CSC242: Artificial Intelligence

Lecture 4
Local Search

Upper Level Writing

- Topics due to me by next class!
- First draft due Mar 4
- Goal: final paper 15 pages +/- 2 pages
  - 12 pt font, 1.5 line spacing
- Get in touch early!

Assignments

- Homework 1
  - Will be posted on BB tonight
  - Solutions will be given out (hardcopy) in class Tuesday
- Project 1: Othello
  - Will be posted Tuesday night

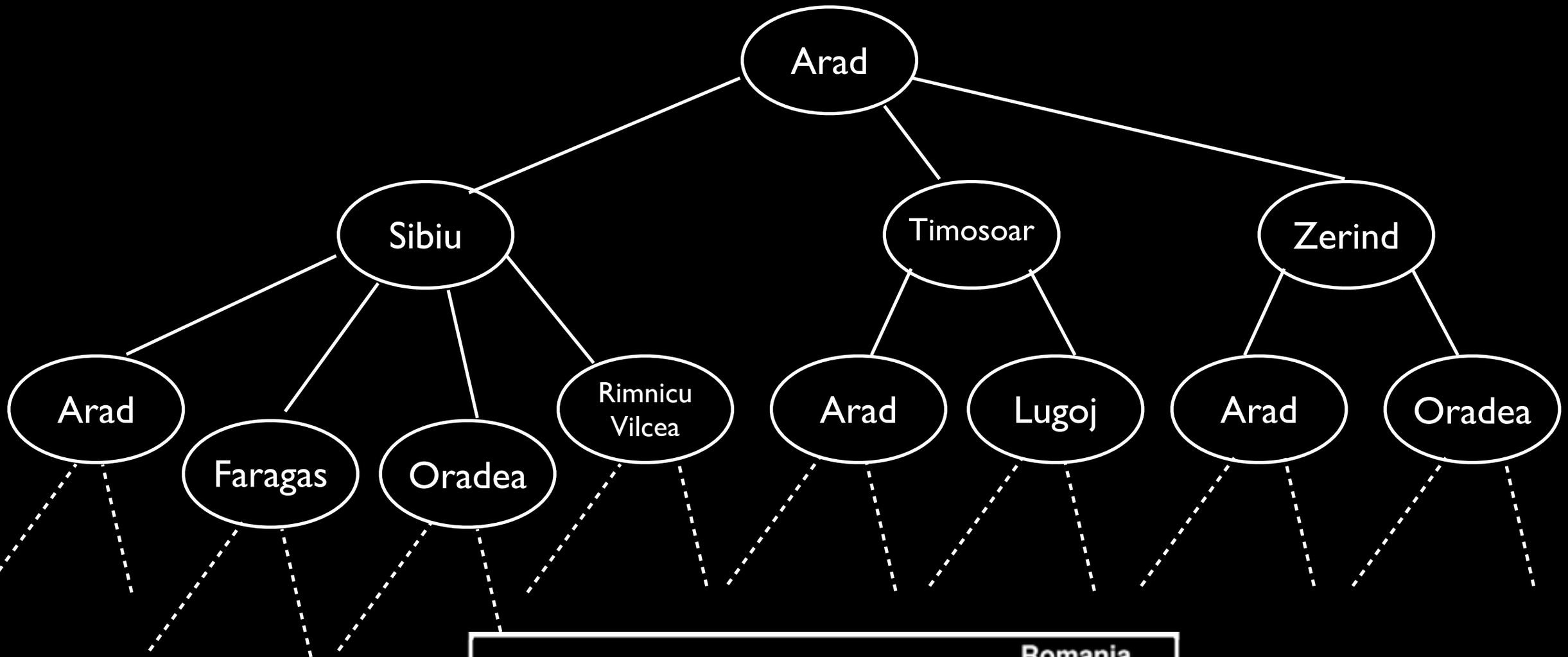
Local Search

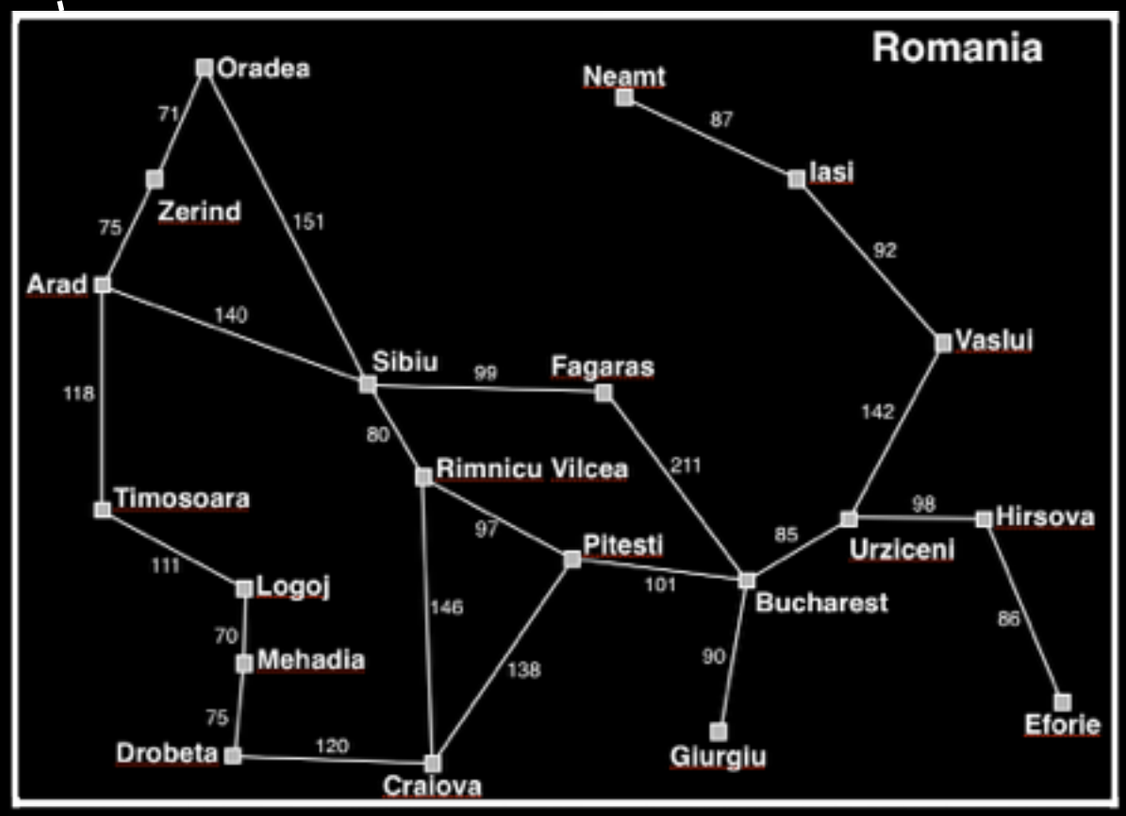
States + Actions + Transition Model

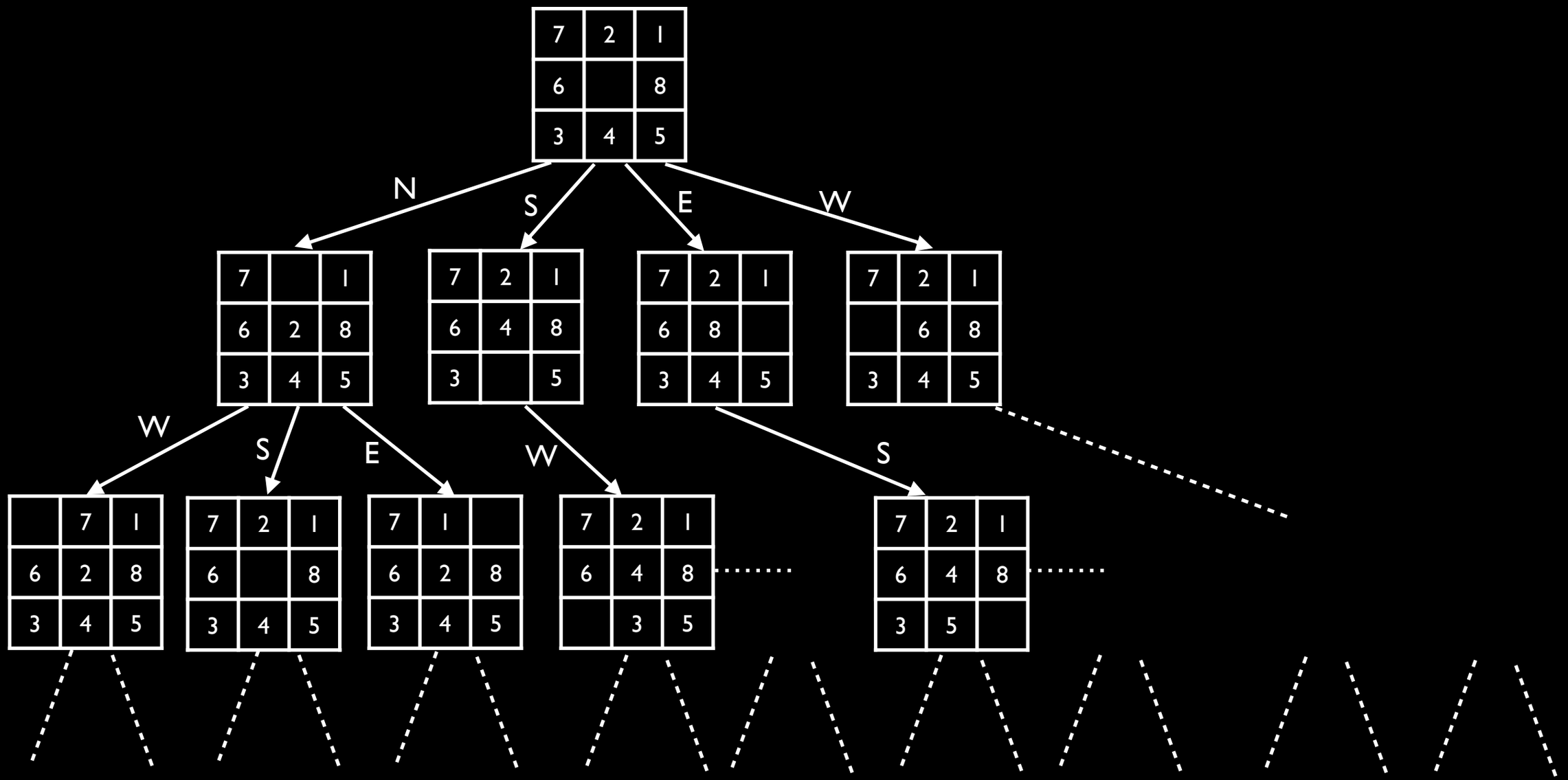
=

State Space

The set of all states reachable from the initial state by some sequence of actions


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


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


7 2 1
6 8
3 4 5
N
S
E
W
7 1
6 2 8
3 4 5
7 2 1
6 4 8
3 5
7 6 8
7 1
6 2 8
3 4 5
7 2 1
6 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 4 8
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

```
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);
      }
    }
  }
}
```

```
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);
      }
    }
  }
}
```

Search Strategies

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

* If step costs are identical

† With an admissible heuristic

Systematic Search

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

Systematic Search

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

Good:

Systematic Search

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

Good: Systematic → Exhaustive

Systematic Search

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

Good: Systematic → Exhaustive

Bad:

Systematic Search

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

Good: Systematic → Exhaustive

Bad: Exponential time and/or space

The Problem Last Class!

Java™

Download Help

HELP RESOURCES

- Installing Java
- Disable Java
- **Using Java**
- General Questions
- Mobile Java
- Security
- Support Options

Find expert help on Java installation and setup

Get Help Now!

Printable Version

Why are Java applications blocked by your security settings with the latest Java?

This article applies to:

- **Java version(s):** 7.0

SYMPTOMS

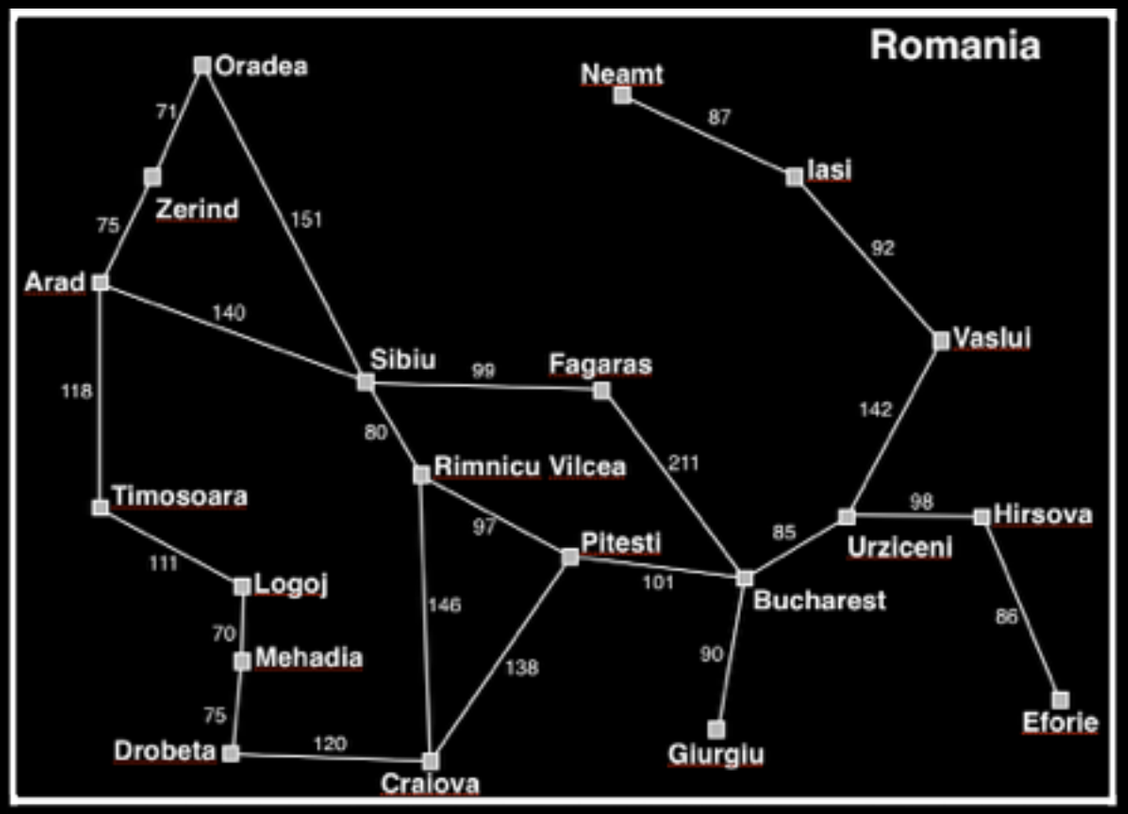
Trying to run the Java applications with Java version 7 Update 51, generates messages that says

> Java applications are blocked by your security settings.
>
> Missing Application-Name manifest attribute
>
> Missing required Permissions manifest attribute in main jar

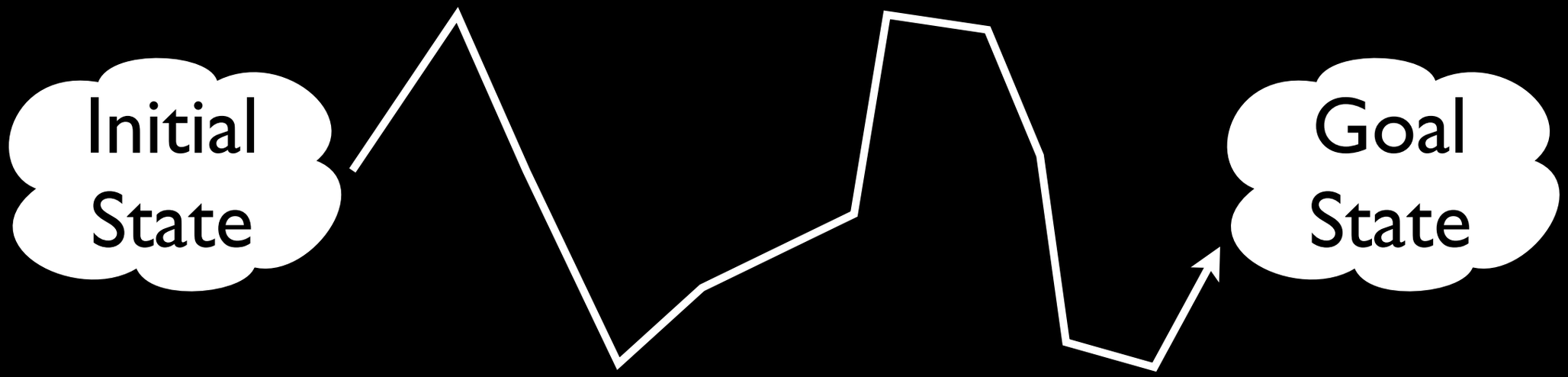
CAUSE

Starting with Java 7 Update 51, Java has enhanced security model to make user system less vulnerable to the external exploits. The new version of Java does not allow users to run the applications that are not signed (Unsigned), Self signed (not signed by trusted authority) and the applications that are missing permission attributes.

Local Search

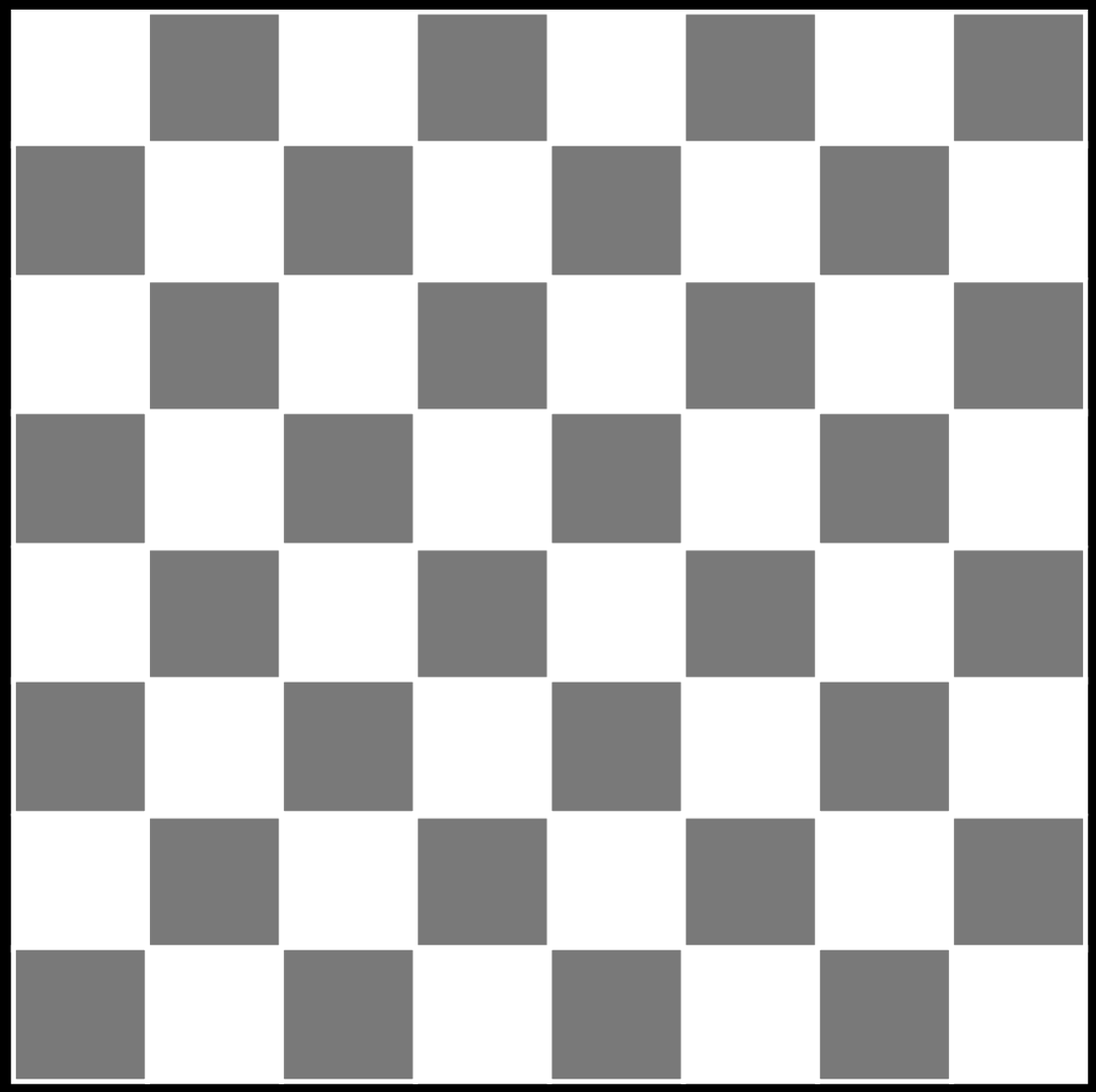

Romania
Oradea
Neamt
71
87
Iasi
Zerind
151
75
92
Arad
Vaslui
140
Sibiu
99
Fagaras
118
142
80
Rimnicu Vilcea
211
Timosoara
98
Hirsova
97
Pitesti
85
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
138
90
75
Eforie
Drobeta
120
Giurgiu
Craiova


Thinkfun
Fifteen Puzzle
Fifteen Puzzle


Initial
State
Goal
State

Initial State

Goal State



N-Queens as State-Space Search Problem

N-Queens as State-Space Search Problem

- State

N-Queens as State-Space Search Problem

- State
- Actions

N-Queens as State-Space Search Problem

- State
- Actions
- Transition Model

N-Queens as State-Space Search Problem

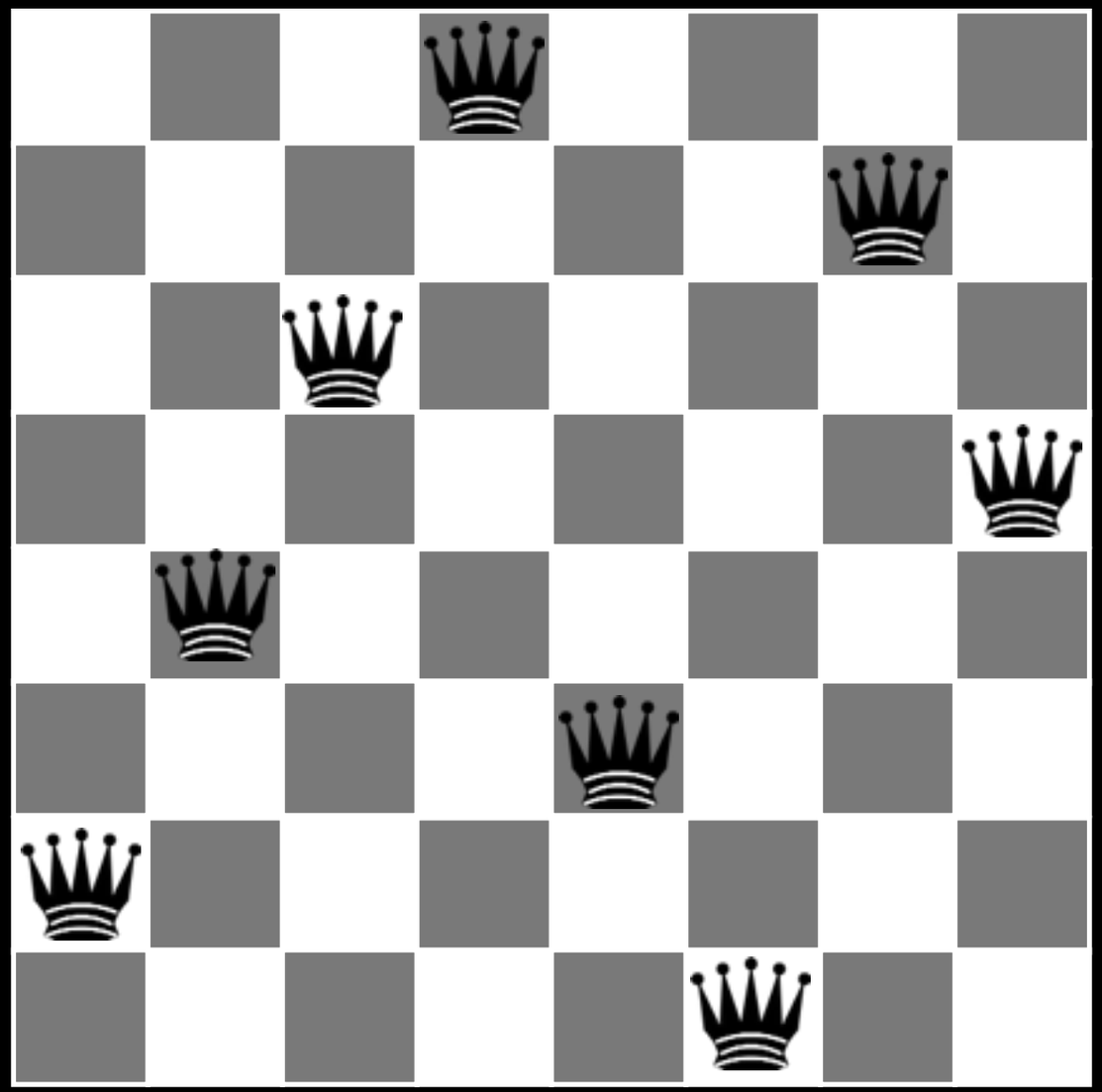
- State
- Actions
- Transition Model
- Initial State

N-Queens as State-Space Search Problem

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

N-Queens as State-Space Search Problem

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



Local Search

Local Search

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

Local Search

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

Good:

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

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:

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

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()) {



    }
  }
}
```

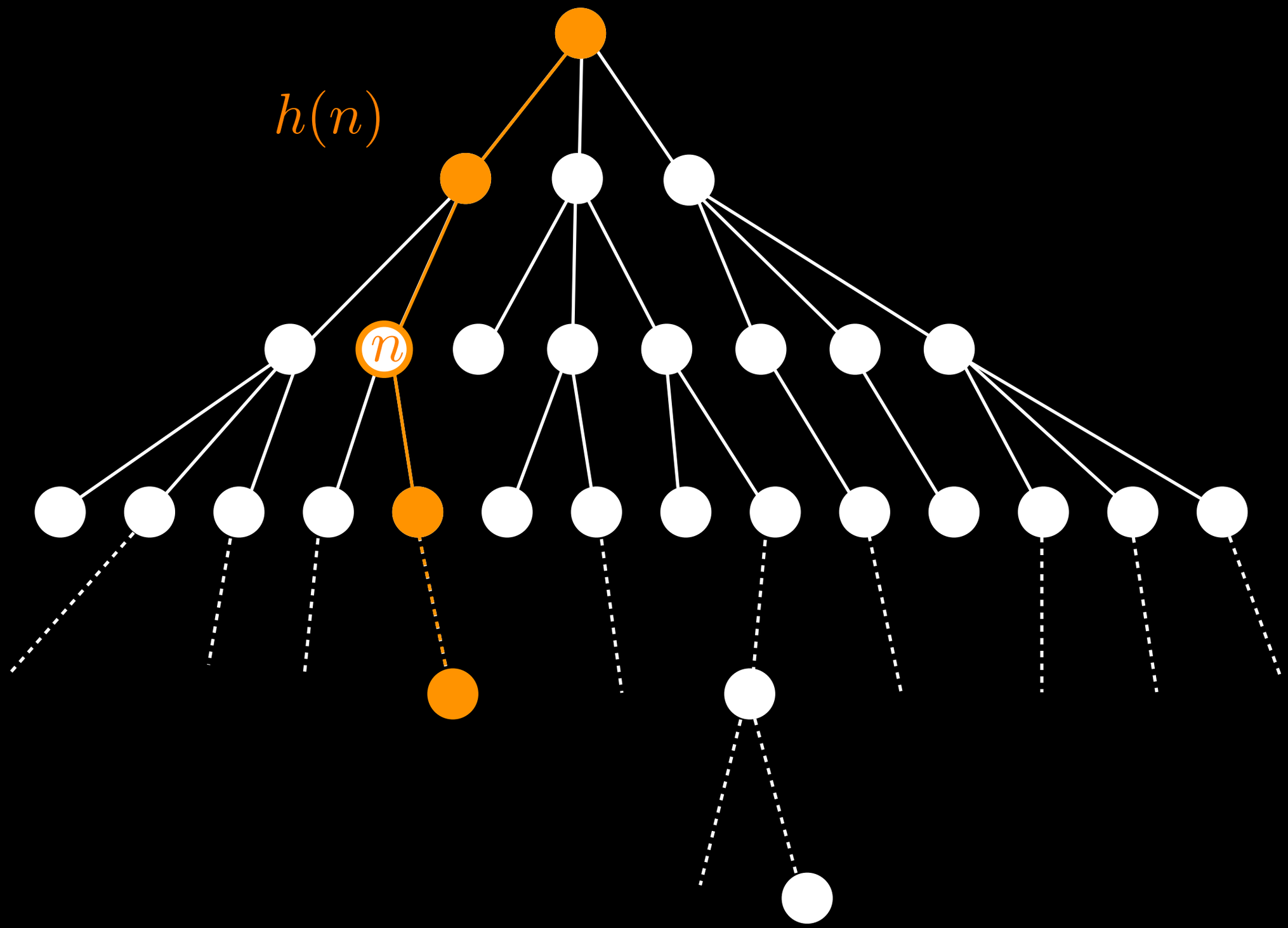
```
State localSearch(Problem p) {
  Node node = new Node(p.getInitialState());
  while (true) {


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

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

      ???

    }
  }
}
```


$h(n)$
n

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

Hill-climbing Search

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

Hill-climbing Search

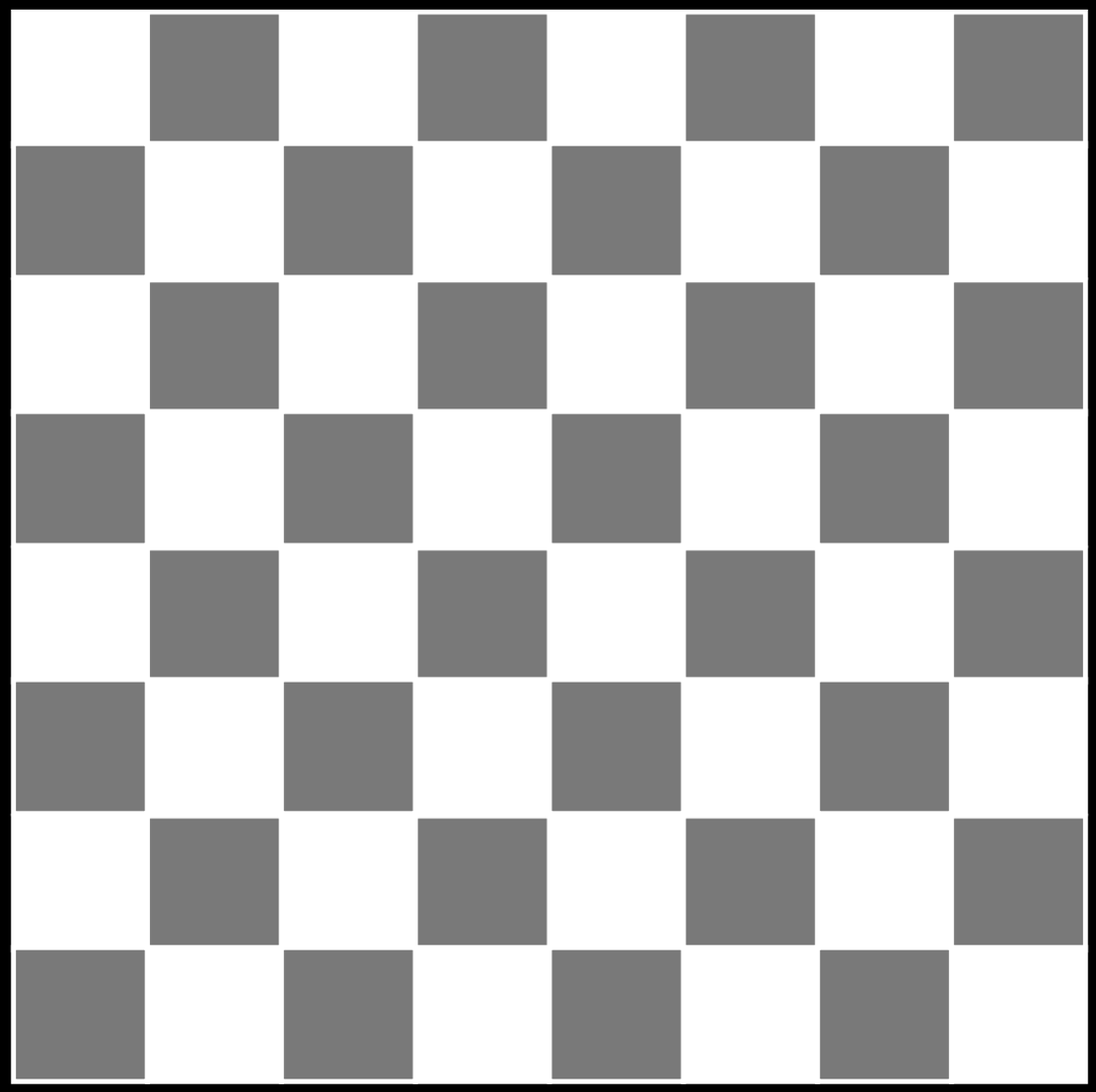
- Move through state space in the direction of increasing value ("uphill")

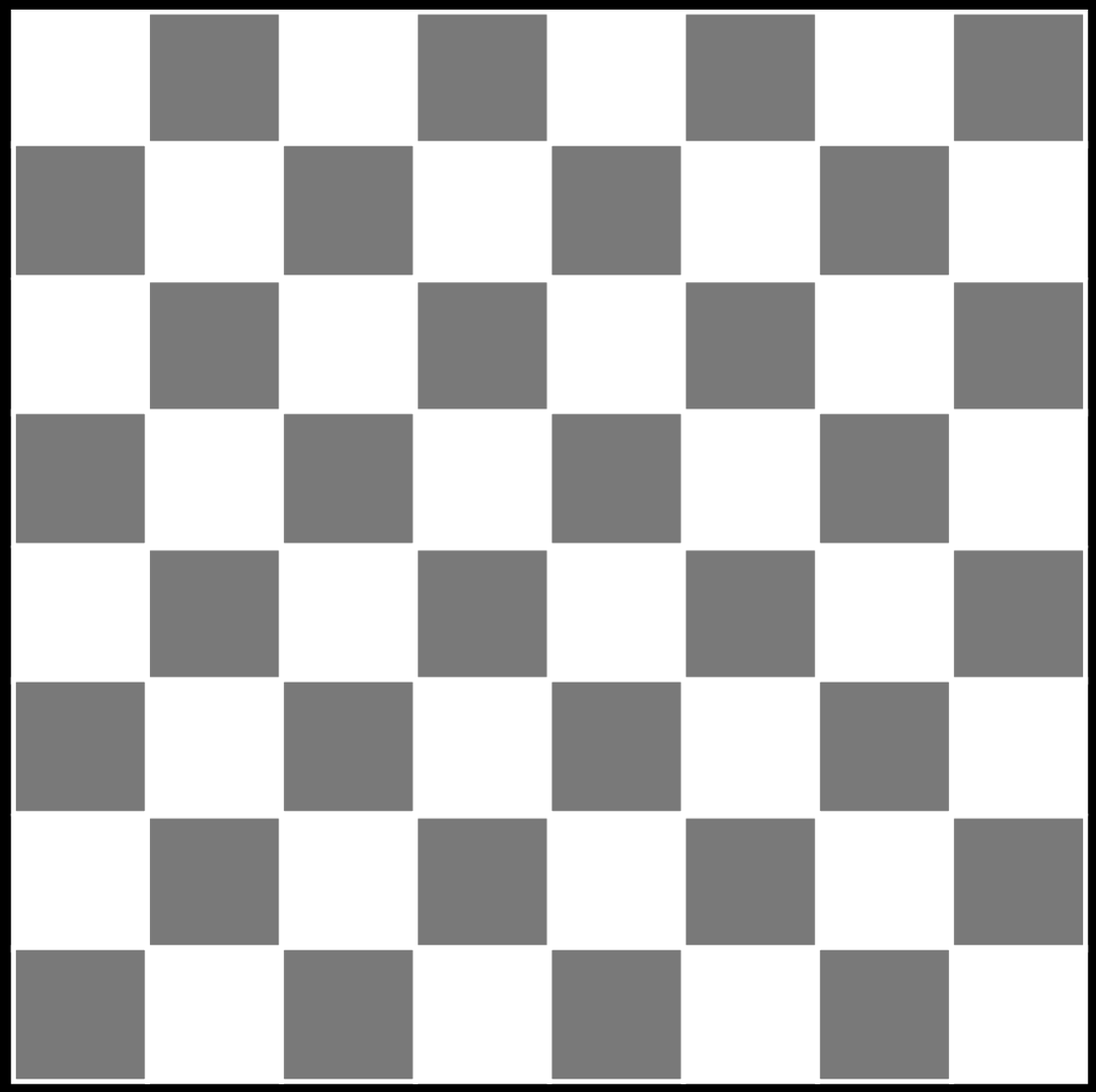
State-space landscape

Hill-climbing Search

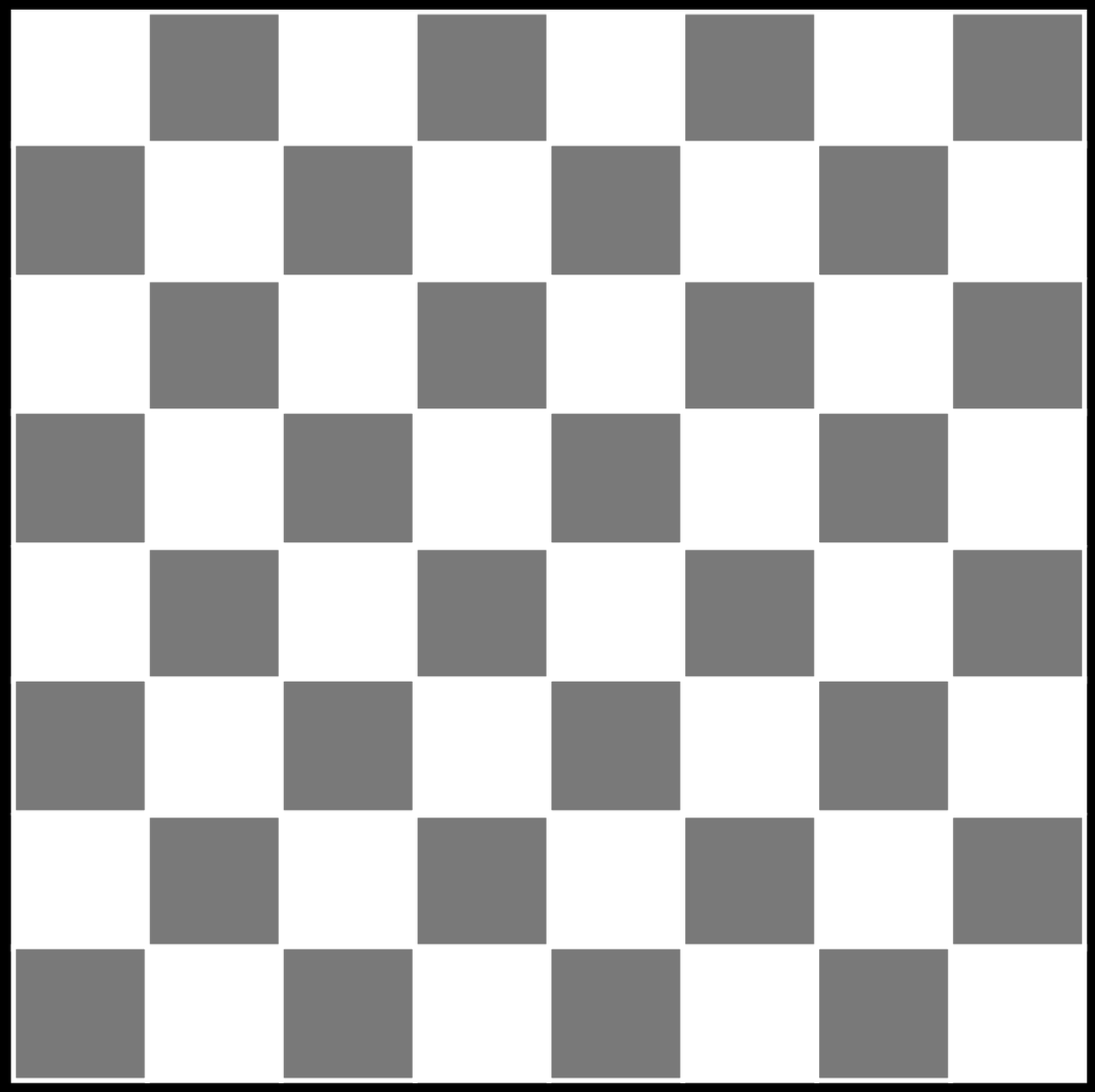
- Move through state space in the direction of increasing value ("uphill")

State-space landscape

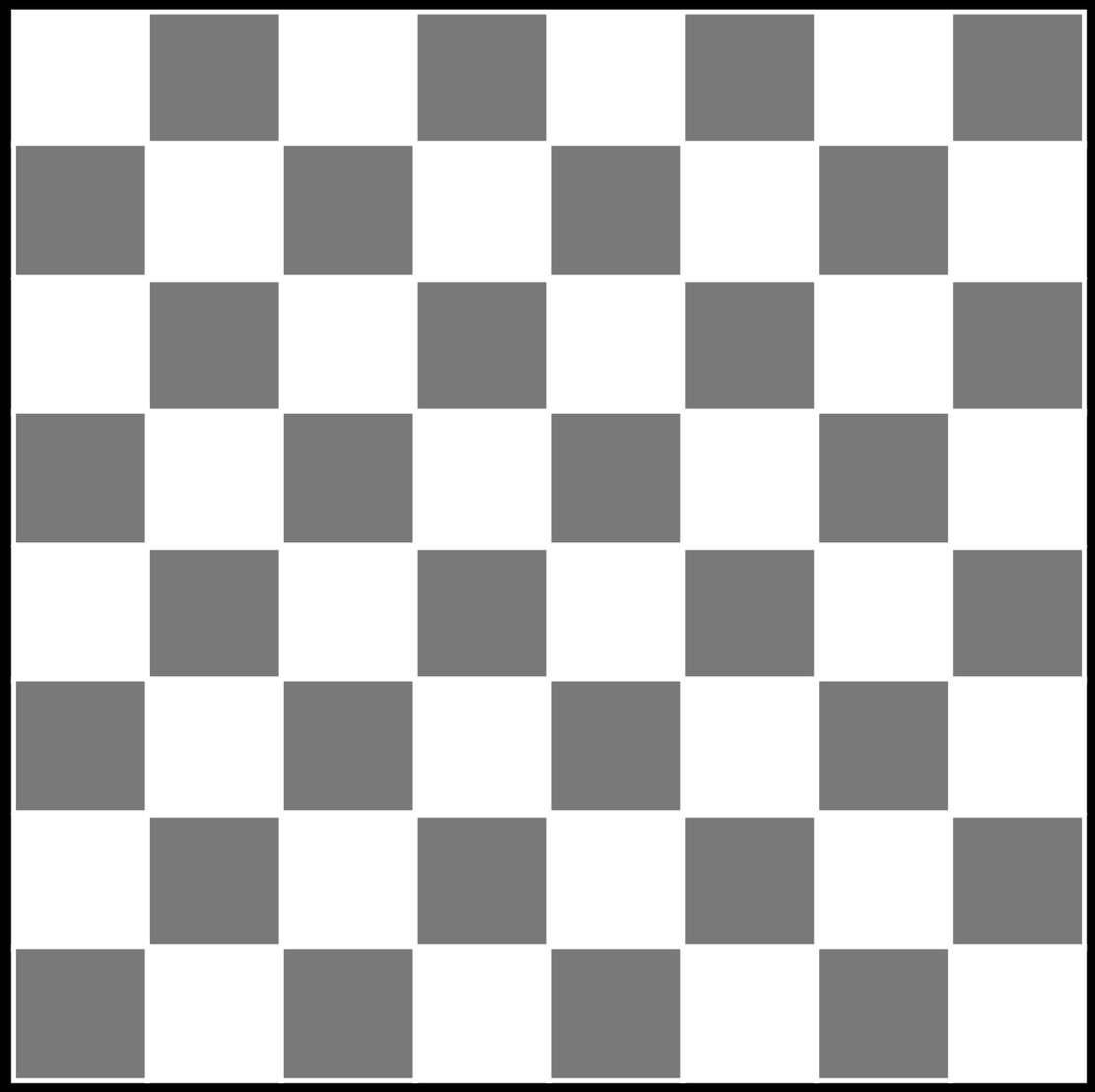




State: $[r_0, \ldots, r_7]$

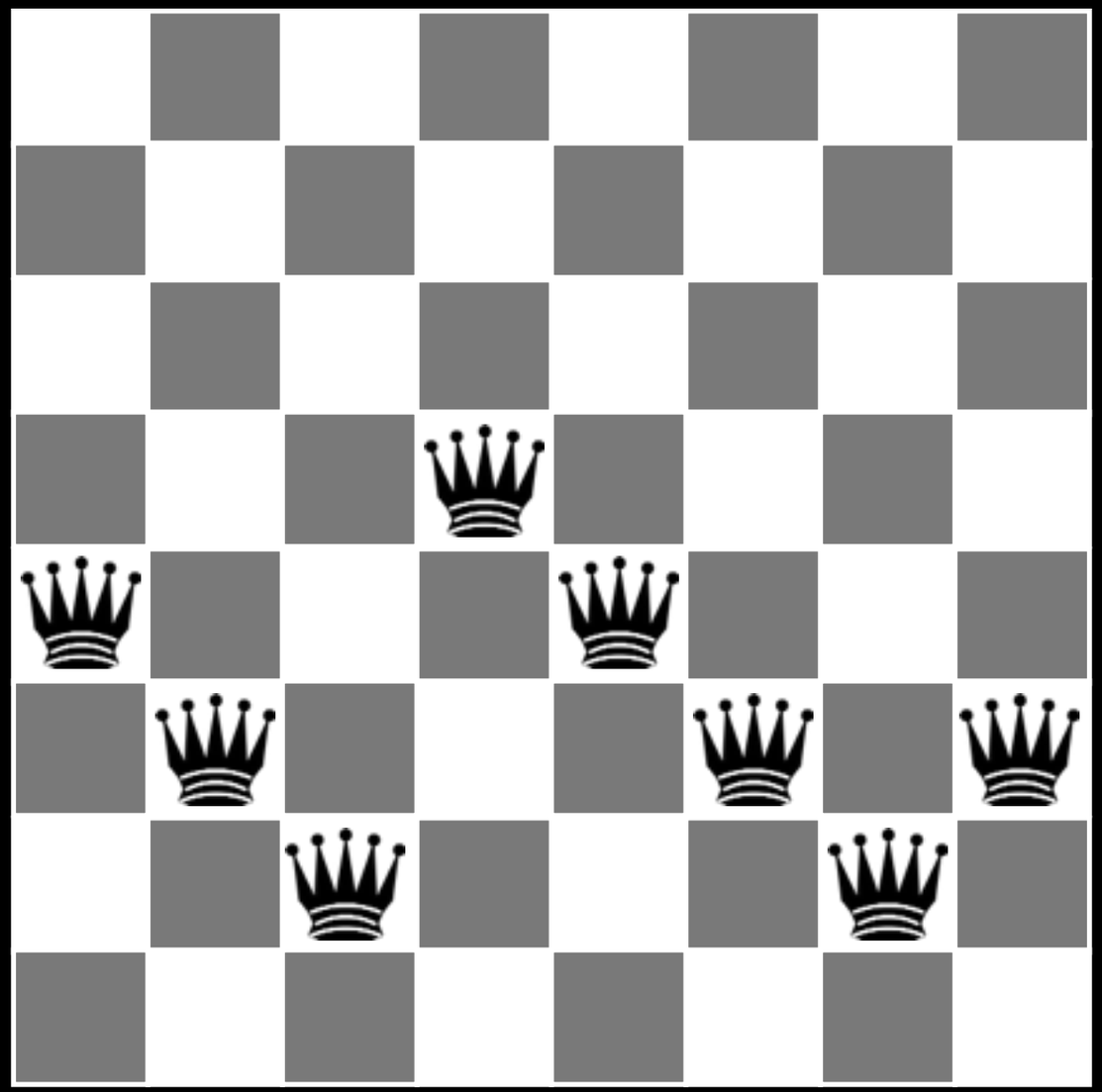


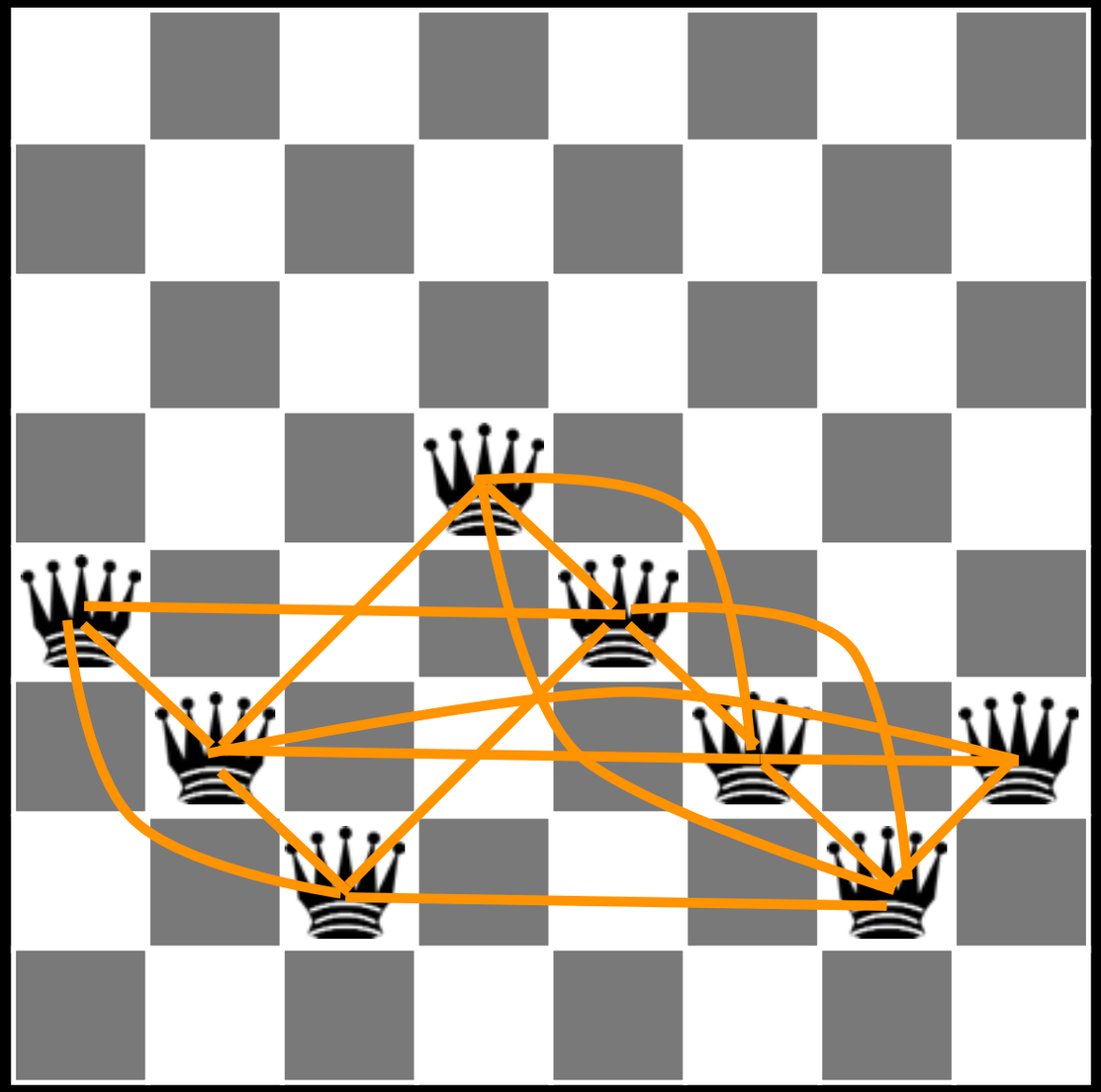
State: $[r_0, \ldots, r_7]$ Action: $< i, r_i >$



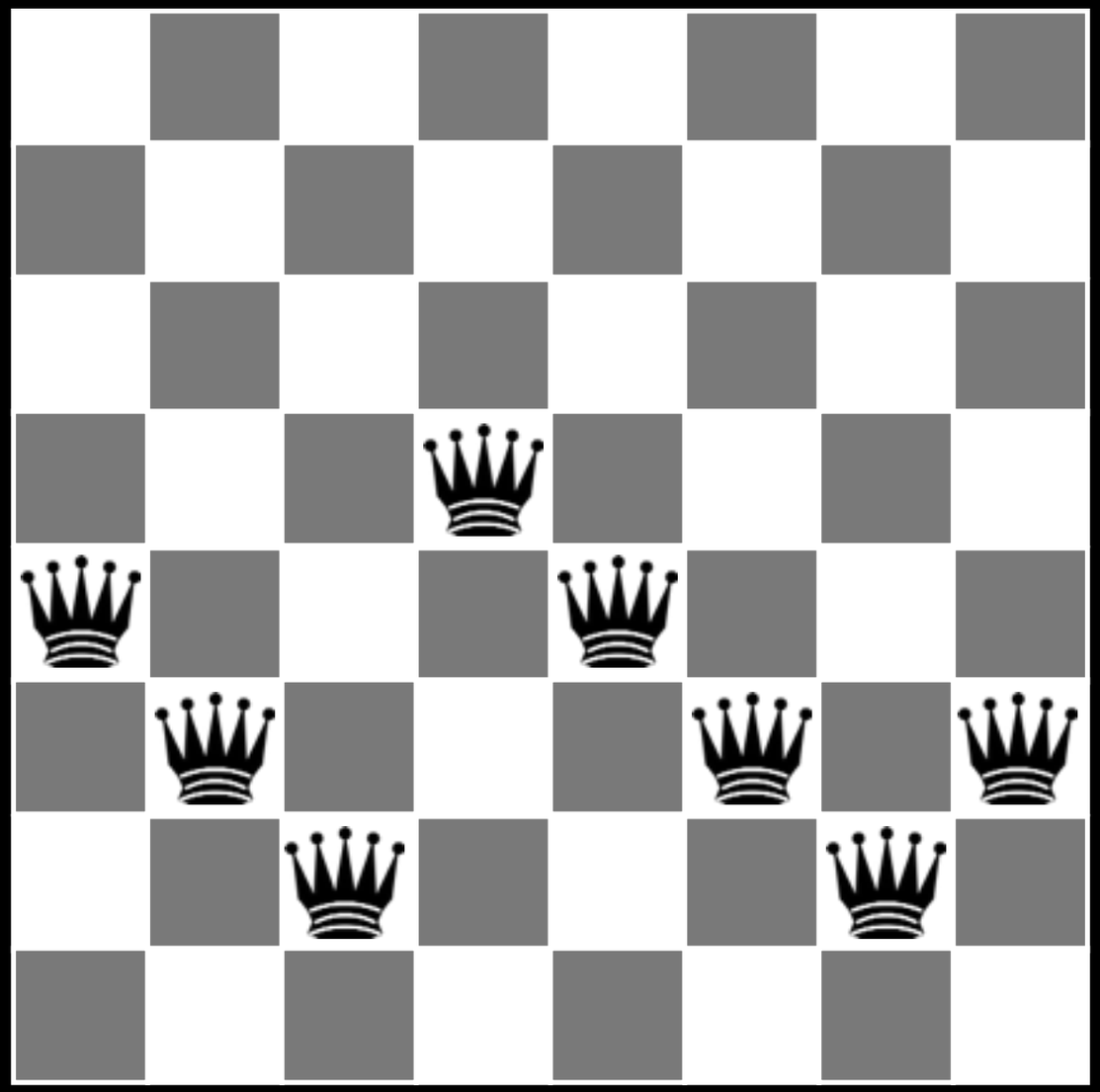
State: $[r_0, \ldots, r_7]$ Action: $< i, r_i >$

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





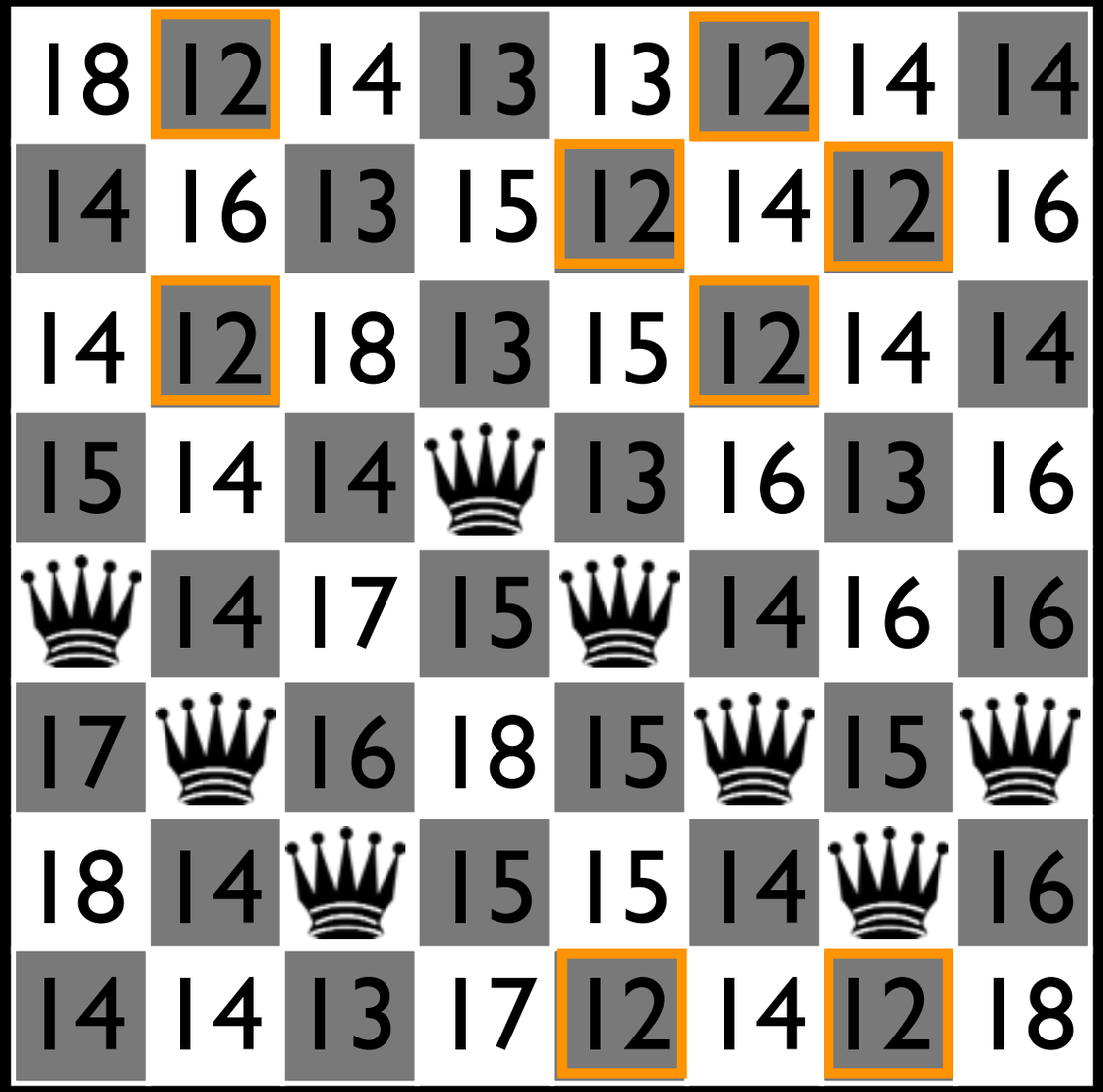
$$h(n) = 17$$



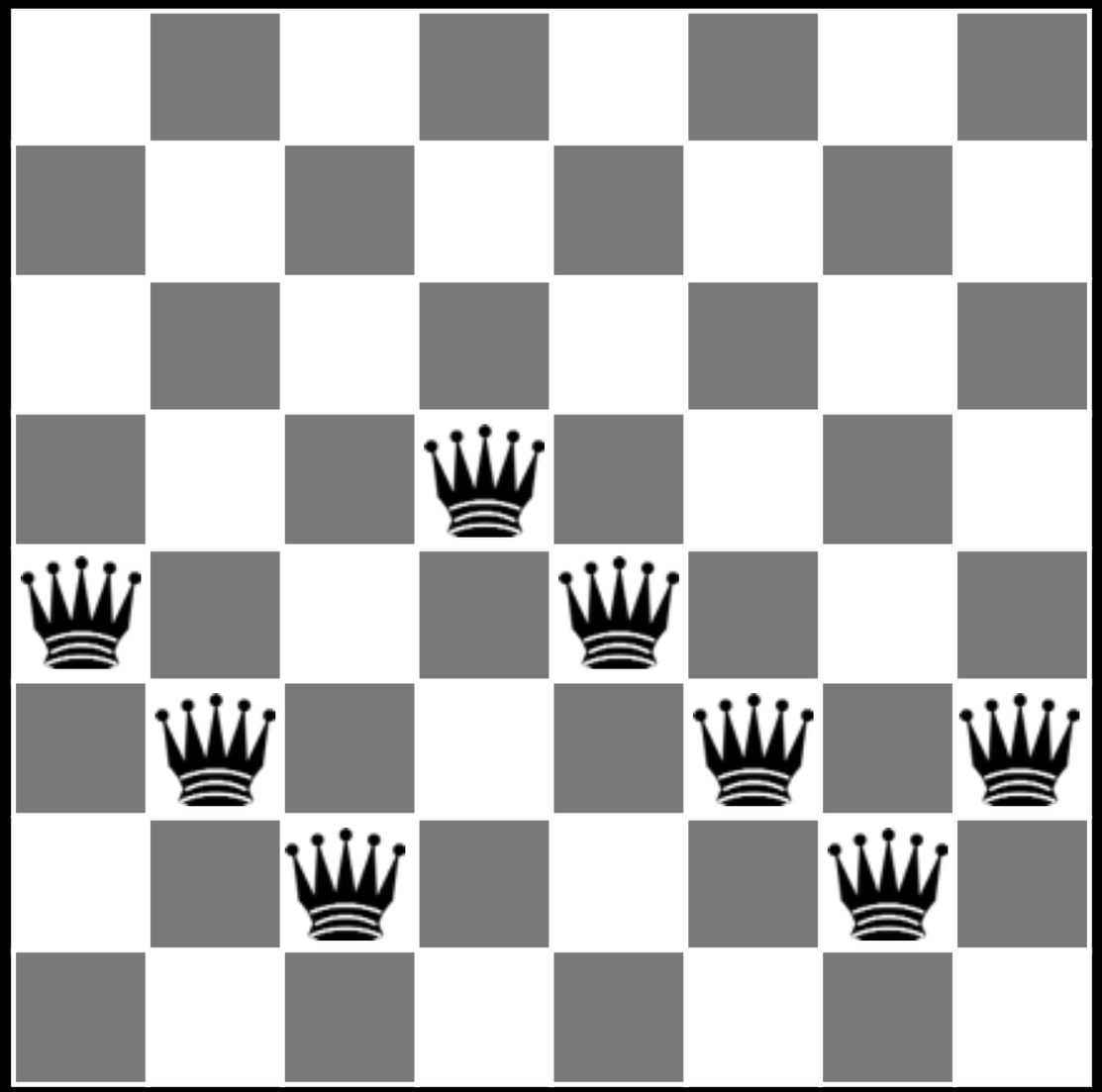
$$h(n) = 17$$

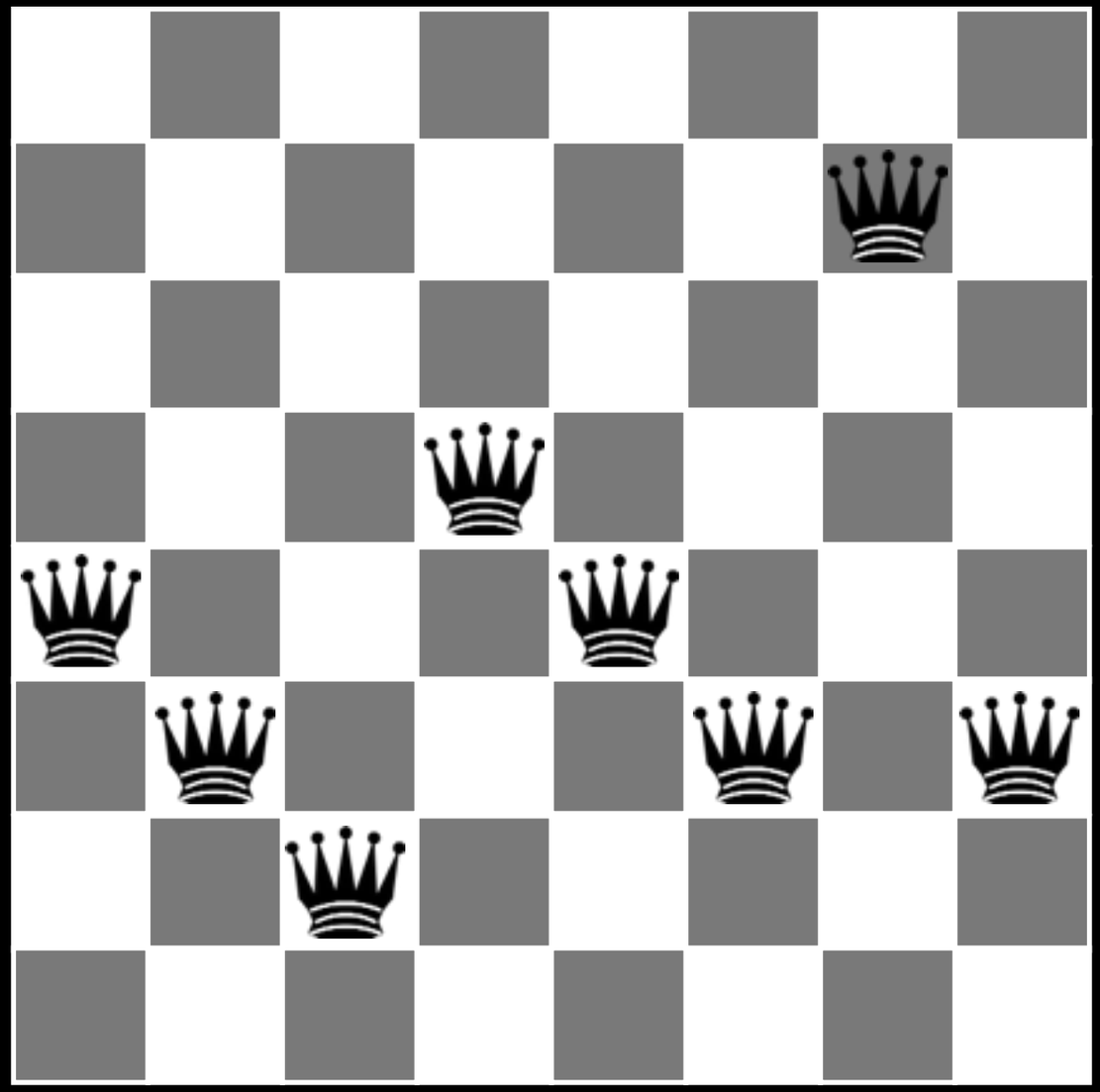
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

$$h(n) = 17$$


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

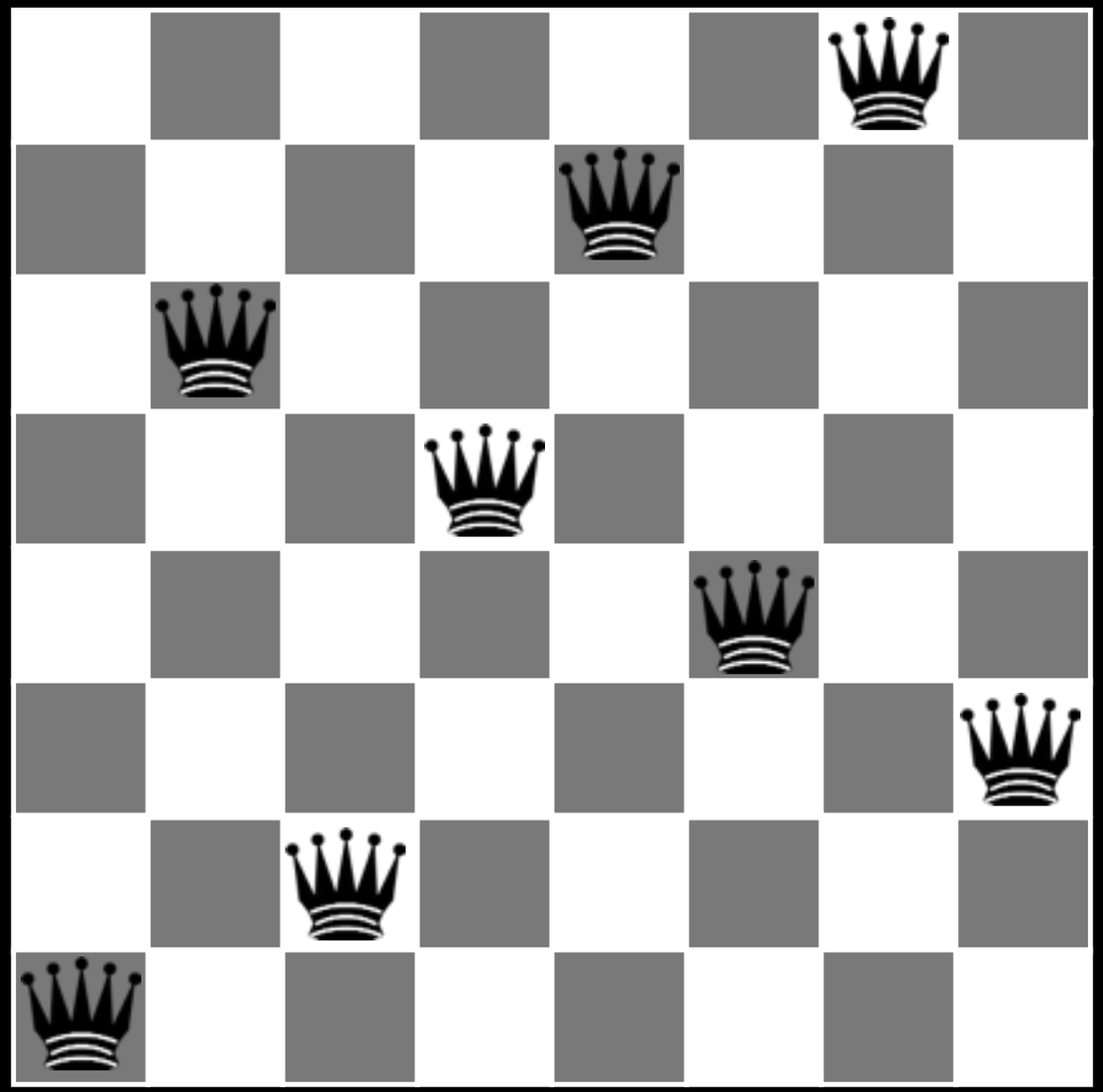
$$h(n) = 17$$

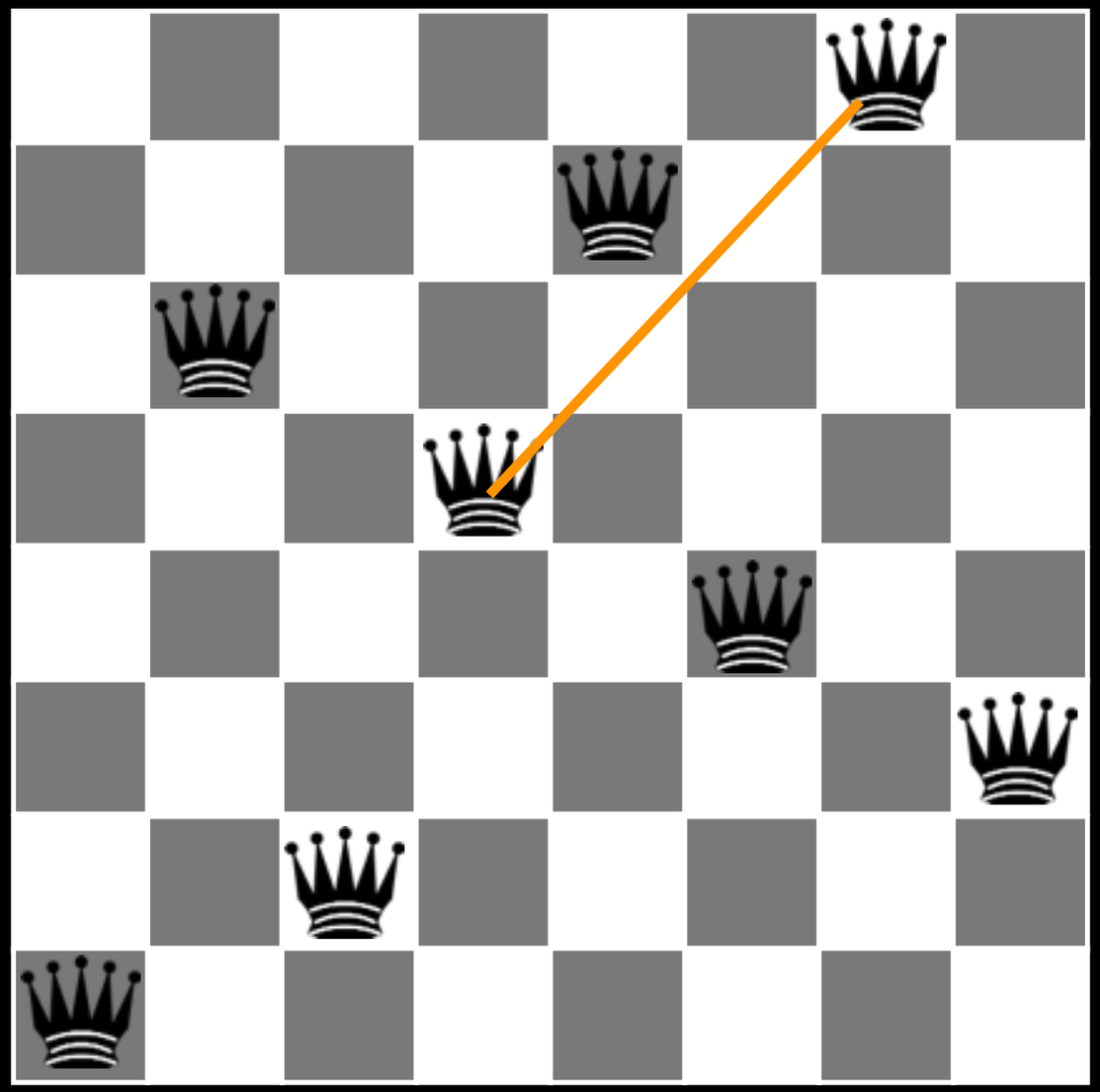




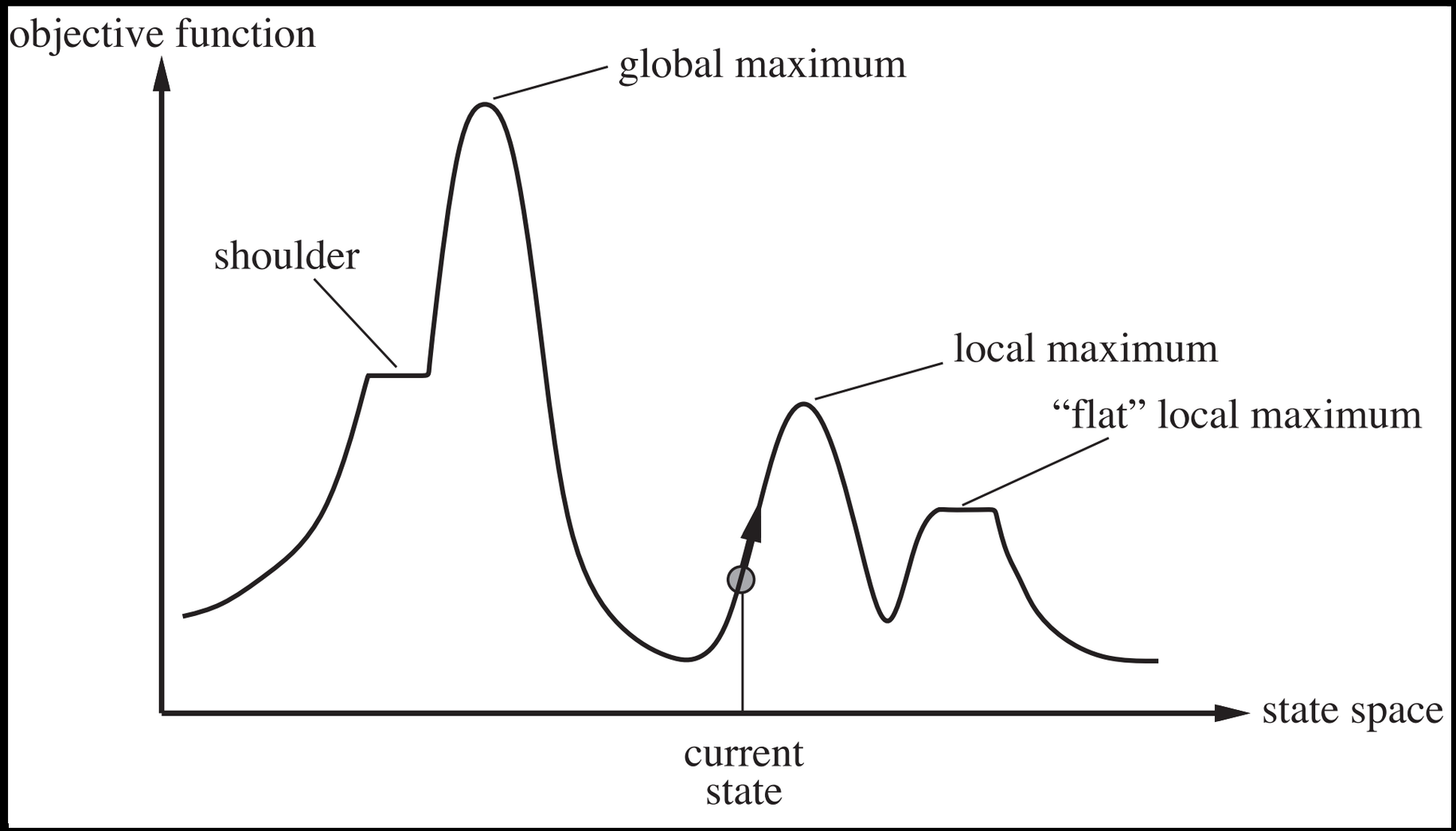
$$h(n) = 12$$

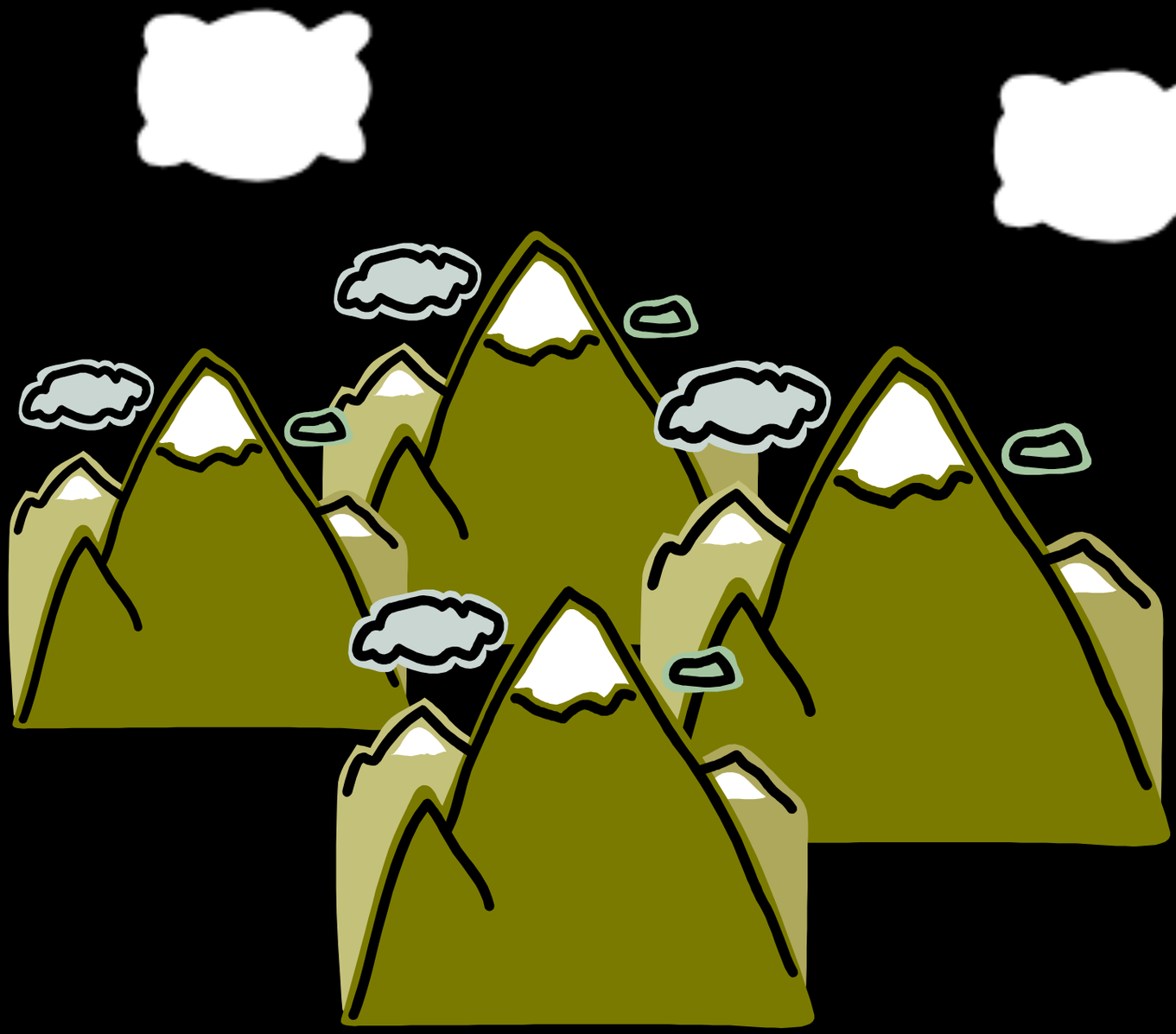
```
State hillClimb(Problem p) {
  Node node = new Node(p.getInitialState());
  while (true) {
    Node next = null;
    for (Node n : node.expand()) {
      if (p.value(n) >= p.value(node)) {
        next = n;
      }
    }
    if (next == null) {
      return node.getState();
    } else {
      node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

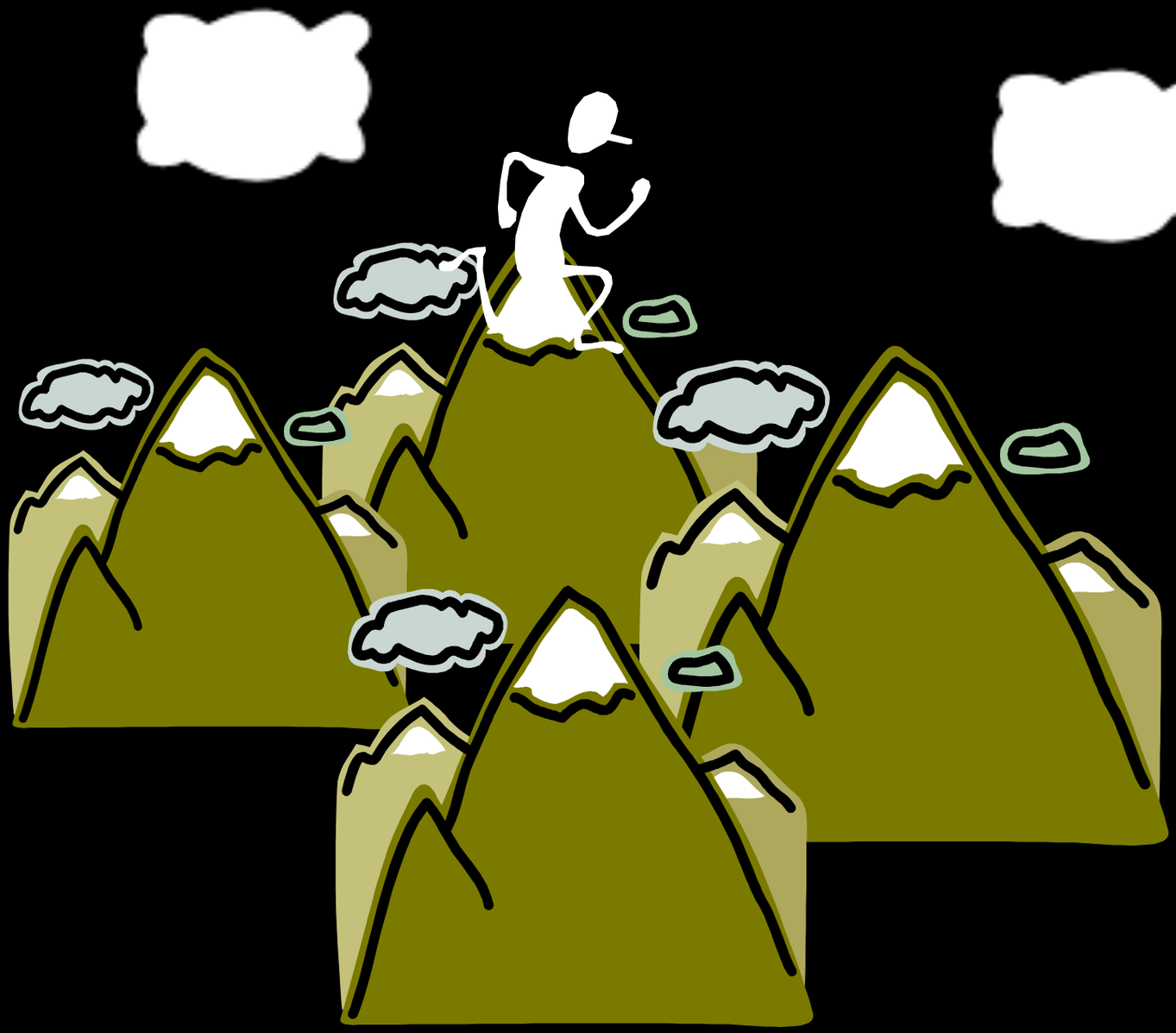




$$h(n) = 1$$


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





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

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

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

Does it work?

```
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)

```
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?

```
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 = $1/p$

```
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 = 0.14

Expected # of tries = $1/p \approx 7$

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

Randomness

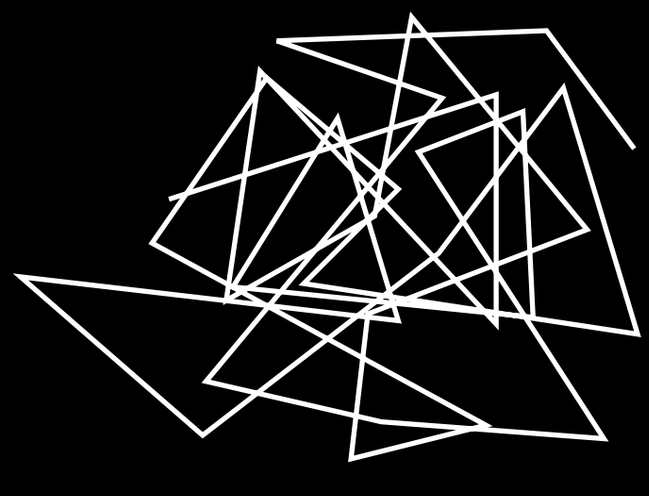
```
State randomRestart(Problem p) {
  while (true) {
    p.setInitialState(new random State);
    State solution = hillClimb(p);
    if (p.isGoal(solution)) {
      return solution;
    }
  }
}
```

Randomness

Stochastic hill climbing

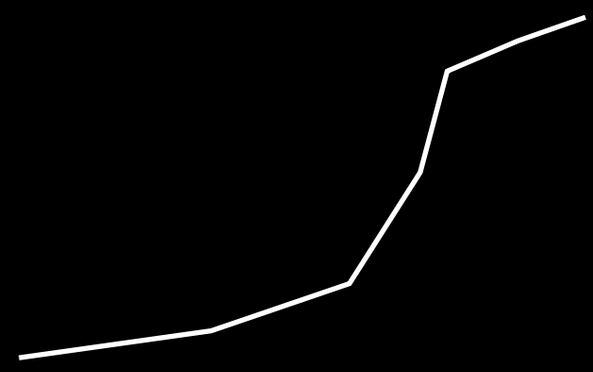
Randomness in Search

Pure random walk



Complete,
but horribly slow

Greedy local search

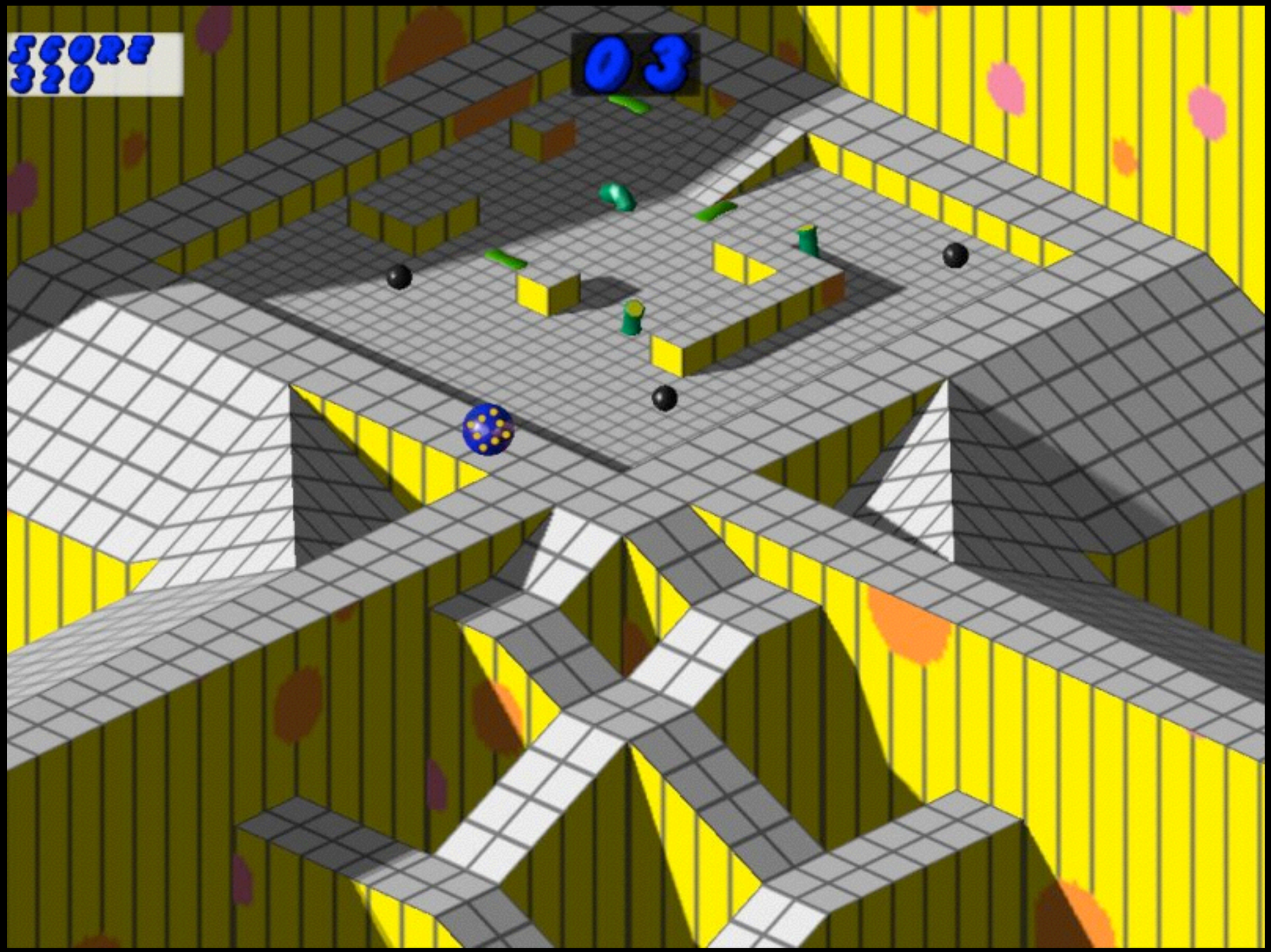


Incomplete,
but fast

MARBLE MADNESS™

Amiga Version by:
Larry Reed

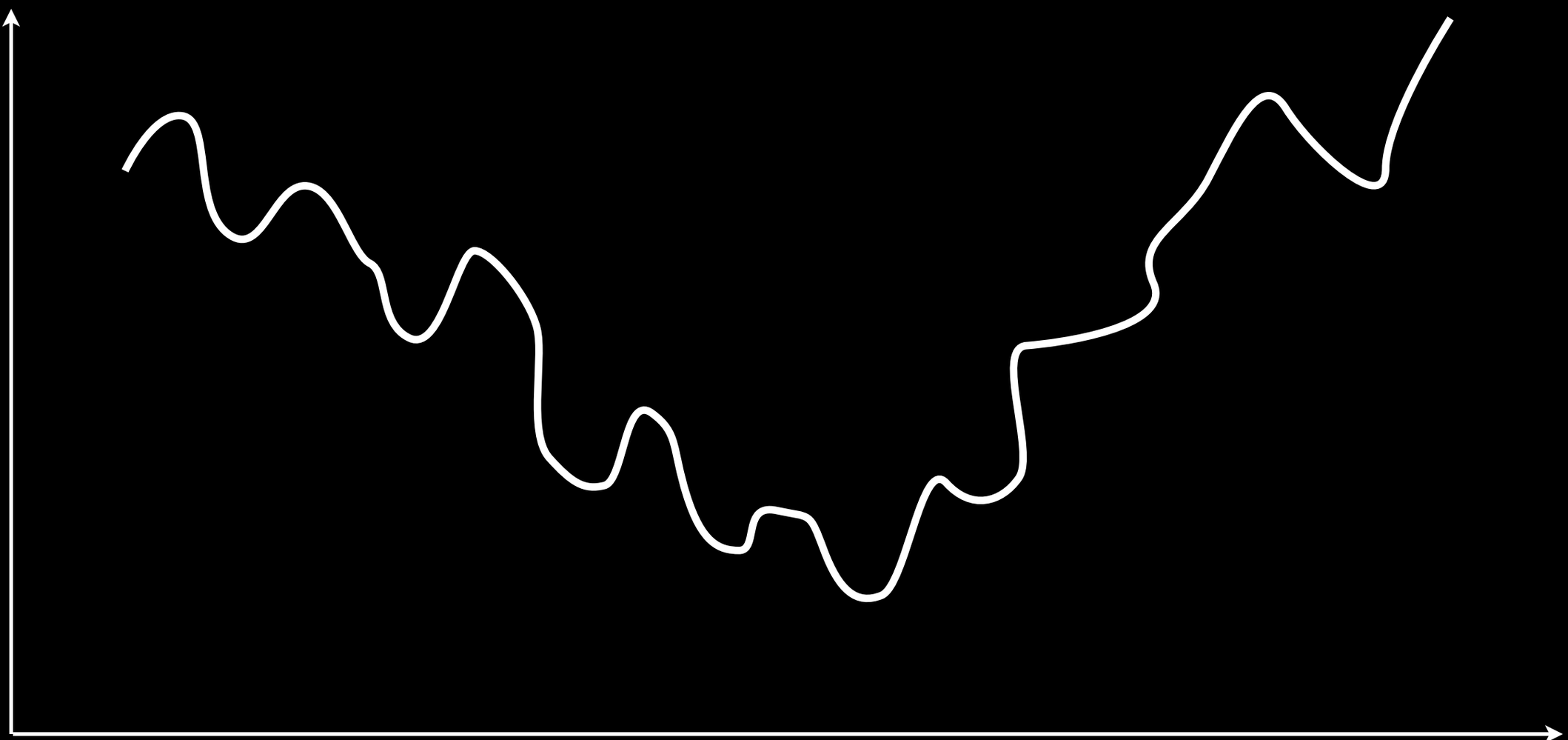
EA


SCORE
320
03

Simulated Annealing

- Follow landscape down towards global minimum of state cost function
- Occasionally allow an upward move ("shake") to get out of local minima
- Don't shake so hard that you bounce out of global minimum

Cost

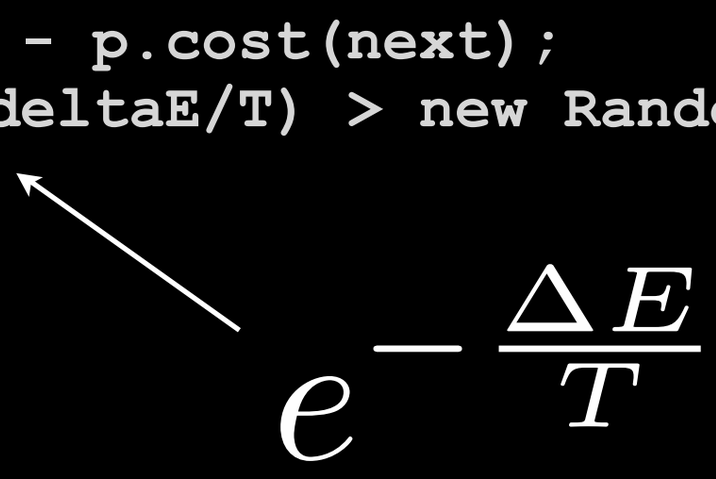


State space

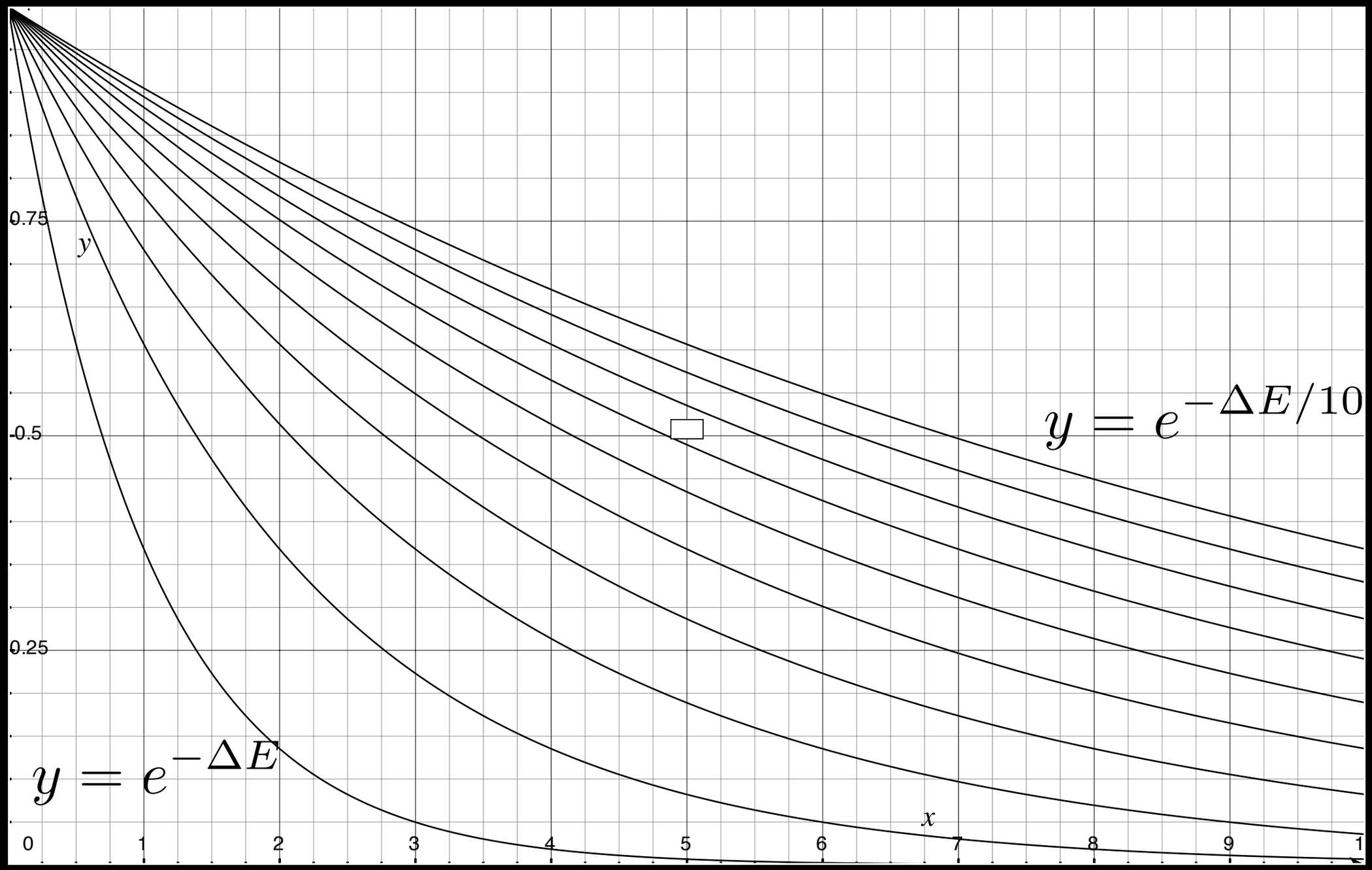
Annealing

- A heat treatment that alters the microstructure of a material causing changes in properties such as strength and hardness and ductility
- High temp => Atoms jumping around
- Low temp => Atoms settle into position

```
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;
    }
  }
}
```



$$e^{-\frac{\Delta E}{T}}$$


0.75
y
0.5
0.25
$y = e^{-\Delta E/10}$
$y = e^{-\Delta E}$
x
0
1
2
3
4
5
6
7
8
9
1

Simulated Annealing

Simulated Annealing

Complete?

Simulated Annealing

Complete? Yes, but.

Simulated Annealing

Complete? Yes, but.

Optimal?

Simulated Annealing

Complete? Yes, but.

Optimal? Yes, but.

Simulated Annealing

Complete? Yes, but.

Optimal? Yes, but.

"If the schedule lowers T slowly enough, and the search space is connected, 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

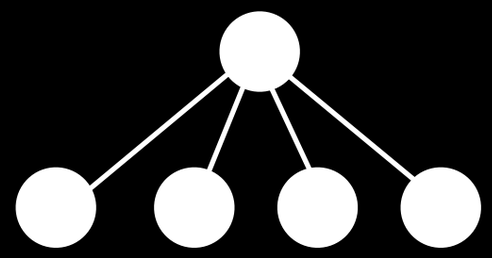
Local Beam Search

- During hill-climbing:
  - Keep track of *k* states rather than just one
    - At each step, generate all successors of all *k* states (*k*b* of them)
    - Keep the most promising *k* of them

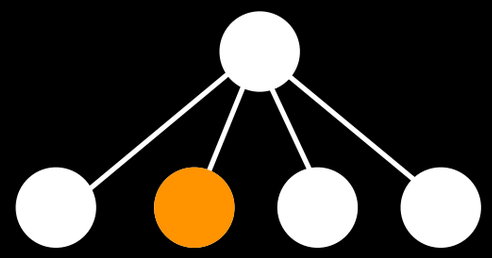
Local Search

•

Local Search

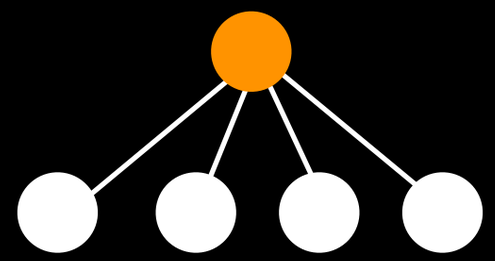


Local Search

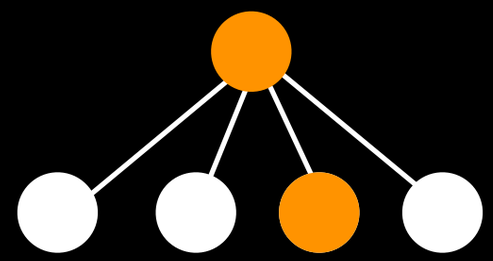


Local Search

Local Search

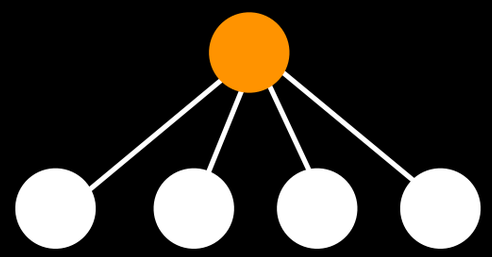


Local Search



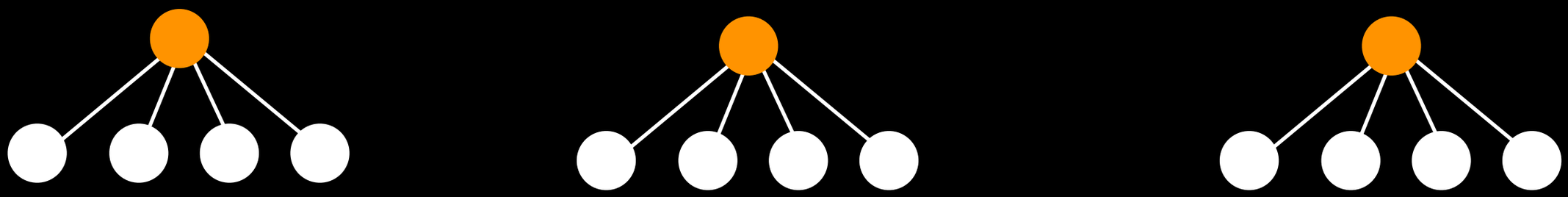
Local Search

Local Search

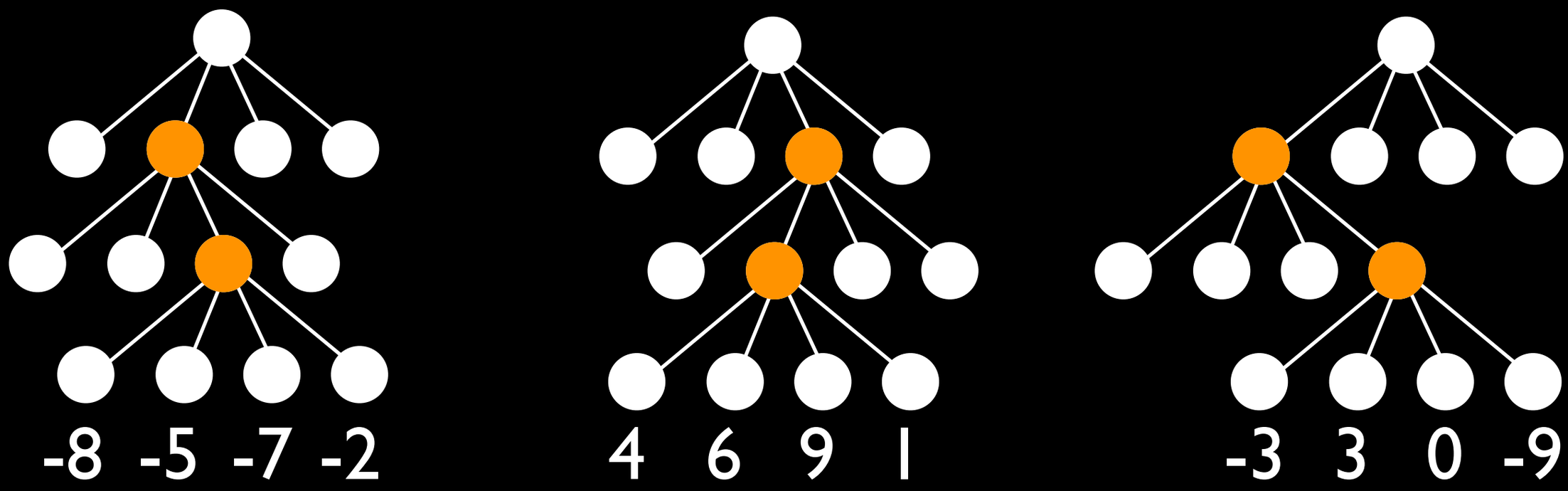


Parallel Local Search

Parallel Local Search

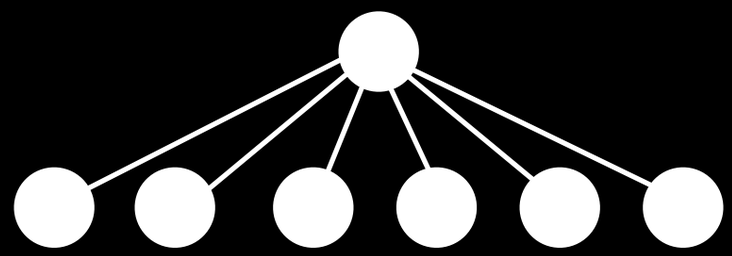


Parallel Local Search

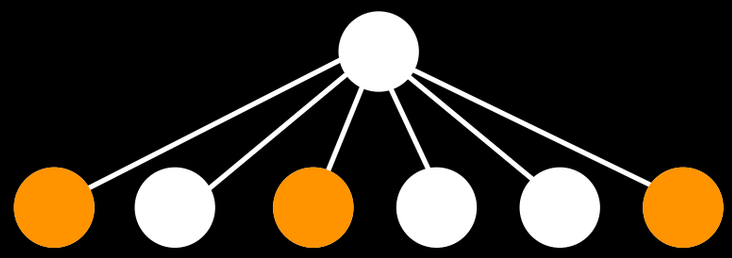


Local Beam Search

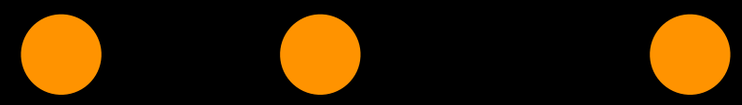
Local Beam Search



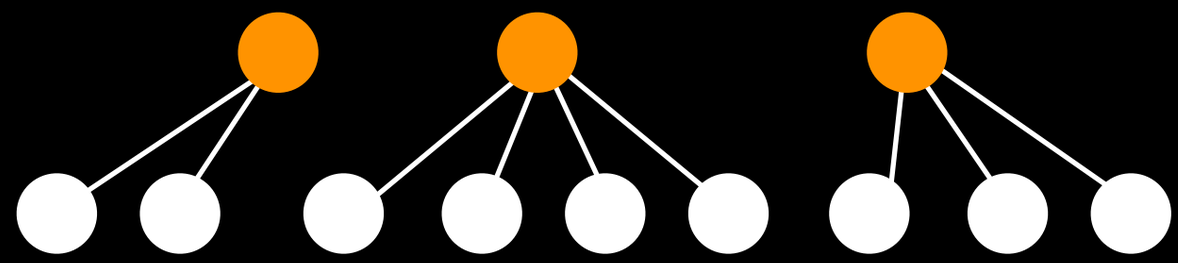
Local Beam Search



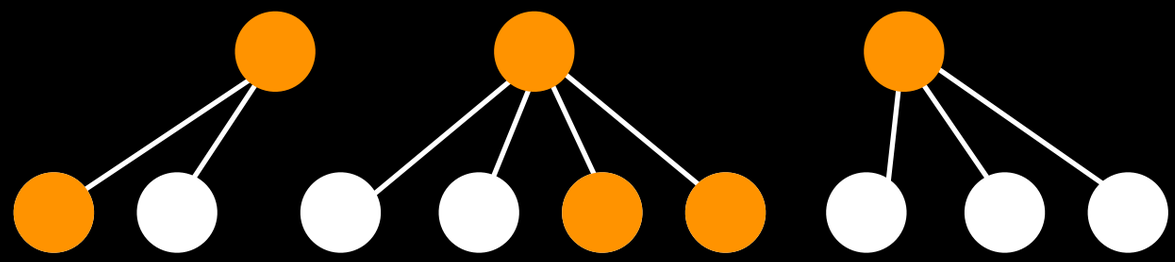
Local Beam Search



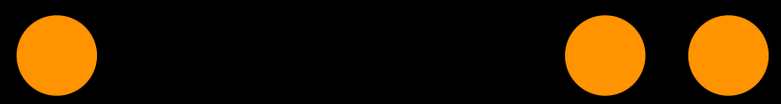
Local Beam Search



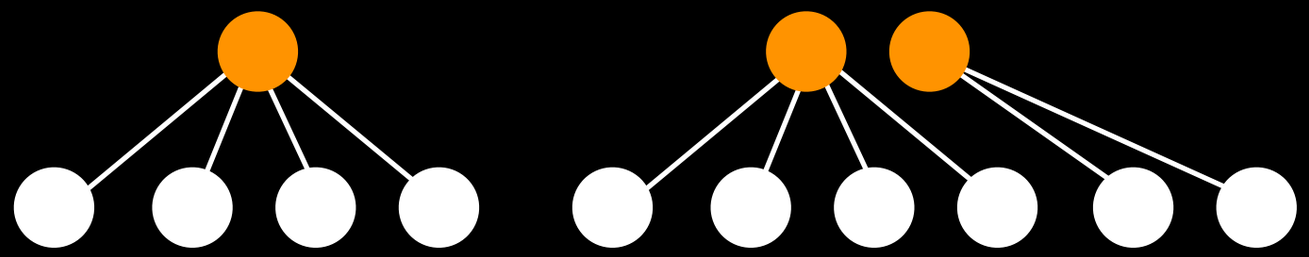
Local Beam Search



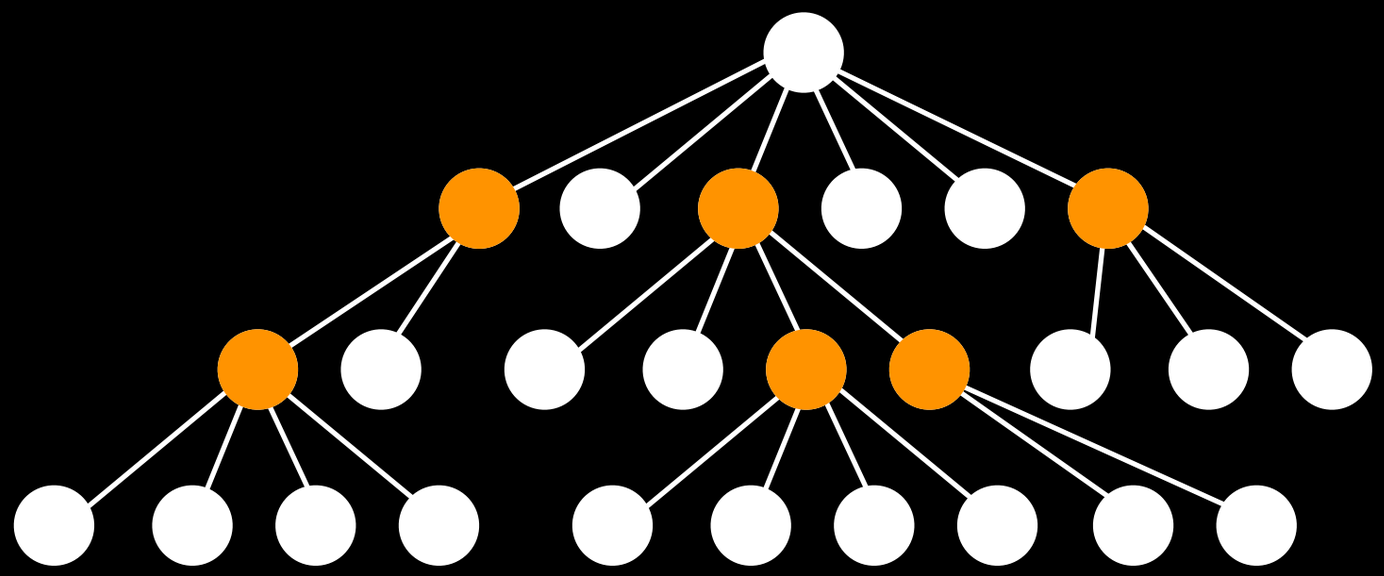
Local Beam Search

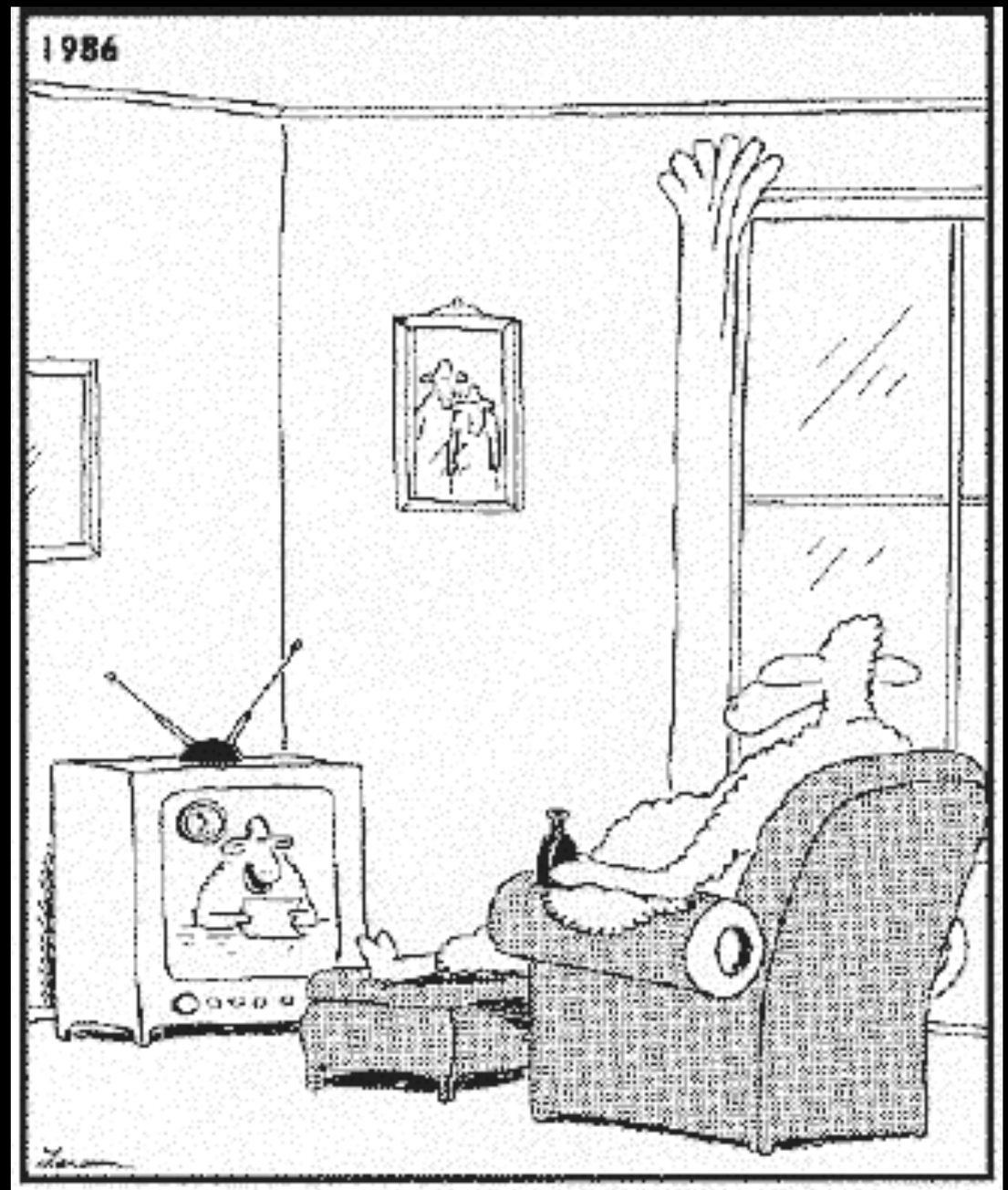


Local Beam Search



Local Beam Search



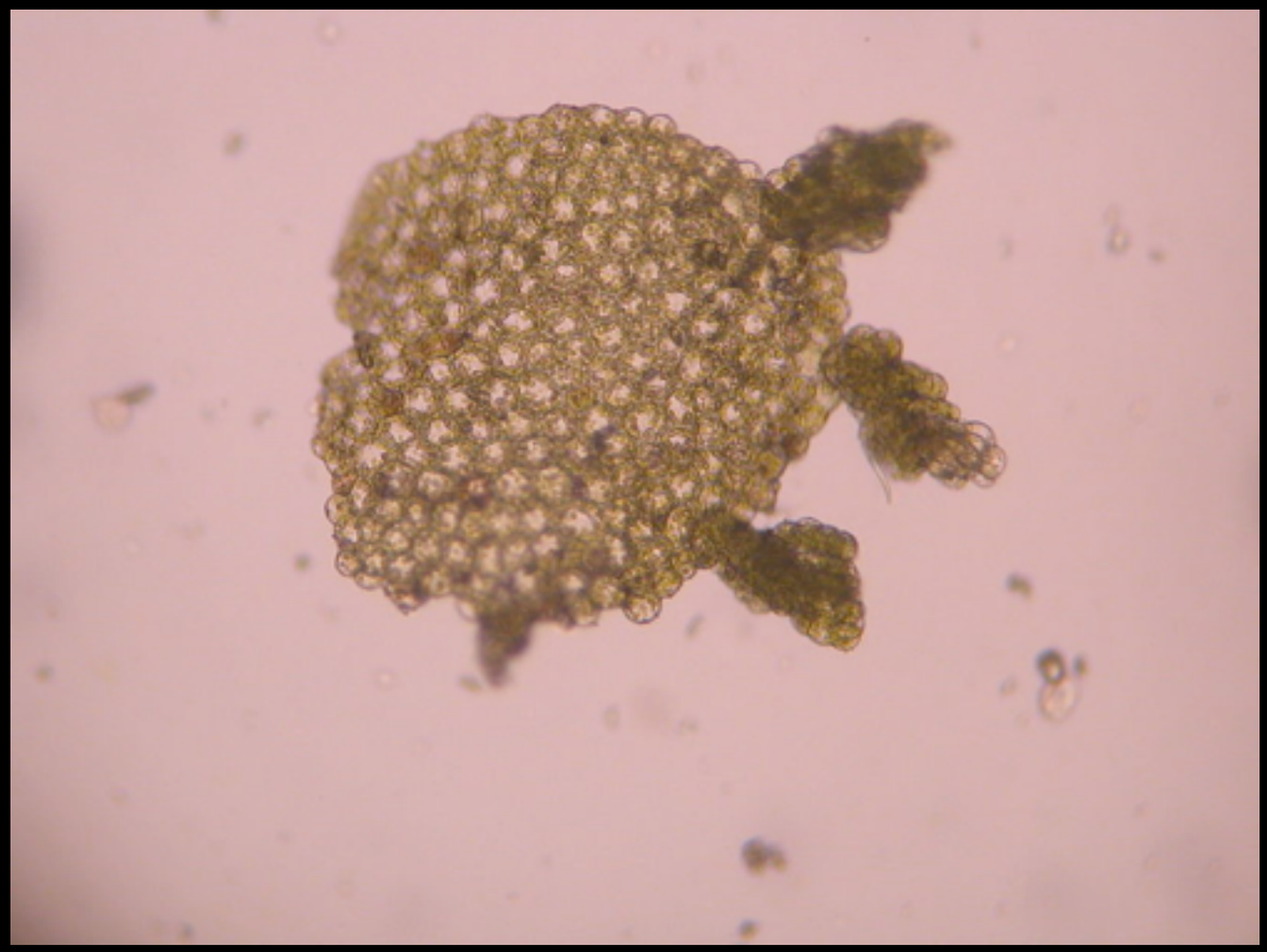


"And this report just in. ... Apparently, the grass *is* greener on the other side."

Local Beam Search

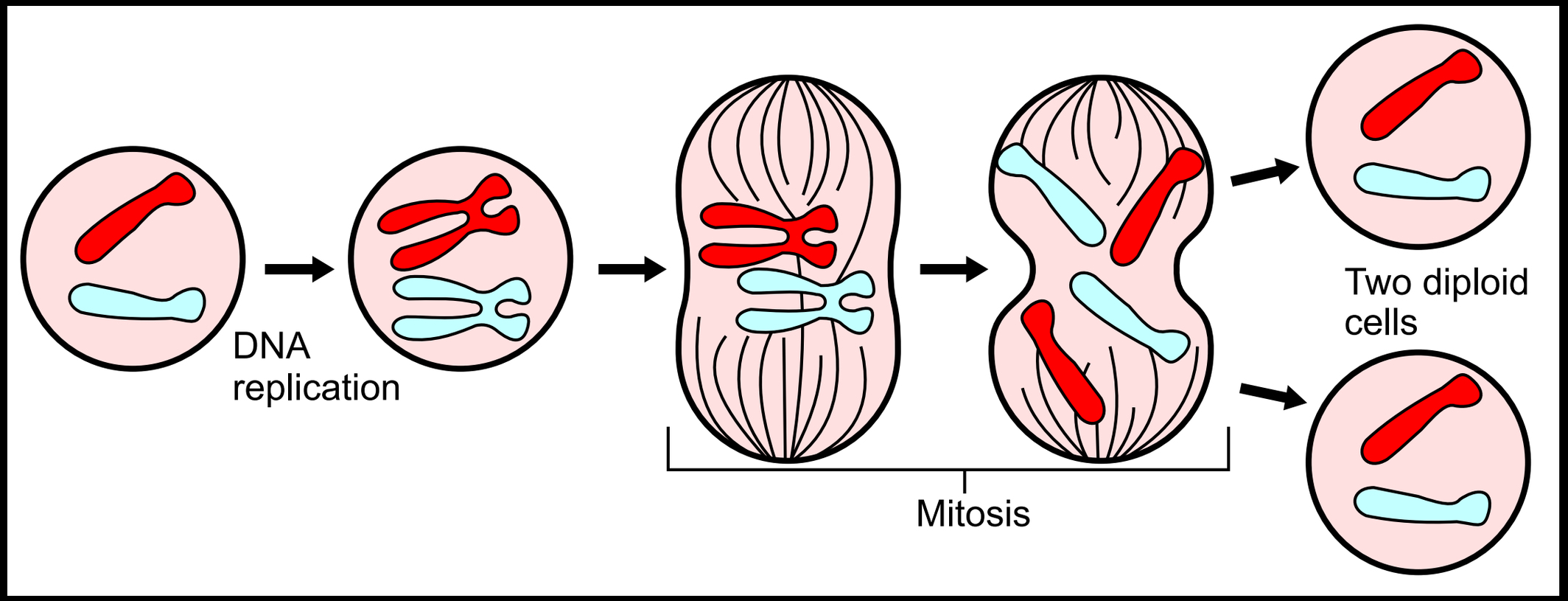
- During hill-climbing:
  - Keep track of *k* states rather than just one
    - At each step, generate all successors of all *k* states (*k*b* of them)
    - Keep the most promising *k* of them

Asexual Reproduction



Wikipedia

Mitosis

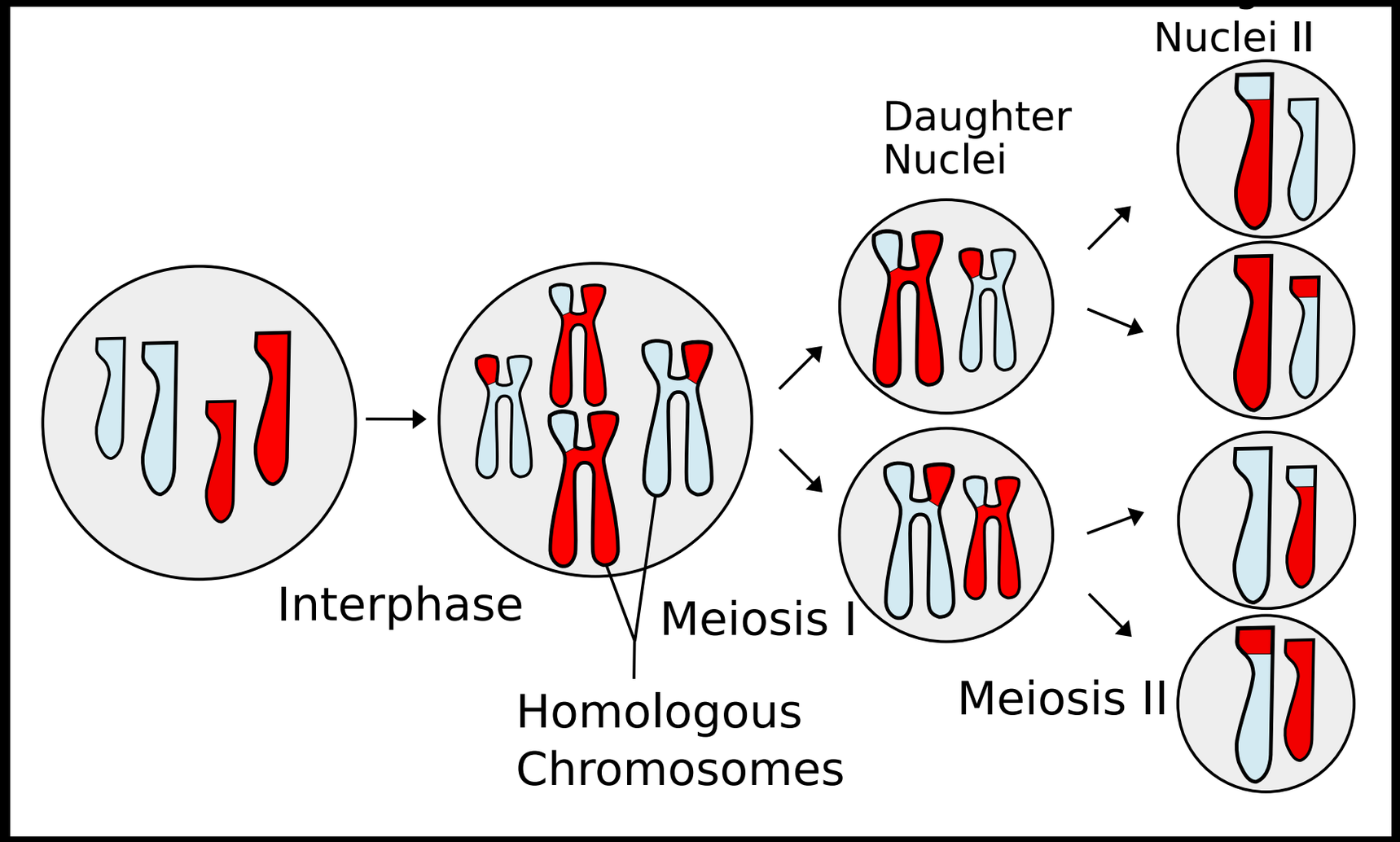


Wikipedia

Sexual Reproduction

Wikipedia

Meiosis



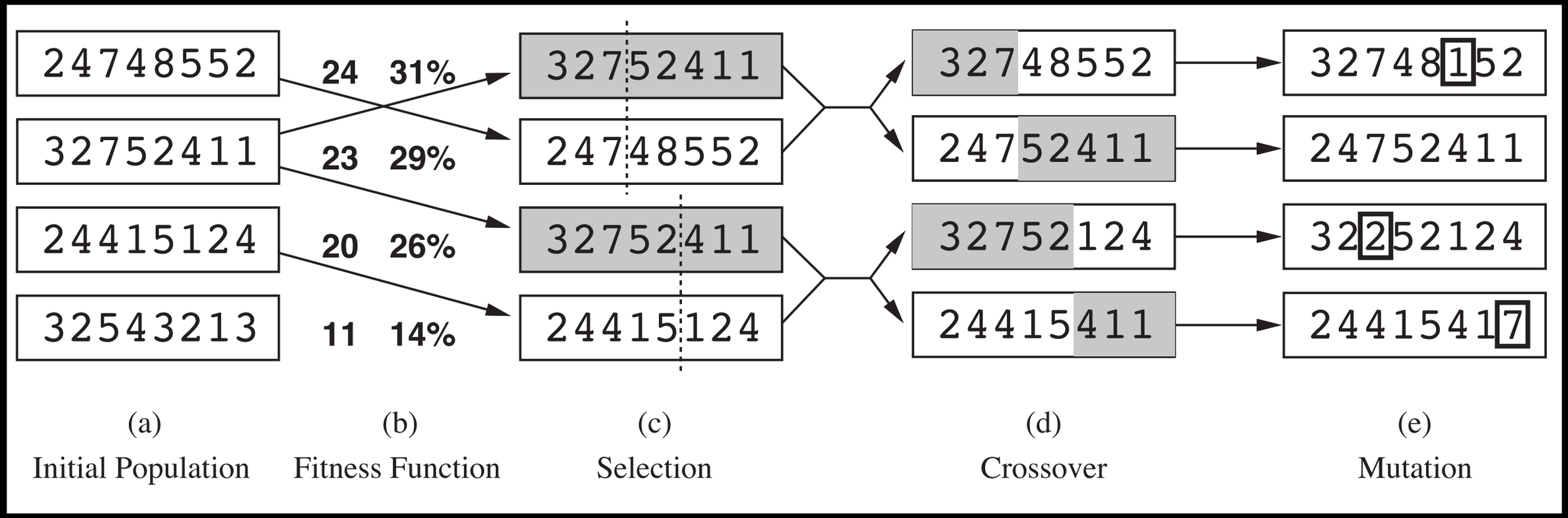
Wikipedia

Genetic Algorithms

- Start with *k* random states
- Select pairs of states and have them "mate" to produce "offspring"
- Most fit (highest-scoring) individuals reproduce more often

Genetic Algorithms

- States encoded as "chromosomes" (linear sequences, a.k.a. strings)
- During mating:
  - Crossover: swap chunks of code
  - Mutation: randomly change bits of code


24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
(a)
Initial Population
(b)
Fitness Function
(c)
Selection
(d)
Crossover
(e)
Mutation

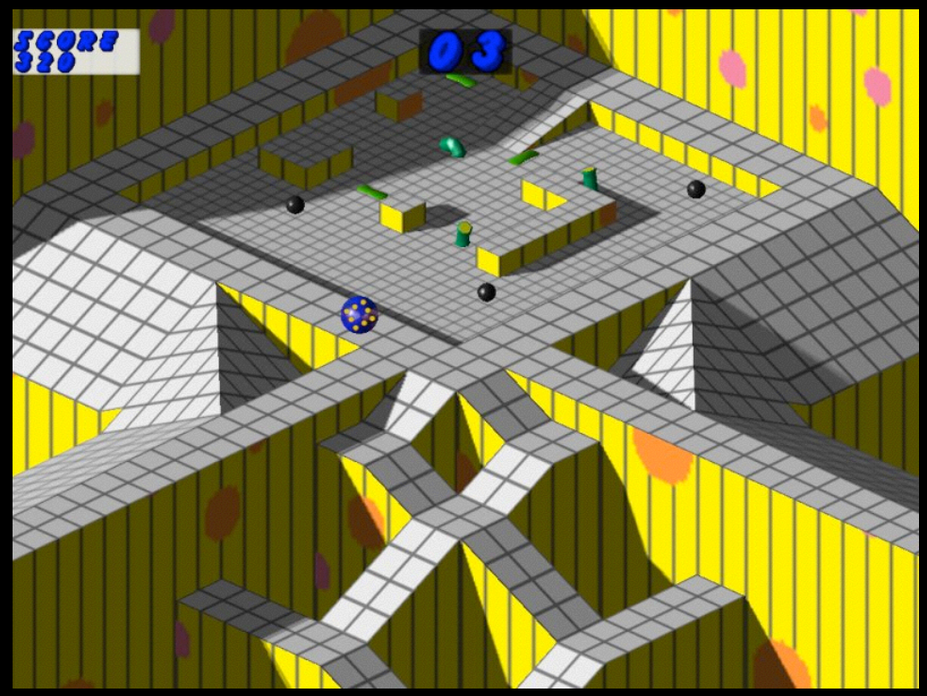
Genetic Algorithms

- States encoded as "chromosomes" (linear sequences, a.k.a. strings)
- During mating:
  - Crossover: swap chunks of code
  - Mutation: randomly change bits of code

GA Summary

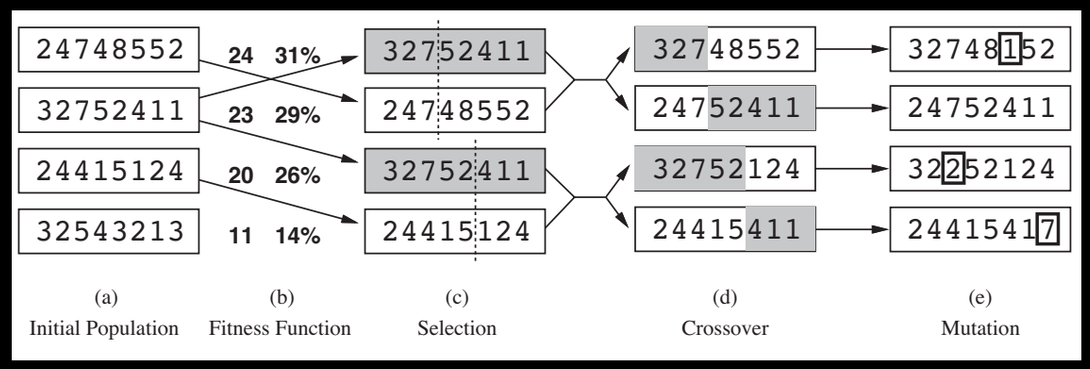
- A version of stochastic local beam search with a special way to generate successor states (motivated by a naive biology analogy)
- "Much work remains to be done to identify the conditions under which genetic algorithms perform well."

Hill-climbing

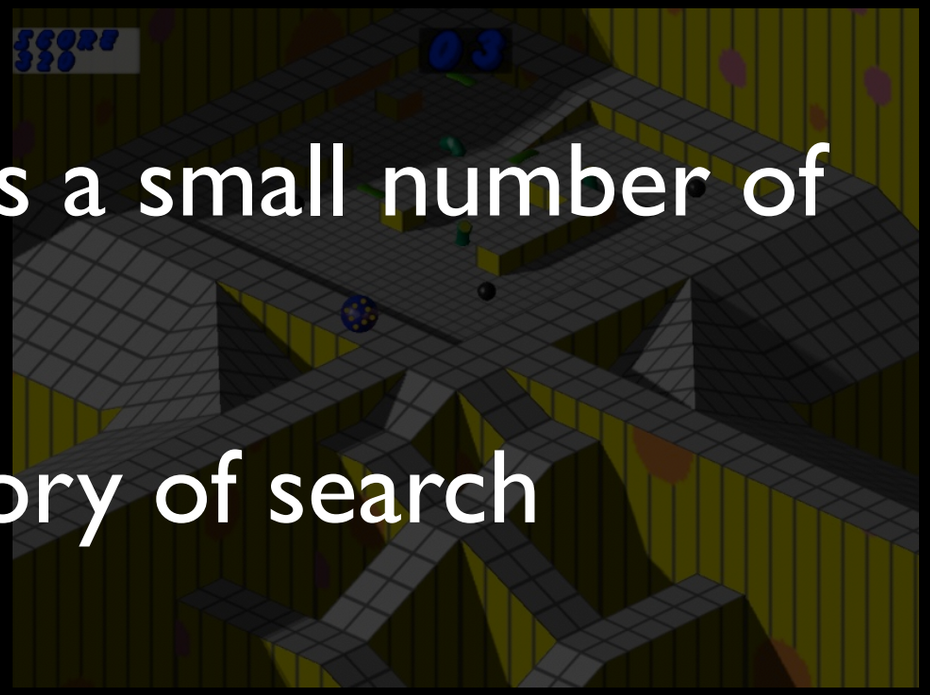


Simulated Annealing

Local Beam Search



Genetic Algorithms



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

Hill-climbing

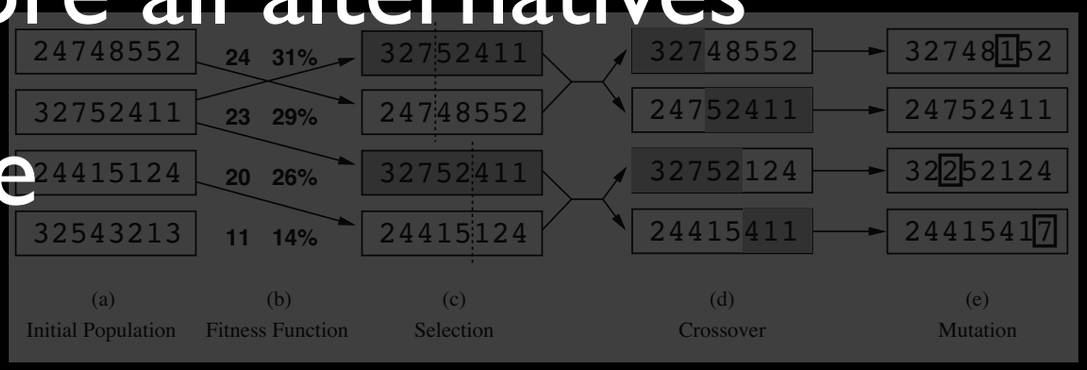
Good: Very little (constant) memory

Bad: May not explore all alternatives

=> Incomplete

Local Beam Search

Simulated Annealing



Genetic Algorithms

For next time:

AIMA 5.0–5.2.2

Upper-level writing: Topics due:

What subject will you be writing about?

What questions about it will you try to answer?

Homework 1: Do it (really!)