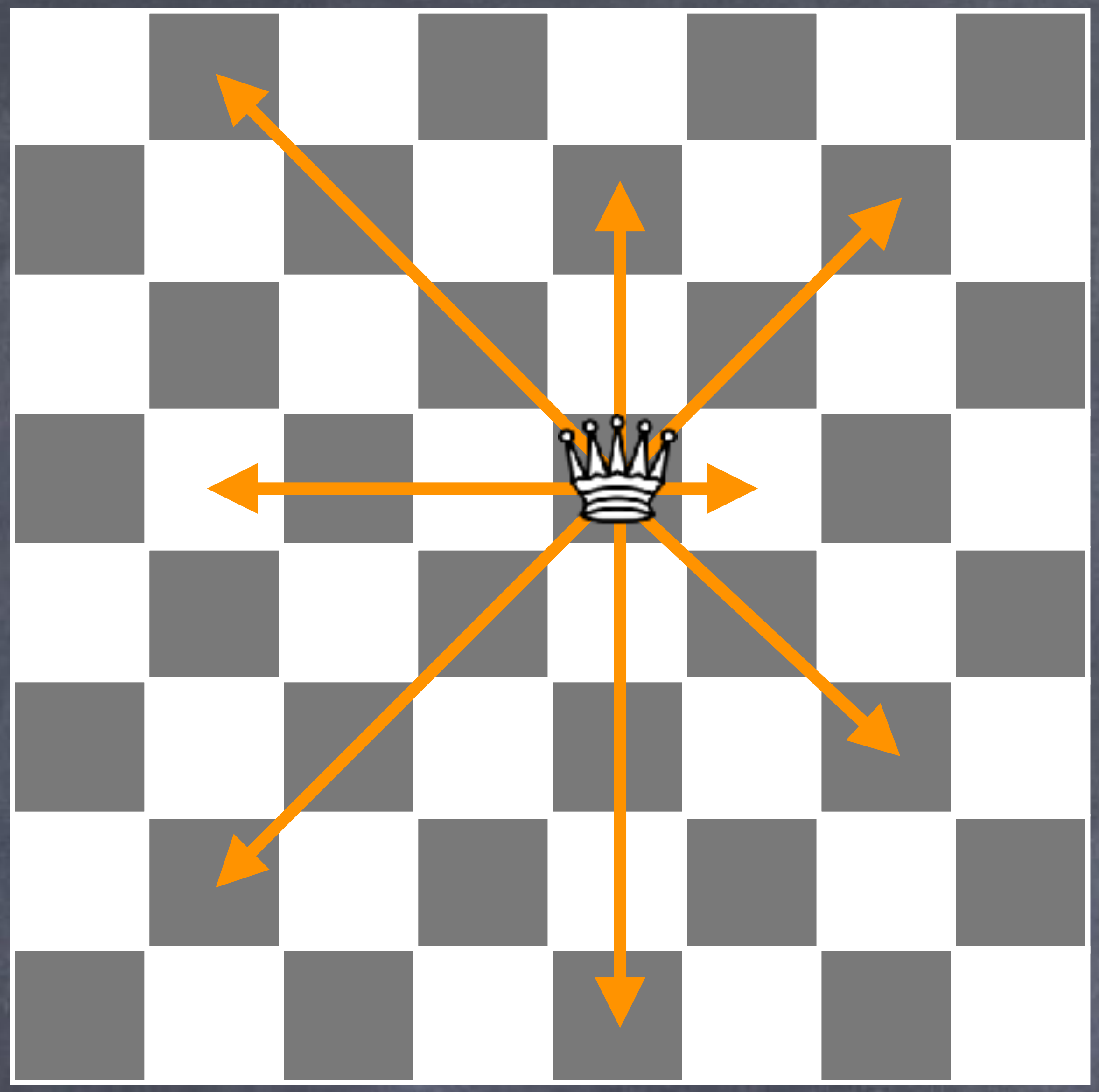
Local Search

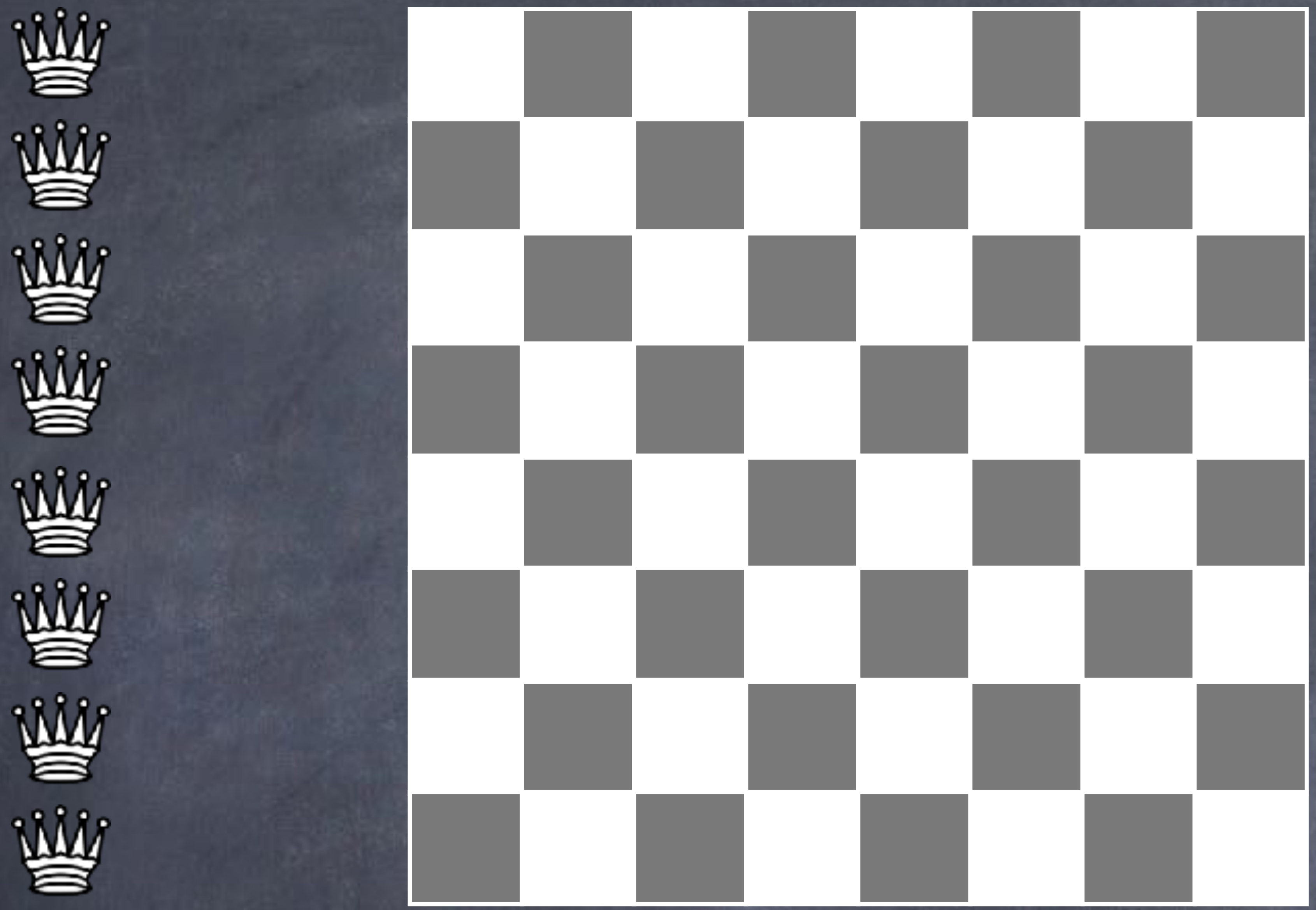

Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Drobeta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
Urziceni
98
Hirsova
85
86
Bucharest
90
Giurgiu
Eforie

ThinkFun
Fifteen Puzzle
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



Initial State
Goal State

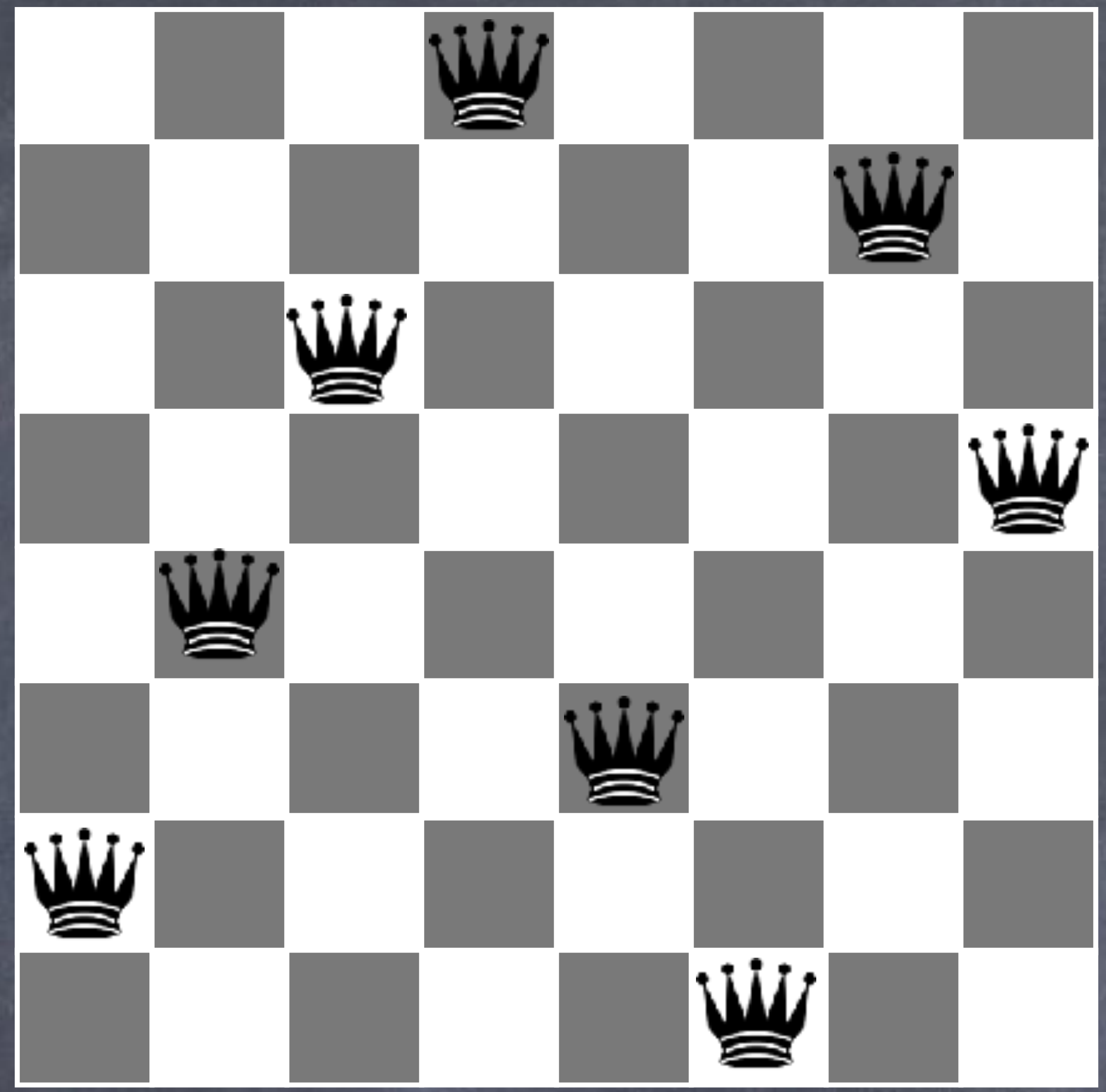




N-Queens as State-Space Search Problem

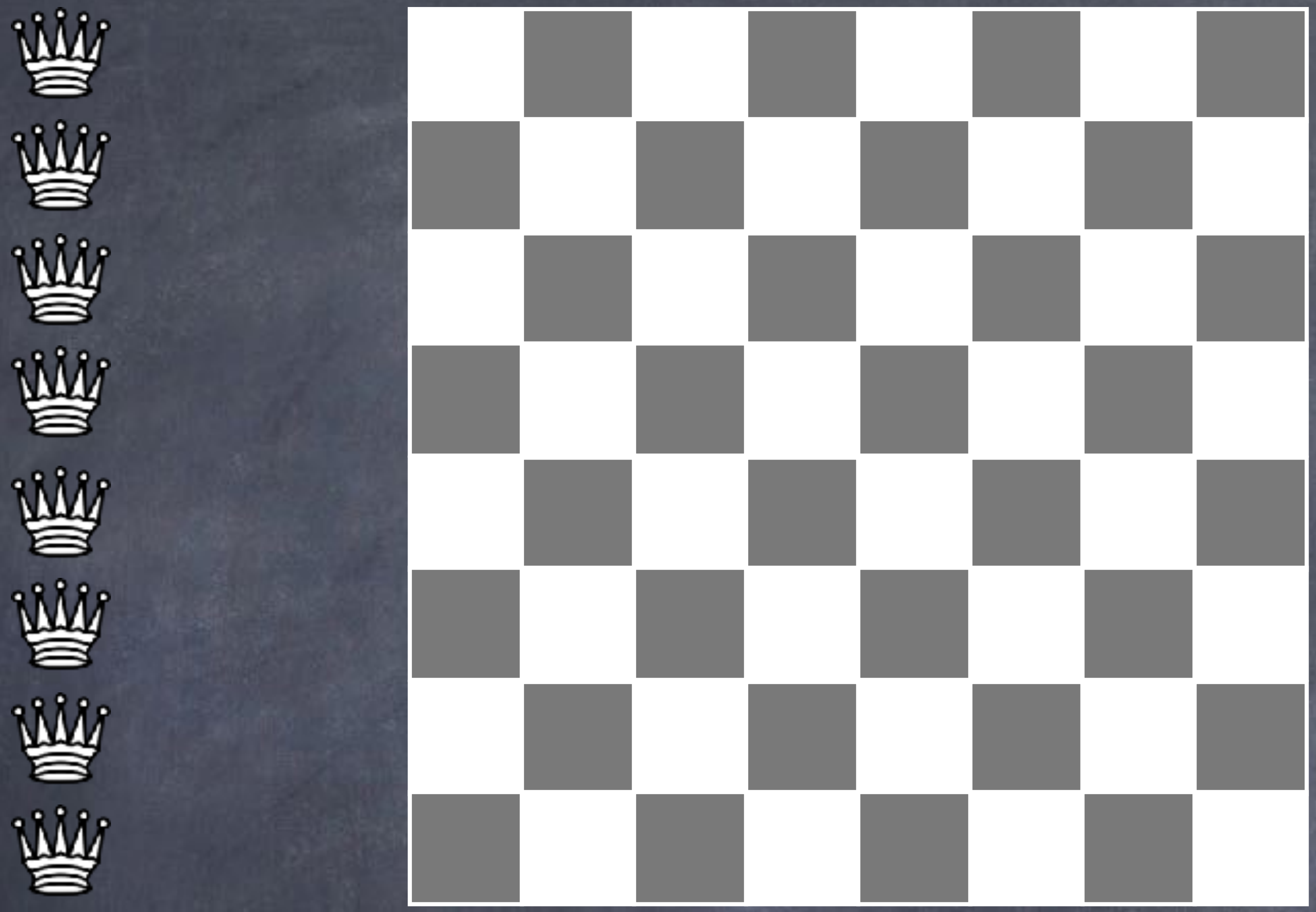
- State
- Actions
- Transition Model
- Initial State
- Goal State(s)/Test
- Step costs

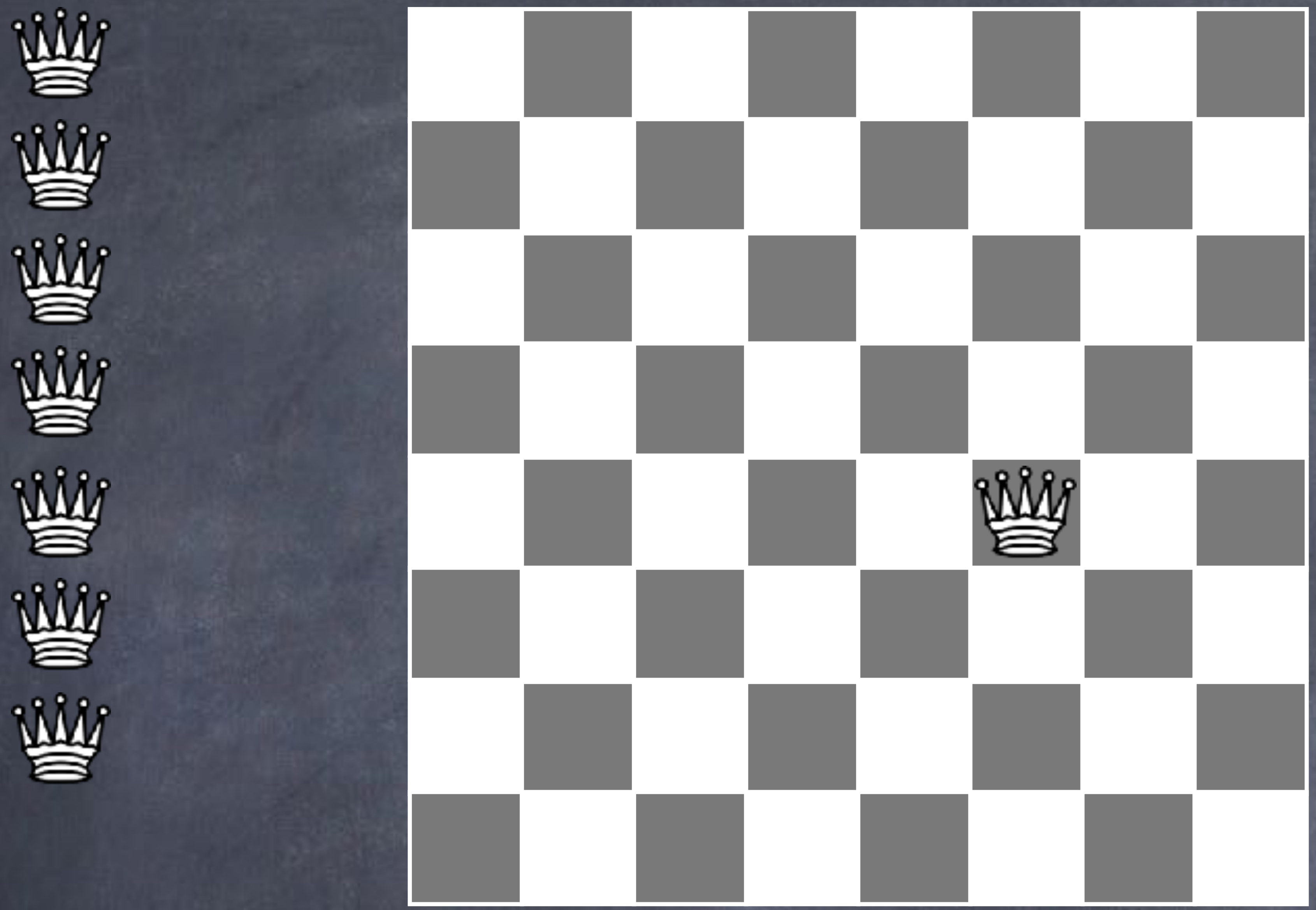


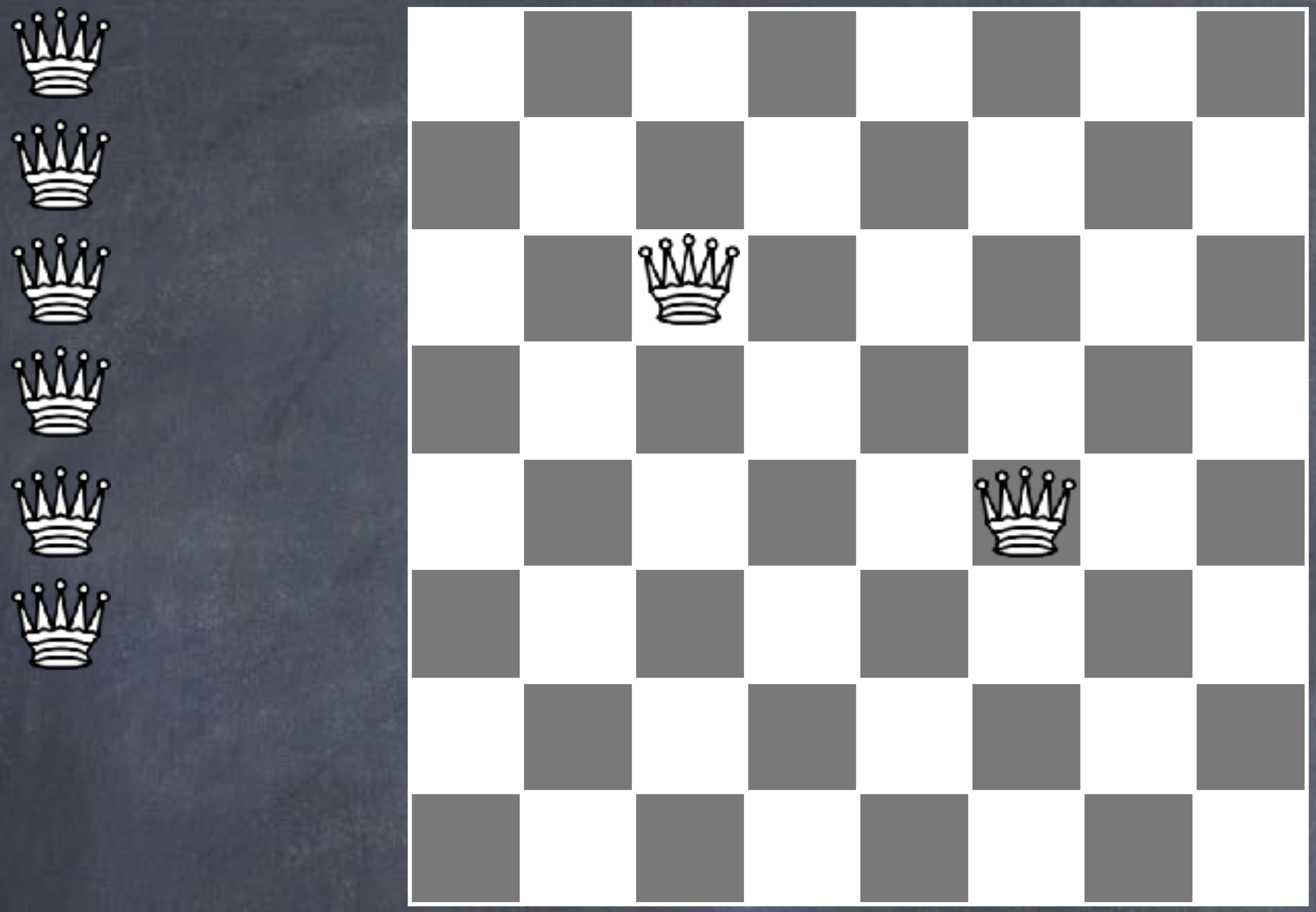
Local Search

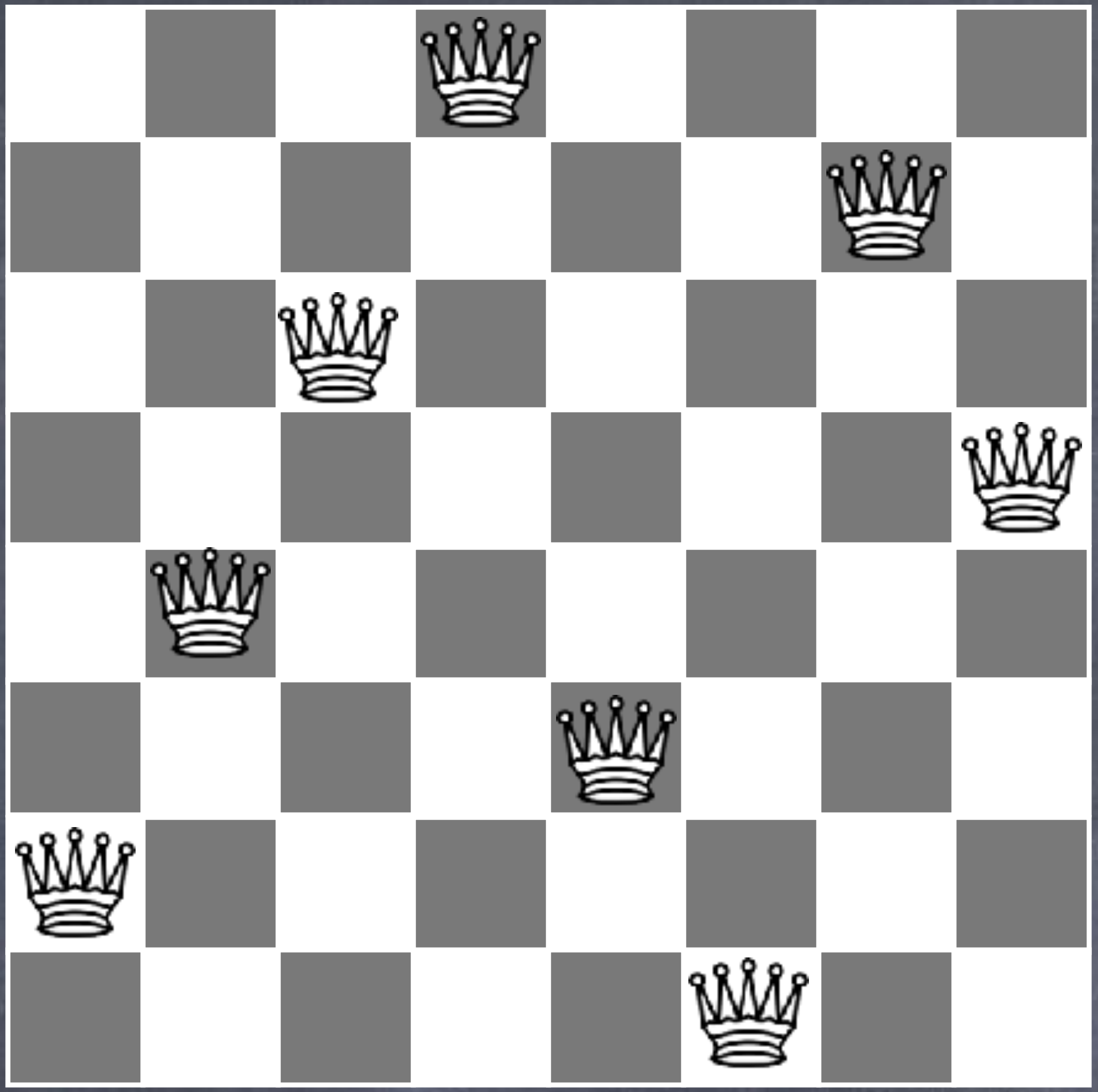
Local Search

- Start in initial state
- Select an applicable action, apply it, update the state
  - Repeat until you have a goal state









Local Search

- Evaluates and modifies a small number of current states
- Does not record history of search (paths, explored set, etc.)

Good: Very little (constant) memory

Bad: May not explore all alternatives

=> Incomplete

```
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
```

```
State localSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
```

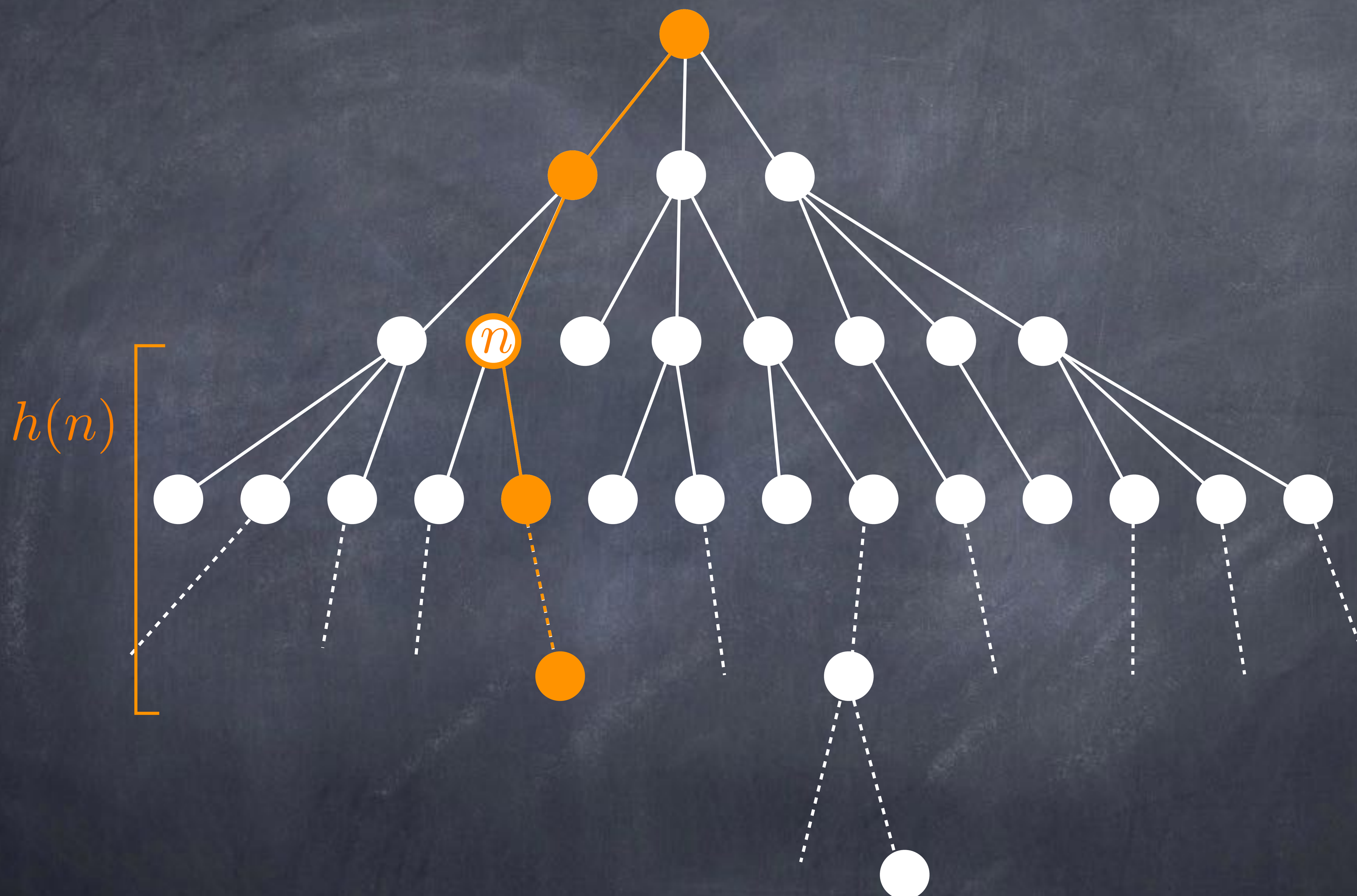
```
State localSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {

        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
```

```
State localSearch(Problem p) {

    while (true) {



        if (p.isGoalState(node.getState())) {

        }

        for (Node n : node.expand()) {



        }
    }
}
```

```
State localSearch(Problem p) {

    Node node = new Node(p.getInitialState());
    while (true) {




        if (p.isGoalState(node.getState())) {

        }

        for (Node n : node.expand()) {




        }
    }
}
```

```
State localSearch(Problem p) {

    Node node = new Node(p.getInitialState());
    while (true) {




        if (p.isGoalState(node.getState())) {
            return node.getState();
        }

        for (Node n : node.expand()) {



        }
    }
}
```

```
State localSearch(Problem p) {

    Node node = new Node(p.getInitialState());
    while (true) {




        if (p.isGoalState(node.getState())) {
            return node.getState();
        }

        for (Node n : node.expand()) {

            ???

        }
    }
}
```


n
$h(n)$

```
State localSearch(Problem p) {

    Node node = new Node(p.getInitialState());
    while (true) {




        if (p.isGoalState(node.getState())) {
            return node.getState();
        }

        for (Node n : node.expand()) {

            ???

        }
    }
}
```

```
State localSearch(Problem p) {

    Node node = new Node(p.getInitialState());
    while (true) {





        if (p.isGoalState(node.getState())) {
            return node.getState();
        }

        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(node)) {
                node = n;
            }
        }
    }
}
```

```
State localSearch(Problem p) {
   Node node = new Node(p.getInitialState());
    while (true) {
        if (p.isGoalState(node.getState())) {
            return node.getState();
        }
        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(node)) {
                node = n;
            }
        }
    }
}
```

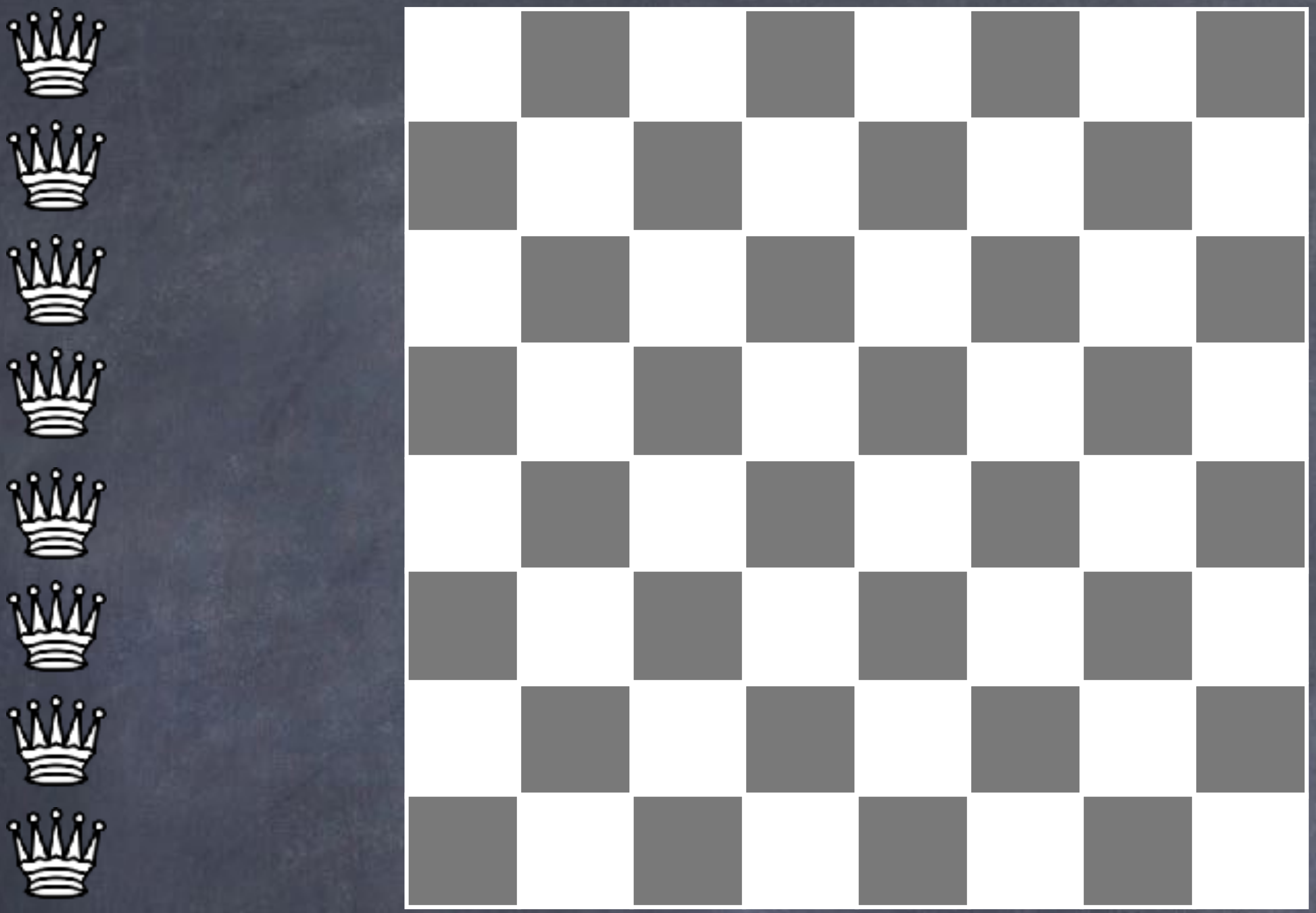
```
State localSearch(Problem p) {
   Node node = new Node(p.getInitialState());
    while (true) {
        if (p.isGoalState(node.getState())) {
            return node.getState();
        }
        Node next = null;
        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(node)) {
                next = n;
            }
        }
        if (next == null) {
            return node.getState();
        } else {
            node = next;
        }
   }
}
```

```
State localSearch(Problem p) {
   Node node = new Node(p.getInitialState());
    while (true) {
       Node next = null;
        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(node)) {
                node = n;
            }
        }
        if (next == null) {
            return node.getState();
        } else {
            node = next;
        }
   }
}
```

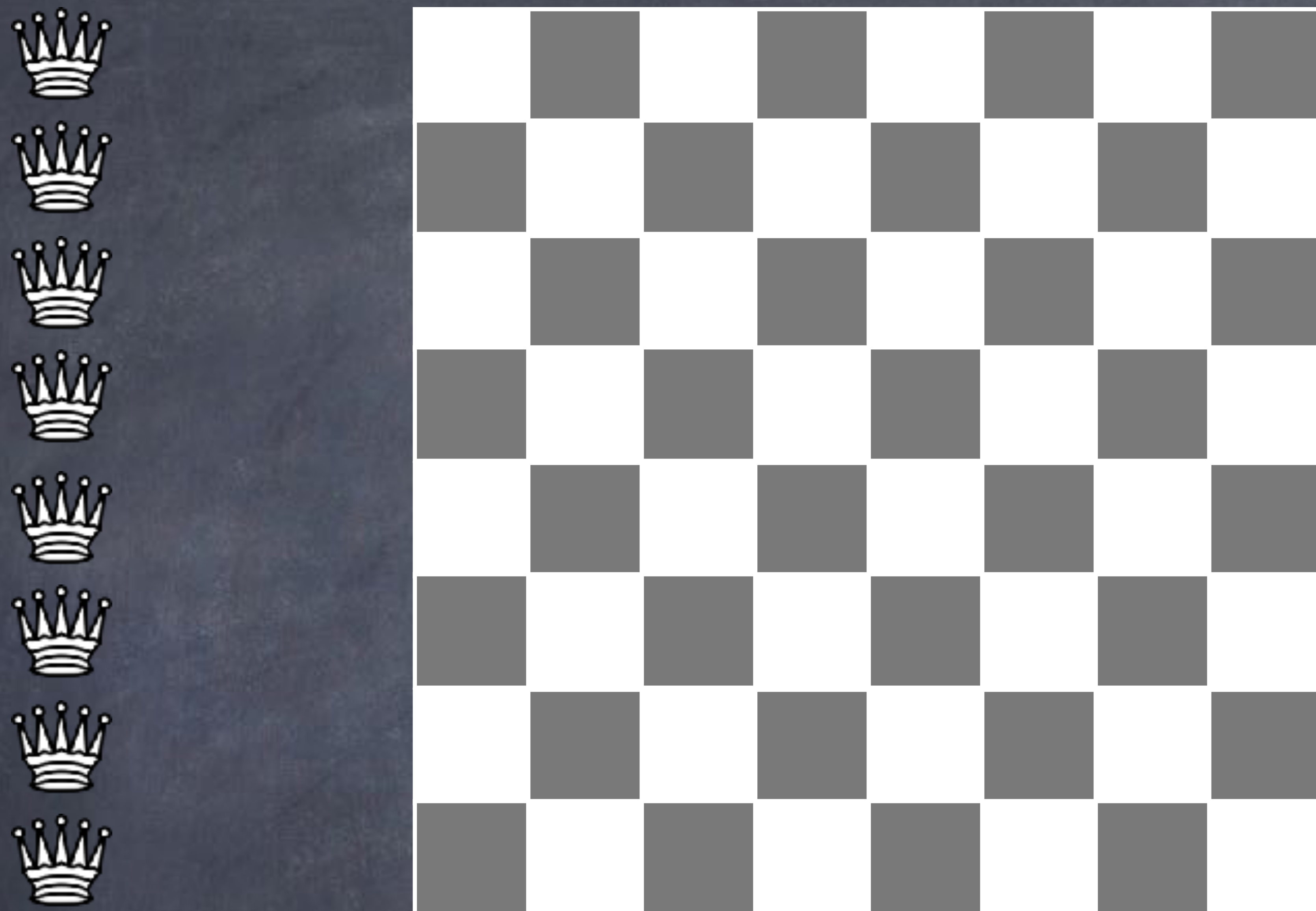
Hill-climbing Search

- Move through state space in the direction of increasing value ("uphill")

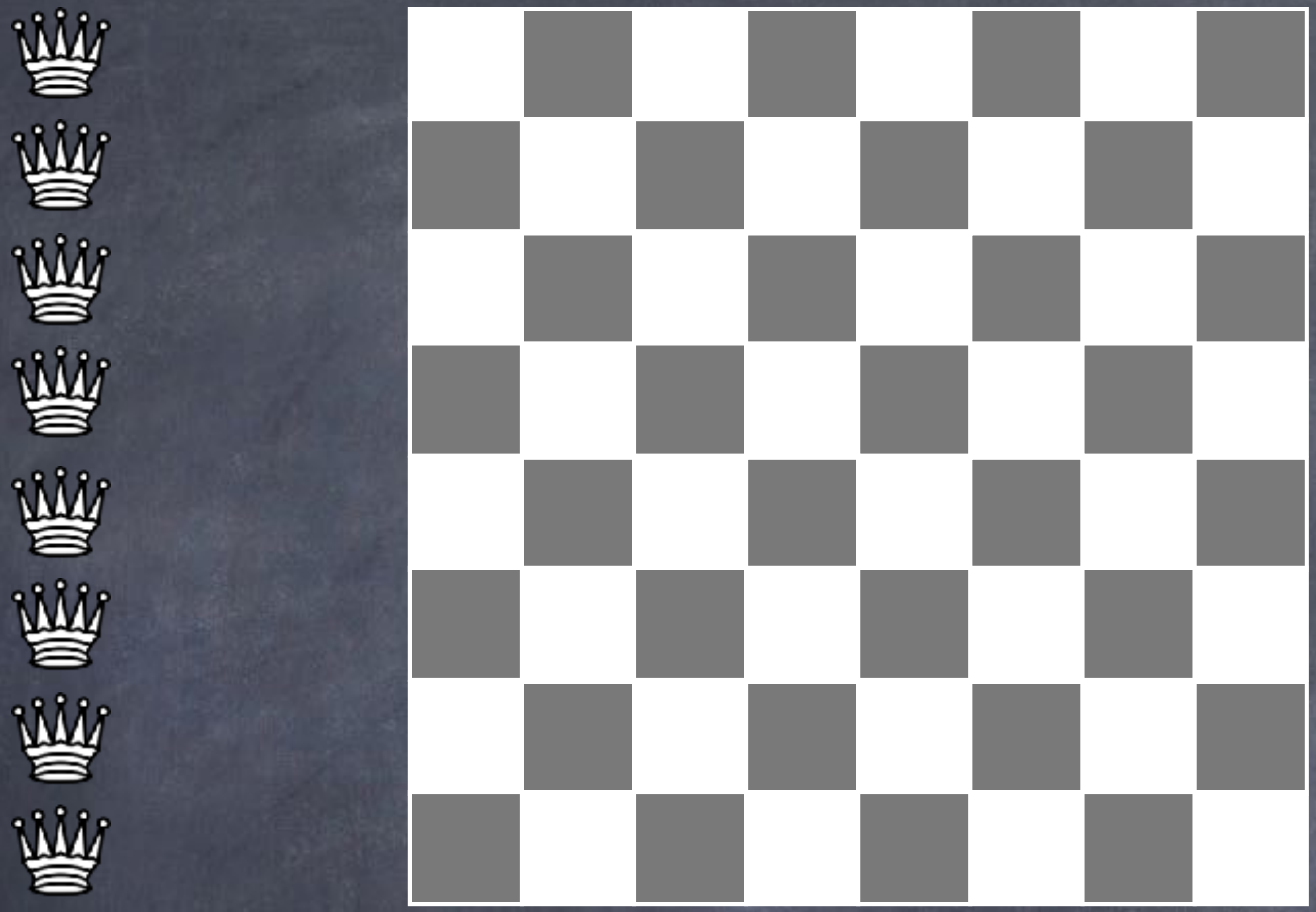
State-space landscape



State: $[r_0, \ldots, r_7]$

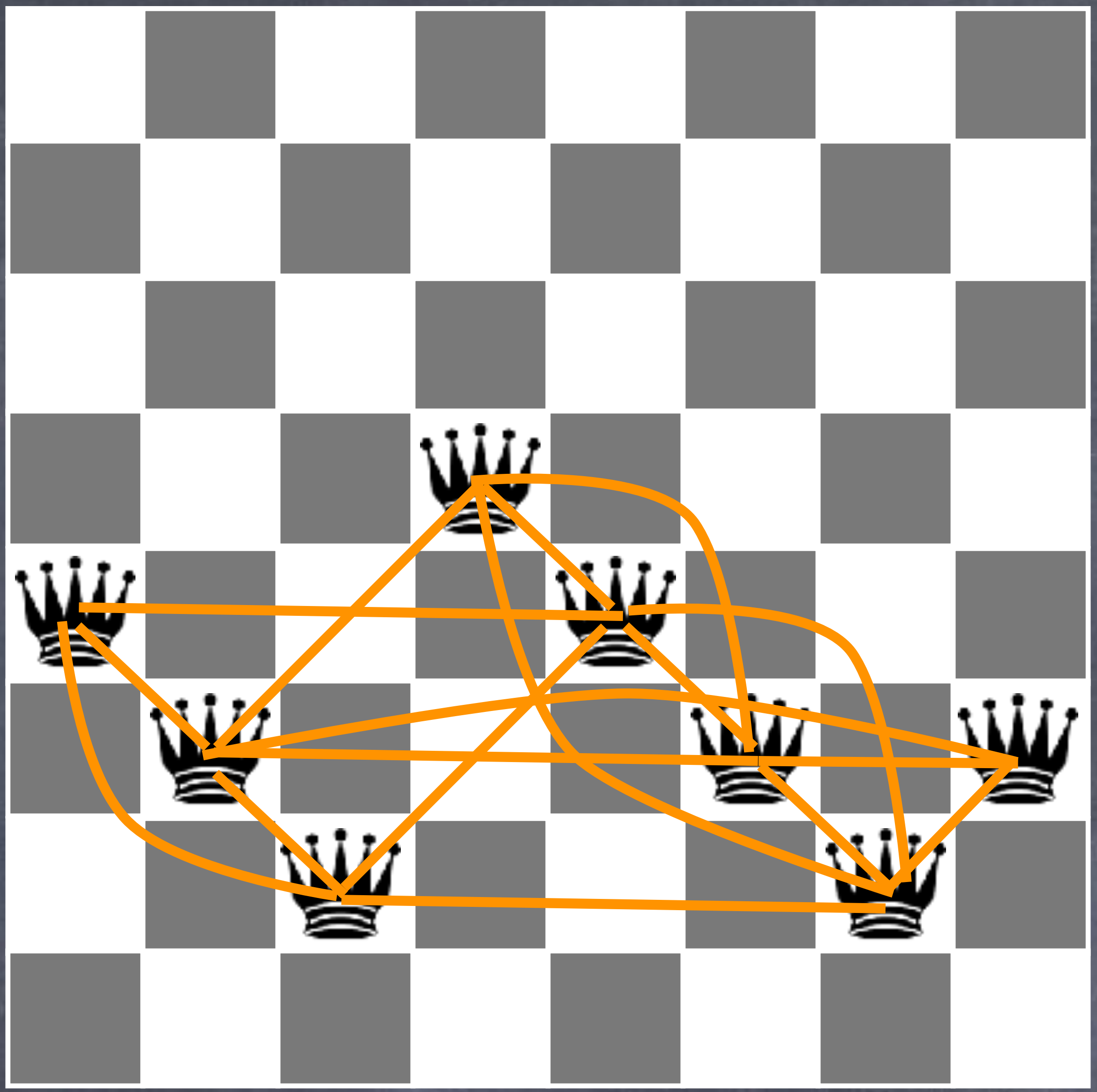


State: $[r_0, \ldots, r_7]$ Action: $< i, r_i >$

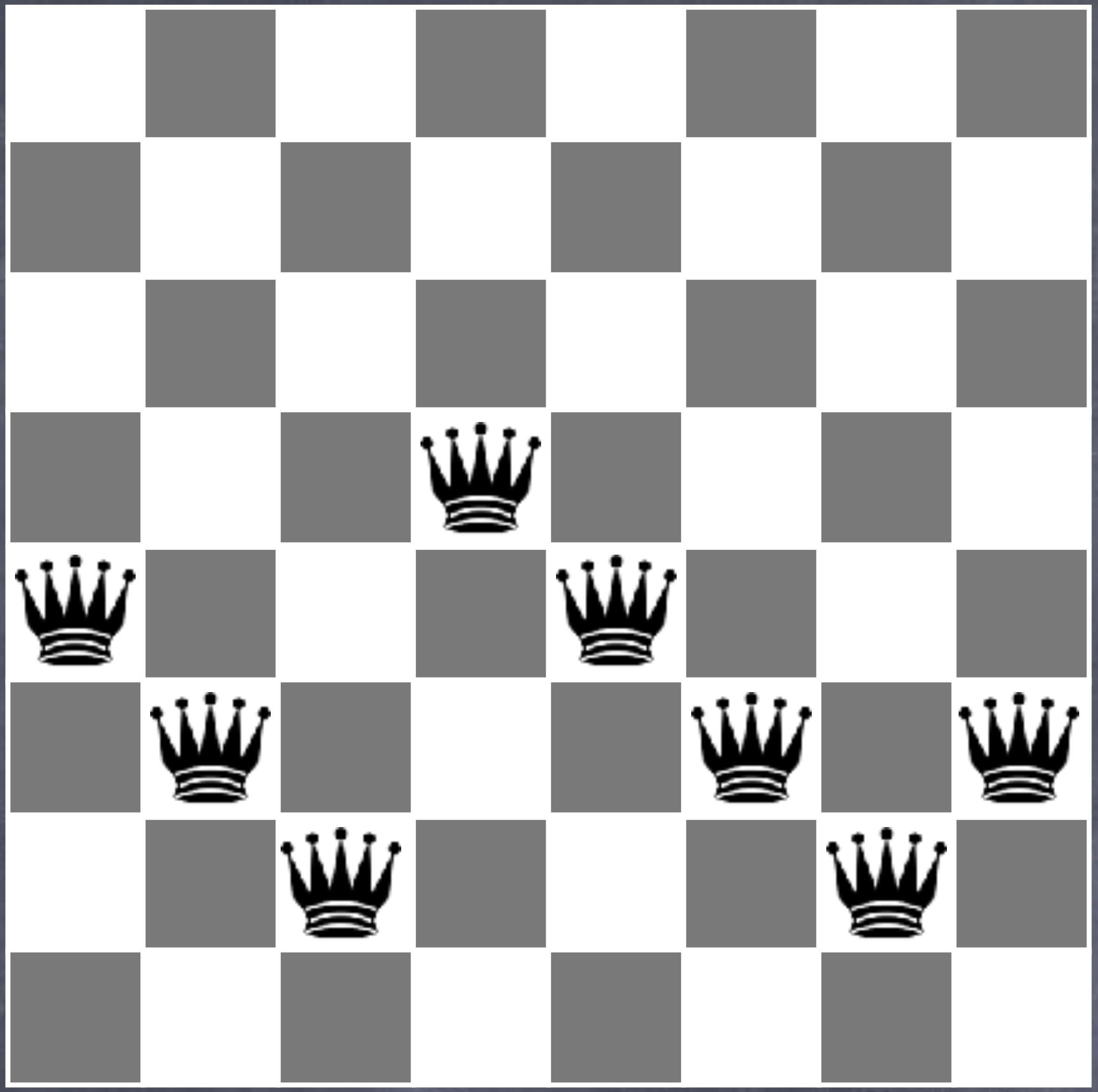


State: $[r_0, \ldots, r_7]$ Action: $< i, r_i >$

$h(n)$ = # of pairs of queens attacking each other



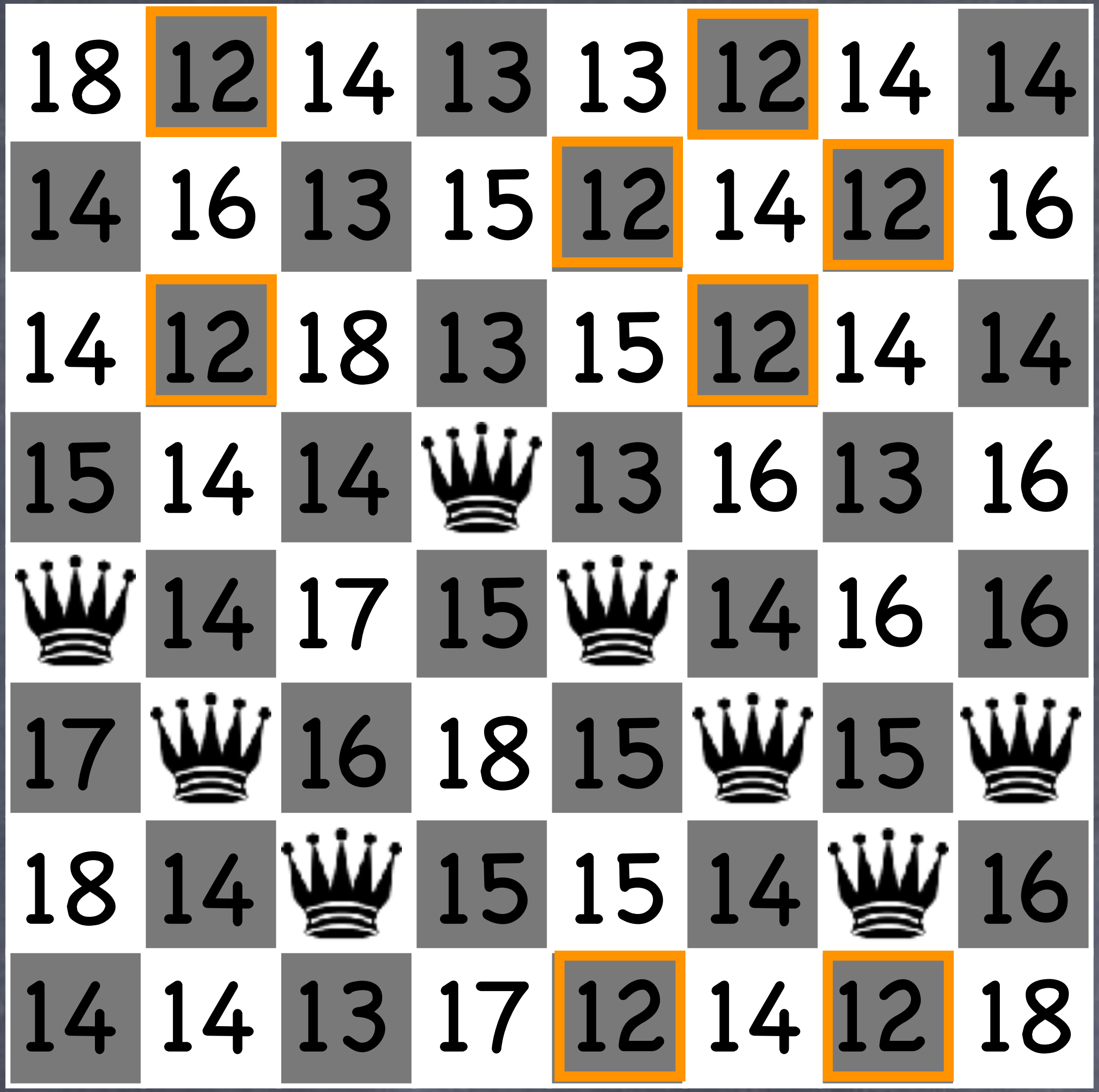
$$h(n) = 17$$



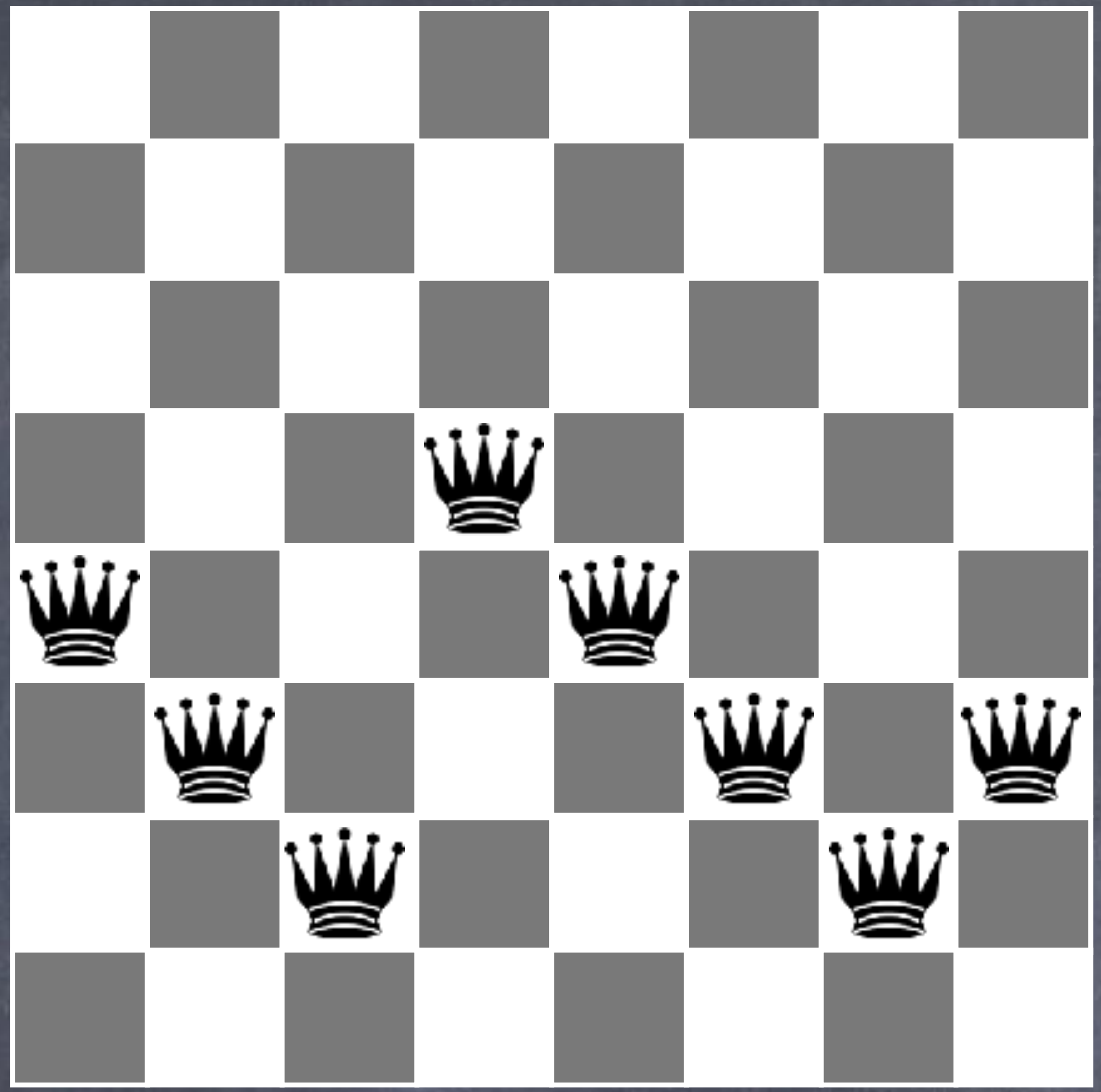
$$h(n) = 17$$

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

$$h(n) = 17$$


18 12 14 13 13 12 14 14
14 16 13 15 12 14 12 16
14 12 18 13 15 12 14 14
15 14 14 13 16 13 16
14 17 15 14 16 16
17 16 18 15 15
18 14 15 15 14 16
14 14 13 17 12 14 12 18

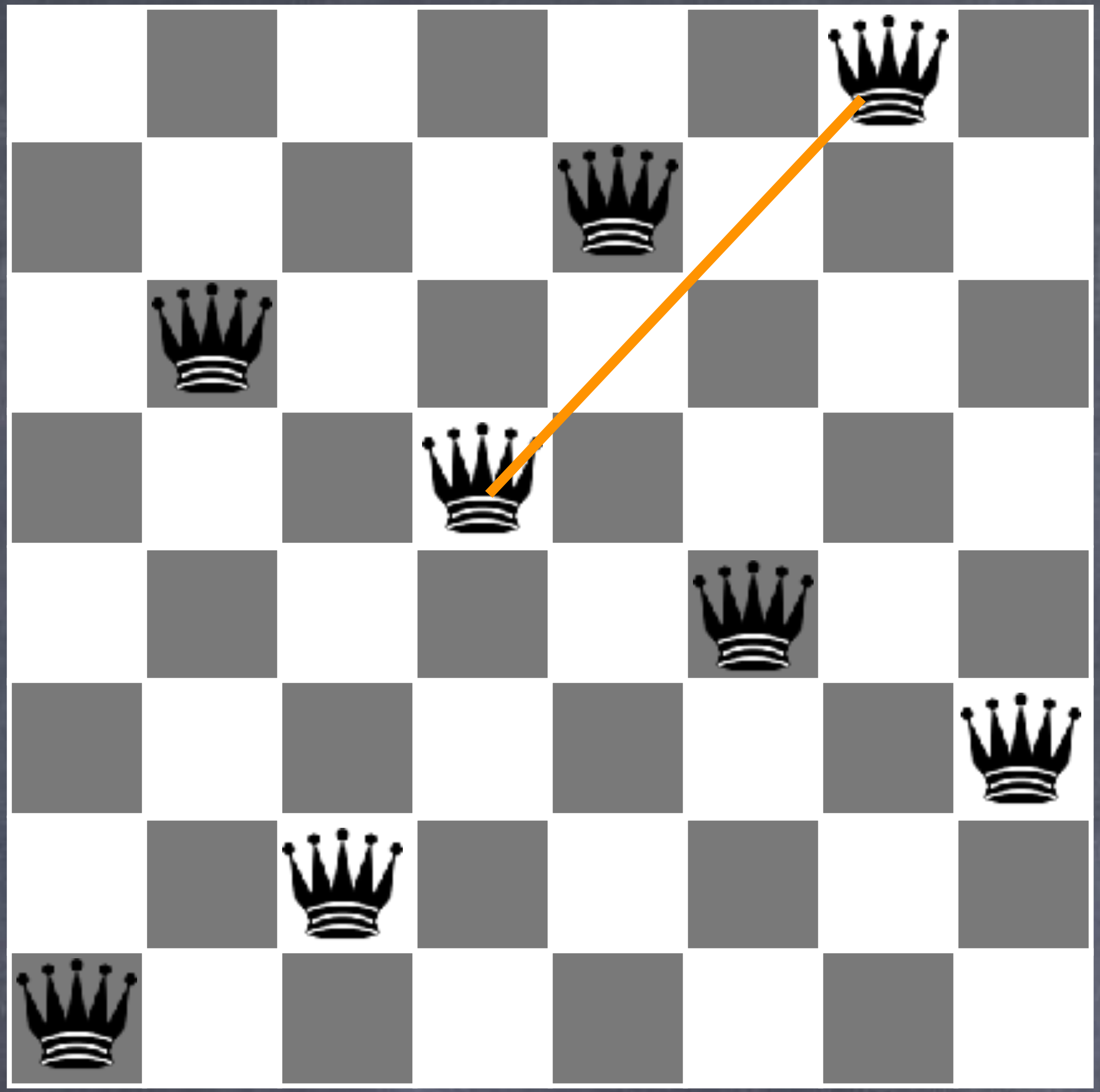
$$h(n) = 17$$



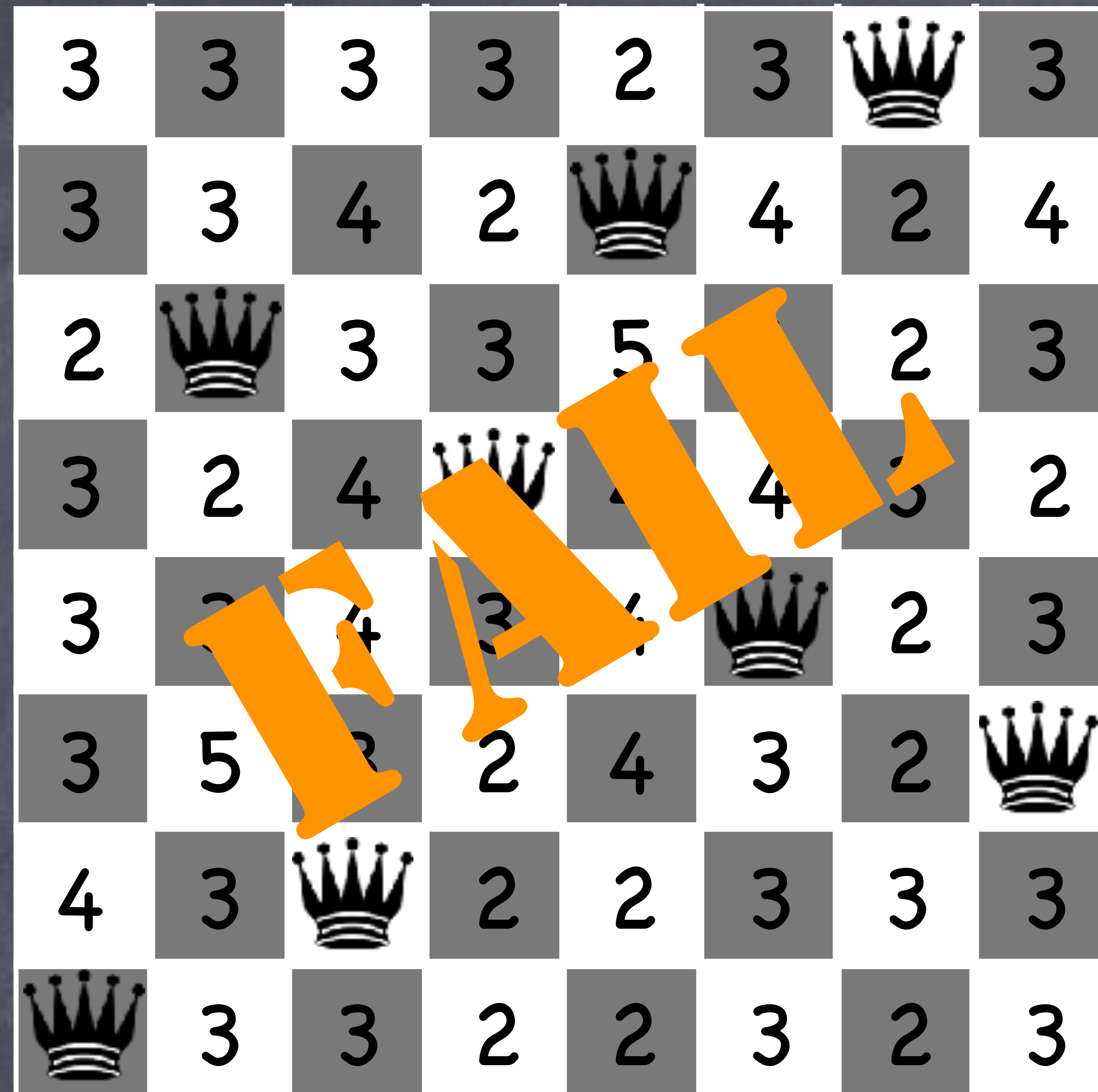
$$h(n) = 12$$

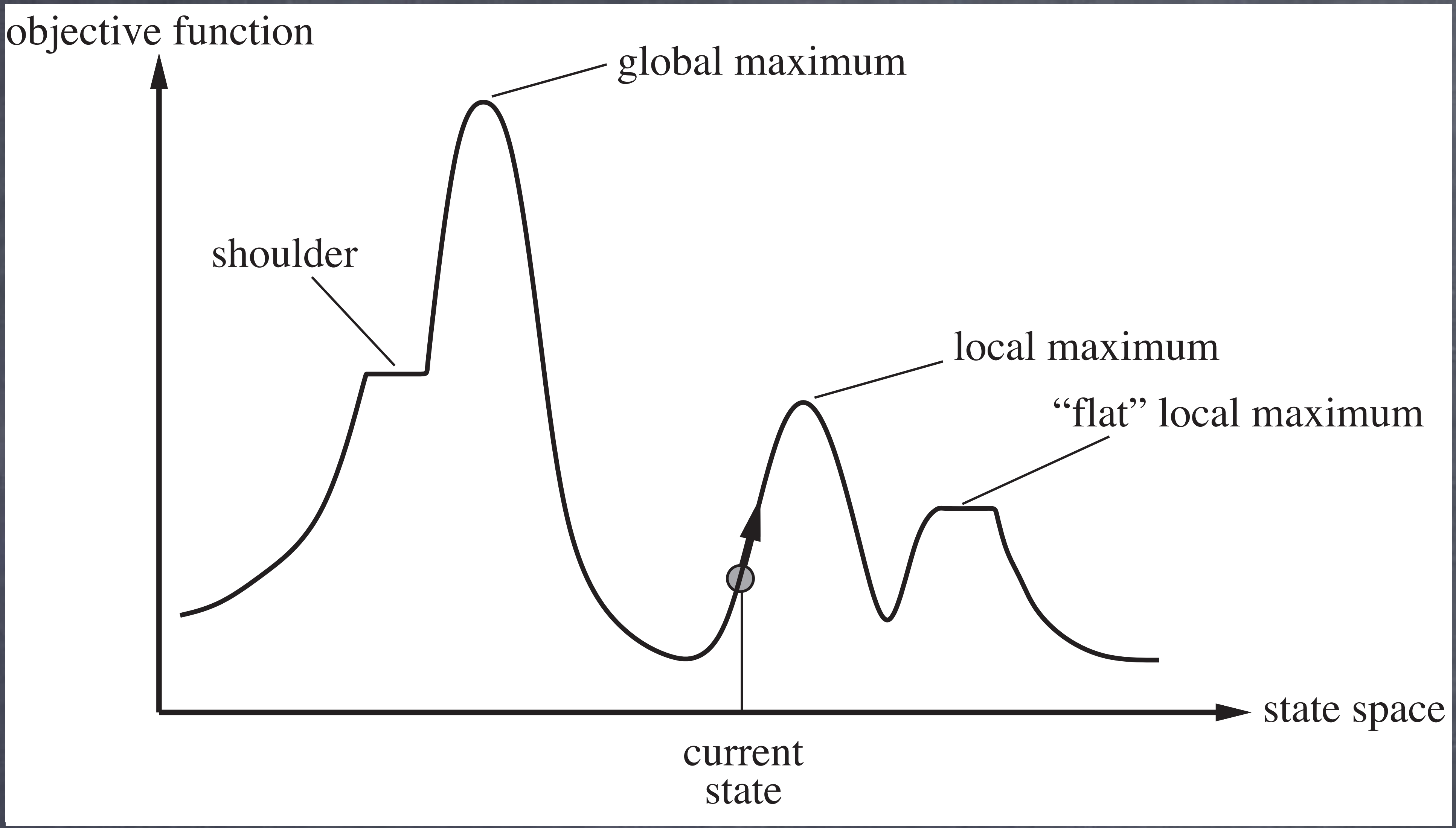
Greedy Local Search (aka Hillclimbing)

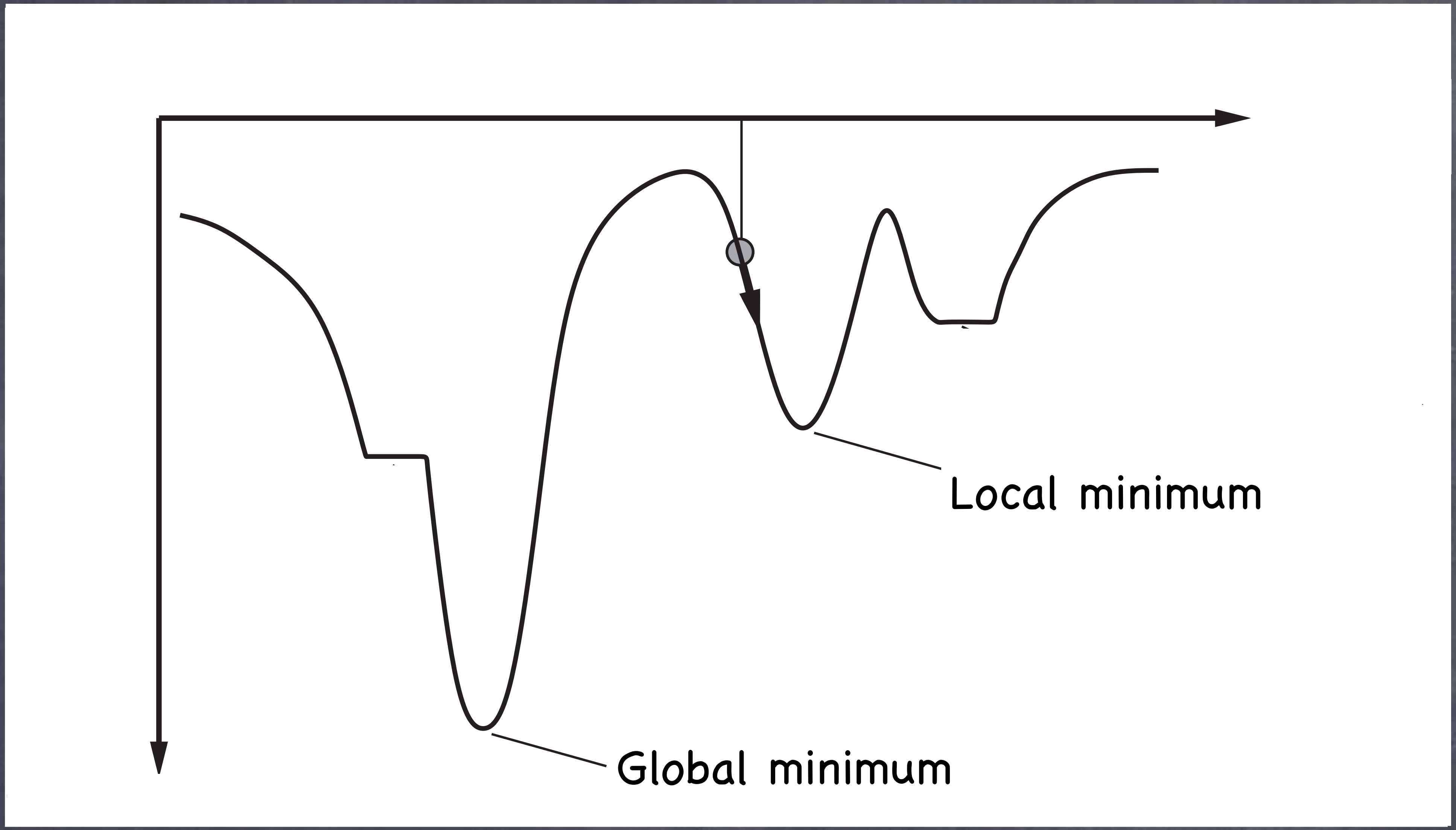
```
State hillClimb(Problem p) {
   Node node = new Node(p.getInitialState());
    while (true) {
       Node next = null;
        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(node)) {
                node = n;
            }
        }
        if (next == null) {
            return node.getState();
        } else {
            node = next;
        }
   }
}
```



$$h(n) = 1$$


FAIL


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state



Local minimum
Global minimum

Greedy Local Search (aka Hillclimbing)

- Stores only one current state
- Follows objective function towards state with maximum value (minimum cost)
- Can get stuck in local maxima (minima)

If at first you don't succeed, try again.

Random Restart Strategy

```
State randomRestart(Problem p) {
  while (true) {
    p.setInitialState(new random State);
    State solution = hillClimb(p);
    if (p.isGoal(solution)) {
      return solution;
    }
  }
}
```

Does it work?   Yes (but)

How well does it work?

Prob of success = p Expected # of tries = $1/p$

= 0.14 ≈ 7

```
State randomRestart(Problem p) {
  while (true) {
    p.setInitialState(new random State);
    State solution = hillClimb(p);
    if (p.isGoal(solution)) {
      return solution;
    }
  }
}
```

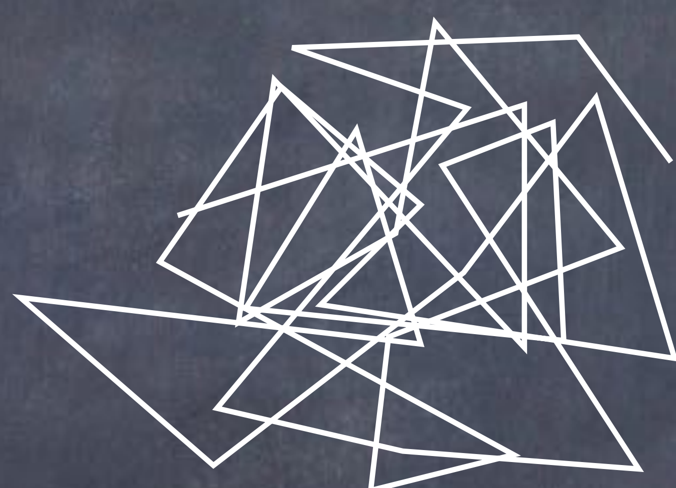
Randomness

Stochastic hill climbing

(see AIMA)

Randomness in Search

Pure random walk



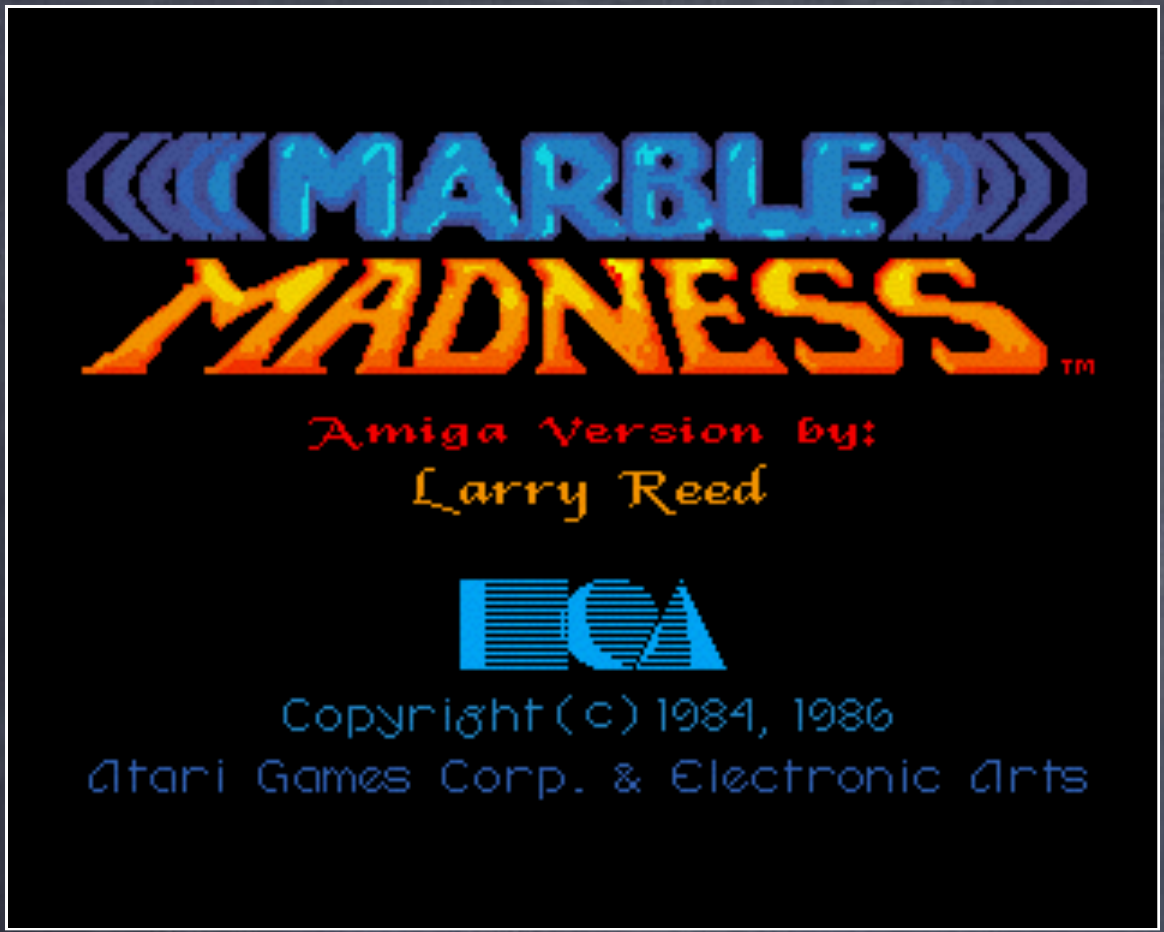
Complete,

but horribly slow

Greedy local search

Incomplete,

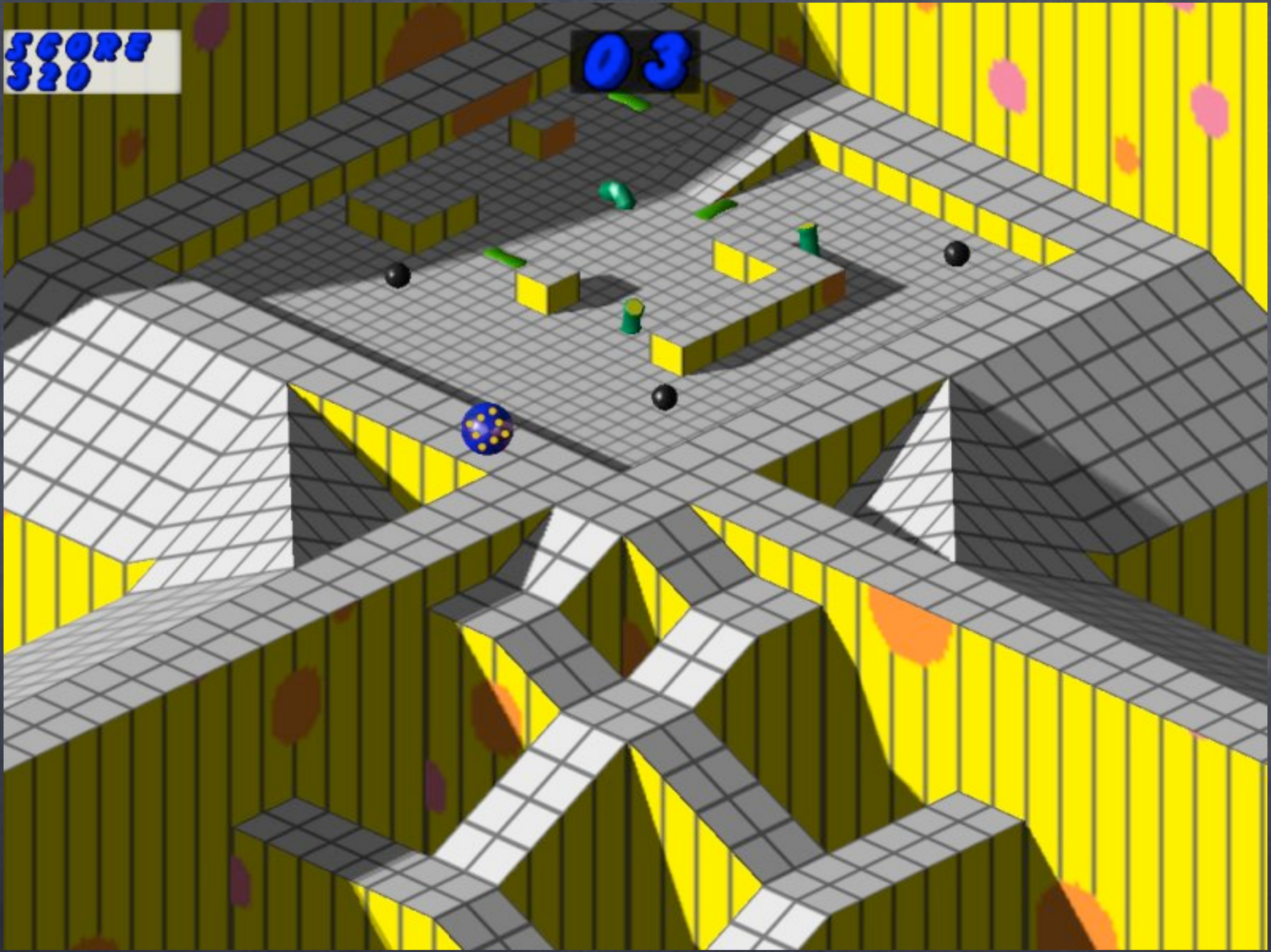
but fast

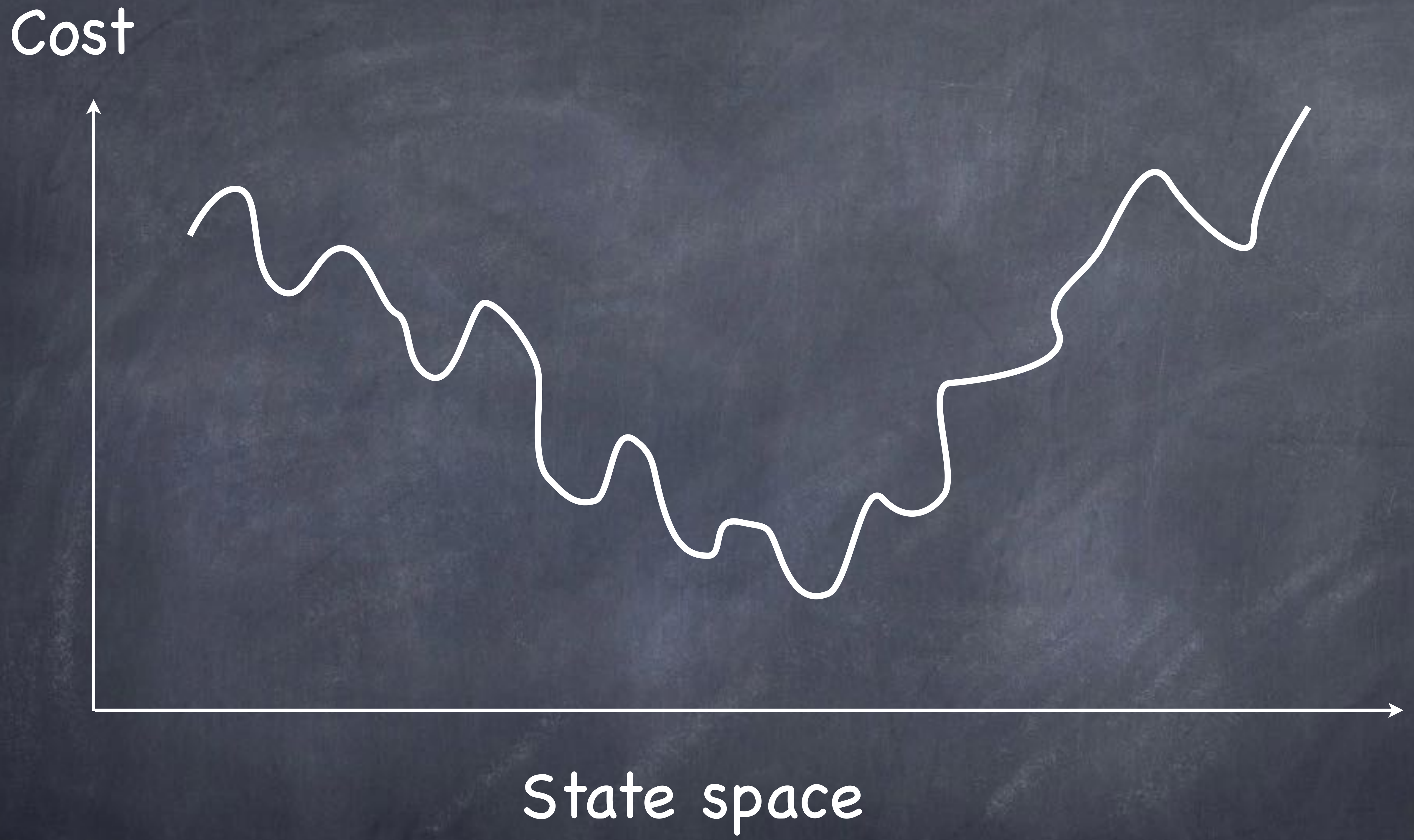

MARBLE
MADNESS™
Amiga Version by:
Larry Reed
ECA
Copyright (c) 1984, 1986
Atari Games Corp. & Electronic Arts


SCORE
55

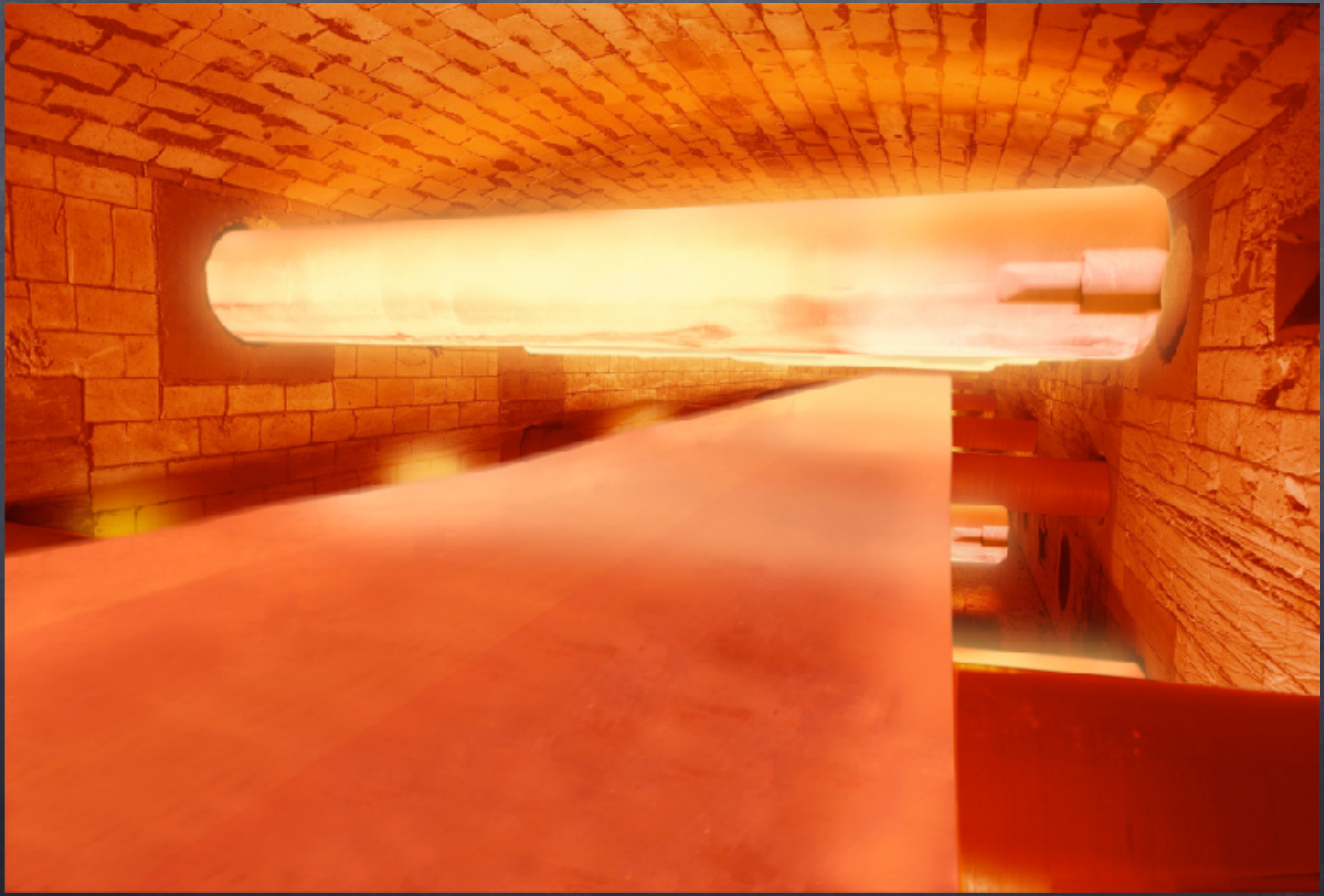

SCORE
55


MARBLE MADNESS
ATARI


SCORE
320
03


Cost
State space

Annealing



Annealing

anneal |əˈnēl|
verb [with obj.]
heat (metal or glass) and allow it to cool slowly, in order to remove internal stresses and toughen it.

ORIGIN Old English *onǣlan*, from *on* + *ǣlan* ***'burn, bake,'*** from *āl* ***'fire, burning'*** The original sense was ***'set on fire,'*** hence (in late Middle English) ***'subject to fire, alter by heating'***

Simulated Annealing

- Greedy local search (Hill-descending)
- Select states with lower cost
  - OR with some probability even if higher cost
- “High temperature”: higher probability
- “Low temperature”: lower probability
- “Cool” according to schedule

Simulated Annealing

```
State simulatedAnnealing(Problem p, Schedule schedule) {
  Node node = new Node(p.getInitialState());
  for (t=1; true; t++) {
    Number T = schedule(t);
    if (T == 0) {
      return node;
    }
    Node next = randomly selected successor of node
    Number deltaE = p.cost(node) - p.cost(next);
    if (deltaE > 0 || Math.exp(-deltaE/T) > new Random(1)) {
      node = next;
    }
  }
}
```

"with probability $e^{\frac{\Delta E}{T}}$"

Simulated Annealing

Complete? No.

Optimal? No.

"But if the schedule lowers T slowly enough, simulated annealing will find a global minimum with probability approaching one."

Local Search

- Evaluates and modifies a small number of current states
- Does not record history of search

Good: Very little (constant) memory

Bad: May not explore all alternatives

=> Incomplete

For next time:

Chapter 4.3–4.4;
4.2, 4.5 FYI