FACULTY OF
COMPUTER SCIENCE

IA@FII

# Artificial Intelligence

3rd year, 1st semester

## State based models for decision problems

### Week 2: models and uninformed search strategies

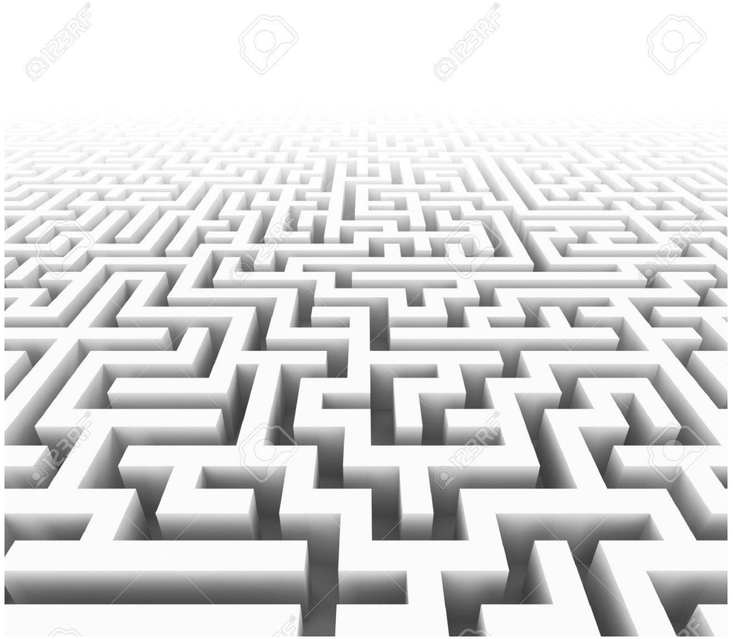
“Intelligence is what you use when you don't know what to do”
Jean Piaget

## We aim to develop rational AI systems:

- we define a problem and goals (instances)
- the AI reaches the goals (solves the instances)
- the AI outputs the result (unexplainable AI) or the solution (explainable AI)

## First approach: state-based models

- in what way should I change the current *state* of the problem in order to get closer or reach the goal?
- compute a solution (algorithm - sequence of steps) starting from an initial *state* and ending in the goal *state*
- mostly covered by search strategies, reasoning systems, AI for games



FACULTY OF
COMPUTER SCIENCE

IA@FII

## Solving problems

Problem: We have two numbers, A and B. Which is larger?

Instance: A=?, B=?

A solution should produce the correct answer for **all** possible instances

## What’s the Problem with problems?

P = set of problems that can be solved/checked in Polynomial time by a deterministic TM.

NP = set of problems that can be solved in Polynomial time by a non-deterministic TM.
Validating a NP solution is a P problem.

NP-complete = subset of NP with the property that any problem in NP can be reduced to a NP-complete problem in polynomial time.

NP-hard = set of problems with the property that any problem in NP can be reduced to a NP-hard problem in polynomial time.

P=NP ?

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Let the computer solve our problems! Or not...

Solvable NP-hard problems: QSAT: is a logical proposition using quantified (existential $\exists$ or universal $\forall$) variables satisfiable?

Undecideable NP-hard problems: Turing Halting problem: given an algorithm and an input, determine whether the algorithm execution for that input will eventually halt.

The rest can be solved by a computer.

## Stop looking for solutions!

Don’t think about solutions, think about problems.

Solving problems requires effort, describing problems requires intelligence.

“If I had only one hour to save the world, I would spend fifty-five minutes defining the problem, and only five minutes finding the solution.” A.Einstein

## Let’s find a NP-complete problem and help a computer to solve it

Problem: finding a path in a graph

Describing a model - reducing a problem

- describe a state
- identify special states and the problem space
- describe the transitions and validate them
- specify a search strategy

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Problem: finding a path in a graph

Finding a path in a graph is NP-complete, however it can easily be scaled up to NP-hard by many slight reformulations, such as:

- Finding a path if the graph has cycles
- Finding the longest path
- Graph coloring
- Finding cliques

This means that you can also solve NP-hard problems using the same method by adapting the requirements.

## Generalized Hanoi Towers

We have *n* towers with *m* distinct sized pieces placed on one of the towers. Considering that you can only move one piece at a time and that no piece can be placed on top of a smaller piece, find a sequence of movements that place all pieces on a different tower.

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Choosing a representation for a state

State: all data required to continue looking for a solution

No ambiguity

Expressive enough to include all required data

## Choosing a representation for a state

Compact enough to be easy to store and perform changes on it

A list of towers pieces are placed, in the order of piece size size

(3, 3, 3, 3, 3, 1, 1, 1, 2, 2)

Is this good enough?

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Choosing a representation for a state

Compact enough to be easy to store and perform changes on it

A list of towers where pieces are placed, in the order of piece size, with the number of towers added at the beginning

(3, 3, 3, 3, 3, 3, 1, 1, 1, 2, 2)

$(n, t_1, t_2, \ldots, t_m)$, $1 \leq t_i \leq n$

## Choosing a representation for a state is the most **important and difficult** step

No ambiguity

Compact

Expressive

Includes all required data

There is no generally best representation

## Special states

Initial state (3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

```
State Initialize (int n, int m)
{
        Return $(n, \underbrace{1, 1, 1, \ldots, 1}_{m})$;
}
```

There can be more than one initial state. There has to be at least one initial state.

## Special states

Final state(s) (n, n, n, n, n, n, n, n, n, n, n)

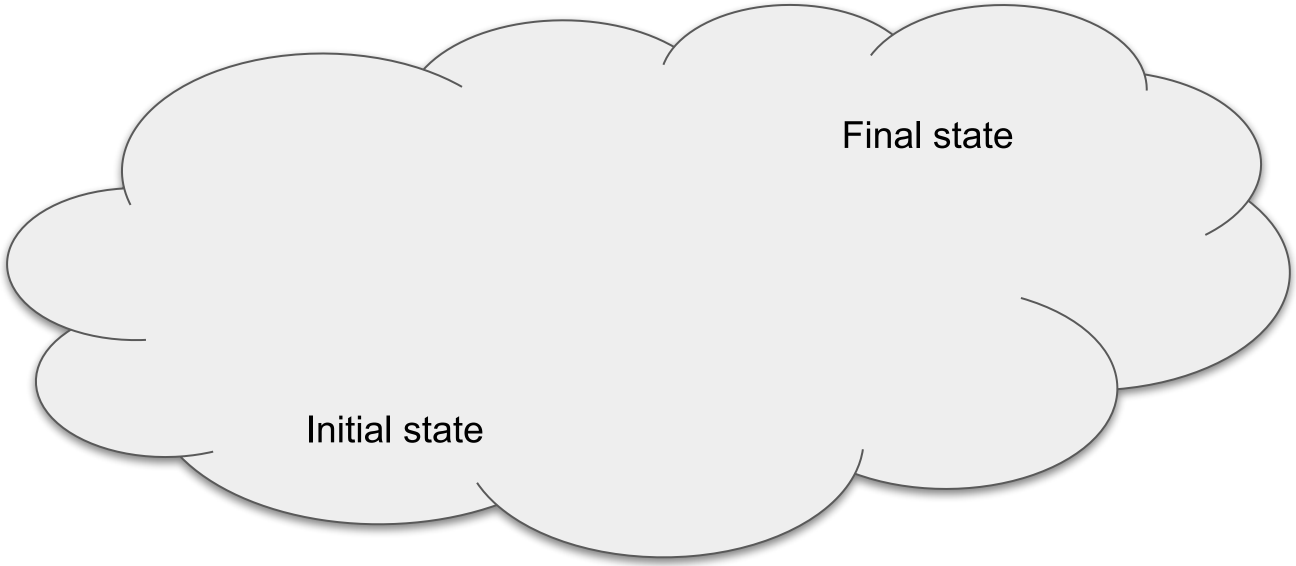
```
Boolean IsFinal (State s)
{
    If s = $\underbrace{(k, k, k, \ldots, k)}_{m+1}$ then return true;
    else return false;
}
```

There can be more than one final state. There has to be at least one final state.

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Problem space: all possible states



Finite space (limited variability in chosen state representation and only valid states are included)

What is the problem now?

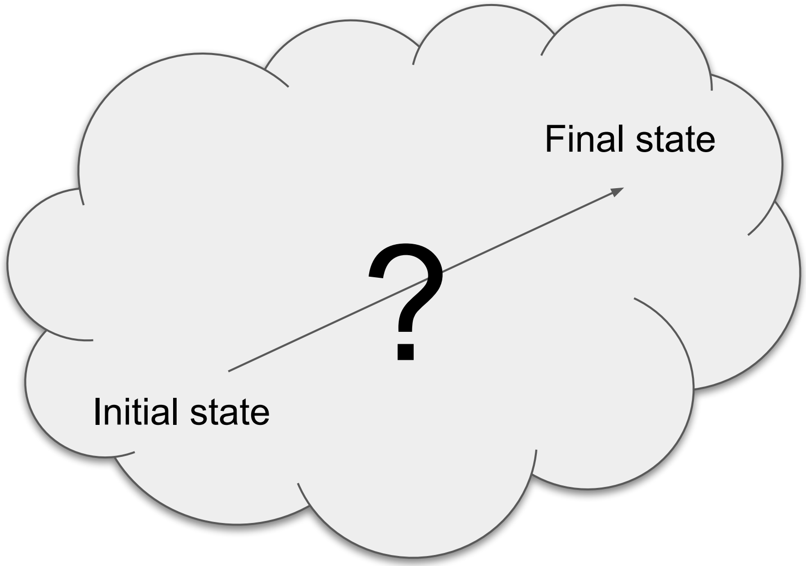
How difficult is it to solve?

Problem space has *m* dimensions

How many states are within the space?

How can you move within the
problem space?

Simon’s Ant



Complexity of the solution is determined by the problem, NOT by the way in which you solve it

## Transitions

Only one possible way to change current state: move one piece to another tower.

$(n, t_{11}, t_{12},\ldots,t_{1m}) \rightarrow (n, t_{21}, t_{22},\ldots,t_{2m})$, where $t_{1i} = t_{2i}$ for all $1 \leq i \leq m$, except exactly one $i = k$

State Transition (State s, piece, tower)

## Valid transitions

1. No smaller piece is placed atop piece k
2. No smaller piece ends up below piece k

1. $t_{1i} \mp t_{1k}$ , for all $1 \leq i < k$
2. $t_{2i} \mp t_{2k}$ , for all $1 \leq i < k$

Boolean Validate (State s, piece, tower)

Implement transitions and validation separately, it’s good practice!

## Search strategy

```
Void strategy(State s)
{
    While (!isFinal(s))
        {
            Choose piece, tower;
            If (Validate (s, piece, tower))
                s = Transition(s, piece, tower);
        }
}
```

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Search strategies

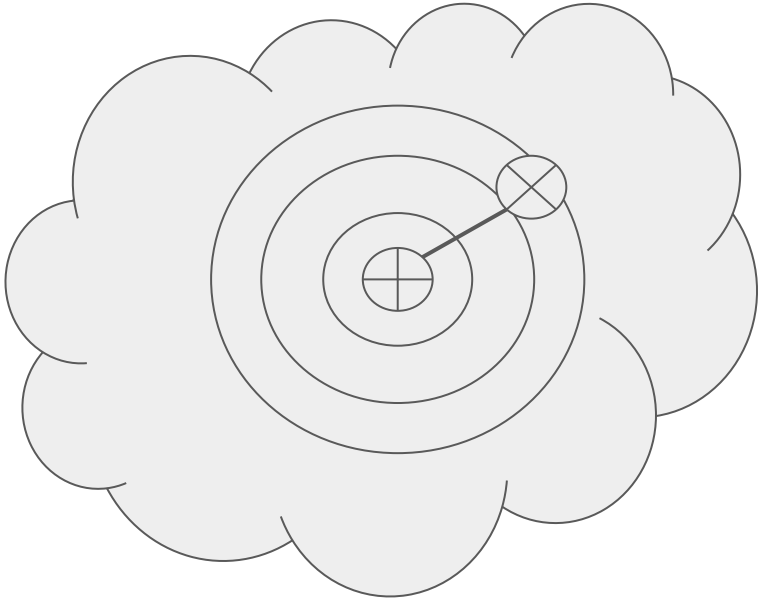
- Uninformed - no distinction between states
  - Random
  - BFS and Uniform Cost
  - DFS and Iterative Deepening
  - Backtracking
  - Bidirectional
- Informed - heuristics to help distinguish between states
  - Greedy best-first
  - Hillclimbing and Simulated Annealing
  - Beam
  - A* and IDA*

FACULTY OF
COMPUTER SCIENCE

IA@FII

## Breadth First Search and Uniform Cost

- Visit states in the order of distance (number of transitions) from the initial state.
- Explores all immediate neighbors (accessible states) until no more neighbors or final state found.
- Has to memorise each generated state: very costly.
- Might visit the same state multiple times.
- Finds shortest path (optimum solution)
- Uniform cost: if options have different costs, explore cheaper paths first

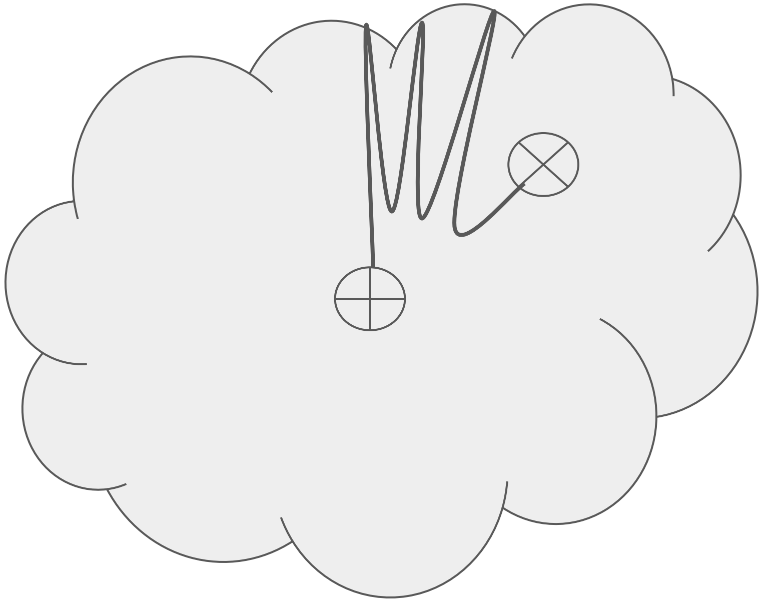


FACULTY OF
COMPUTER SCIENCE

IA@FII

## Depth First Search and Iterative Deepening Search

- Visit one immediate neighbor of the current state until final state is found, return to previous unexplored neighbor if no more neighbours available for current state.
- Has to memorise each generated state: very costly.
- Might visit the same state multiple times.
- Might not finish (if loops are present in the problem space)
- IDS: Explore only up to an increasing depth (distance from initial state) - no more infinite paths



## Random vs DFS

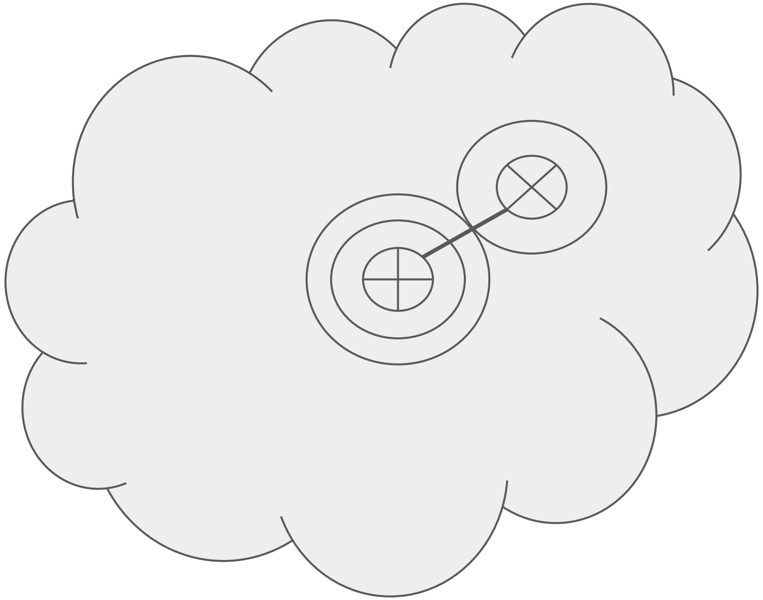
- The same, if random remembers visited neighbours

## Backtracking vs DFS

- NOT the same
- Backtracking sets an order for the neighbor states
- How do you order states?
- Backtracking doesn't need to memorize visited states!
- Backtracking can avoid loops WITHOUT storing visited states
- Backtracking  can prune invalid branches earlier, if the order and selection of neighbours is optimal

## Bidirectional Search

- Starts exploring from both the initial and final state simultaneously
- Has to memorise each generated state, but they should not be that many
- Finds shortest path (optimum solution)
- What are the reverse transitions?



## Comparison

Complexity of the solution is given by the problem, NOT by the way in which you solve it

| Criterion | BFS | DFS | IDS | Bidirectional | BKT | Random |
|---|---|---|---|---|---|---|
| ETC | $N^O$ | $N^A$ | $N^O$ | $N^O$ | $N^A$ | $N^A$ |
| Optimum | Yes | No | Yes | Yes | No | No |
| All | Yes | No | Yes | Yes | Yes | No |

N = average number of accessible states
O = length of the optimum solution
A = length of the average solution

## Additional references

Uninformed Search Algorithms

Stanford Programming Abstractions