

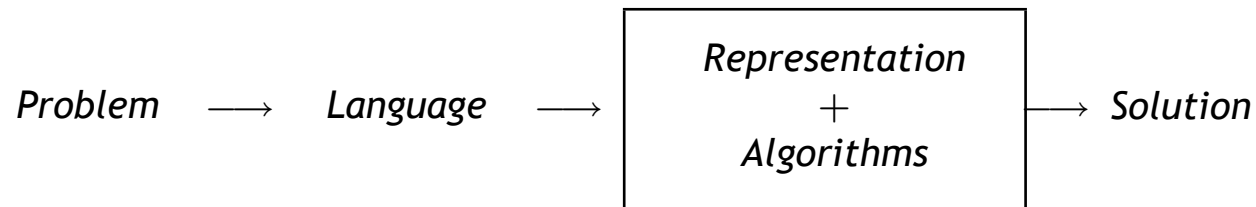
# **Artificial Intelligence 1 : State Models and Search**

Héctor Geffner

ICREA and Universitat Pompeu Fabra  
Barcelona, Spain

# Motivation

- To a large extent Artificial Intelligence (AI) concerned with **modeling** and **solving** problems by means of computers



- Ideally one would describe problem at high-level, and computer would take care of the rest
- Example problems

<i>8-puzzle</i>	<i>rubik</i>	<i>mastermind</i>	<i>12 coins</i>
<i>diagnosis</i>	<i>scheduling</i>	<i>tsp</i>	<i>robot navig</i>
<i>sorting</i>	<i>minesweeper</i>	<i>crypto-arith</i>	<i>vehicle routing</i>
<i>...</i>	<i>...</i>		

# Methodology

*how to make sense of this variety of problems?*

- Many approaches pursued; many useful ideas and techniques
- To a certain extent AI has become large **bag of tools and techniques**
- This is unfortunate for both teaching and research
- Goal of course is to provide **coherent framework** for AI modeling and problem solving
- Coverage is broad but necessarily incomplete

# AI Modeling and Problem Solving

Three distinguished components:

- **Representation languages** for describing problems conveniently
- **Mathematical models** for making sense of classes of problems
- **Algorithms** for solving these models

# Models and Solutions

- **Mathematical models** provide suitable **abstraction** of certain classes of problems
- E.g., games such as the **15-puzzle** and **Rubik's cube** can be described by operations that change the configuration of the game; in both cases, the goal is to assemble a **sequence of operations** that map an initial configuration into a target one
- On the other hand, problems such as **Tic-tac-toe** or **Mastermind** have a different **structure** and a different **solution form** which is **not** a fixed sequence of operations

## Some mathematical models that you may know

- Systems of  $n$  linear equations in  $n$  unknowns; e.g.  $x + y = 60$  and  $x = 3y$
- You also know how to solve them (e.g., gaussian elimination)
- Thus, if you can formulate a problem in this form you can solve it by applying a **general method**
- E.g., What's John's age given that his has 3 times the age of his son and both ages together add up to 60?
- Other models that you probably know: **linear programming models, shortest path models in graphs, . . .**
- Each model has certain degree of generality and defines clearly what's a **problem** and what's is a (optimal) **solution**

# State Models

- State models are the most basic models in AI
- They are characterized by
  - finite and discrete state space  $S$
  - an initial state  $s_0 \in S$
  - a set  $G \subseteq S$  of goal states
  - actions  $A(s) \subseteq A$  applicable in each state  $s \in S$
  - a transition function  $f(s, a)$  for  $s \in S$  and  $a \in A(s)$
  - action costs  $c(a, s) > 0$
- A **solution** is a sequence of applicable actions  $a_i$ ,  $i = 0, \dots, n$ , that maps the initial state  $s_0$  into a goal state  $s \in S_G$ ; i.e.,  $s_{n+1} \in S_G$  and for  $i = 0, \dots, n$ 
$$s_{i+1} = f(a_i, s_i) \text{ and } a_i \in A(s_i)$$
- **Optimal** solutions minimize total cost  $\sum_{i=0}^{i=n} c(a_i, s_i)$

# Examples: Problems mapping into State Models

- Grid Navigation
- 15-puzzle (n-puzzle)
- Route Finding in Map
- TSP (Traveling Salesman Problem)
- Jug Puzzles (e.g., 4 & 3 liter jars, have 2 liters in 4 lit. jar)
- ⋮

# Algorithms for Solving State Models

**Search algorithms** explore/visit state space trying to find (optimal) path from  $s_0$  to  $S_G$

Correspondence between **directed graphs** and **state models**: search algorithms for state models reduce to **single source shortest-path algorithms** in directed graphs:

- **Blind search/Brute force algorithms**

- Goal plays **passive** role in the search

- e.g., Depth First Search (DFS), Breadth-first search (BrFS), Uniform Cost (Dijkstra), Iterative Deepening (ID)*

- **Informed/Heuristic Search Algorithms**

- Search uses a function  $h(s)$  that estimates 'distance' (cost) from state  $s$  to  $S_G$  to guide search

- e.g., A\*, IDA\*, Hill Climbing, Best First Search (BFS), Branch & Bound*

# General Search Scheme

Solve(Nodes)

if Empty Nodes -> Fail

else Let Node = Select-Node Nodes

Let Rest = Nodes - Node

if Node is Goal -> Return Solution

else Let Children = Expand-Node Node

Let New-Nodes = Add-Nodes Children Nodes

Solve(New-Nodes)

- Different algorithms obtained by suitable instantiation of
  - Select-Node *Nodes*
  - Add-Nodes *New-Nodes Old-Nodes*
- Nodes are data structures that contain state and bookkeeping info; initially  $Nodes = \{root\}$
- Notation  $g(n)$ ,  $h(n)$ ,  $f(n)$ : accumulated cost, heuristic and evaluation function; e.g. in  $A^*$ ,  $f(n) \stackrel{\text{def}}{=} g(n) + h(n)$

## Some instances of general search scheme

**Depth-First Search** expands 'deepest' nodes  $n$  first

- Select-Node *Nodes*: Select First Node in *Nodes*
- Add-Nodes *New Old*: Puts *New* before *Old*
- Implementation: Nodes is a **Stack** (LIFO)

**Breadth-First Search** expands 'shallowest' nodes  $n$  first

- Select-Node *Nodes*: Selects First Node in *Nodes*
- Add-Nodes *New Old*: Puts *New* after *Old*
- Implementation: Nodes is a **Queue** (FIFO)

## Additional instances of general search scheme

**Best First Search** expands best nodes  $n$  first;  $\min f(n)$

- Select-Node *Nodes*: Returns  $n$  in Nodes with  $\min f(n)$
- Add-Nodes *New Old*: Performs ordered merge
- Implementation: Nodes is a **Heap**
- Special cases
  - **Uniform cost/Dijkstra**:  $f(n) = g(n)$
  - **A\***:  $f(n) = g(n) + h(n)$
  - **WA\***:  $f(n) = g(n) + Wh(n)$ ,  $W \geq 1$

**Hill Climbing** expands best node  $n$  first and **discards others**

- Select-Node *Nodes*: Returns  $n$  in Nodes with  $\min h(n)$
- Add-Nodes *New Old*: Returns *New*; discards *Old*

# Variations of general search scheme: Bounding

Solve(Nodes, Bound)

```
if Empty Nodes -> Report-Best-Solution-or-Fail
else
  Let Node = Select-Node Nodes
  Let Rest = Nodes - Node

  if f(Node) > Bound
    Solve(Rest, Bound)      ;;;   PRUNE NODE n

  else if Node is Goal -> Process-Solution Node Rest
  else
    Let Children = Expand-Node Node
    Let New-Nodes = Add-Nodes Children Nodes
    Solve(New-Nodes, Bound)
```

**Select-Node & Add-Nodes as in DFS**

# Some instances of general bounded search scheme

## Iterative Deepening (ID)

- Uses  $f(n) = g(n)$
- Calls `Solve` with bounds 0, 1, .. til solution found
- `Process-Solution` returns `Solution`

## Iterative Deepening A\* (IDA\*)

- Uses  $f(n) = g(n) + h(n)$
- Calls `Solve` with bounds  $f(n_0), f(n_1), \dots$  where  $n_0 = root$  and  $n_i$  is cheapest node pruned in iteration  $i - 1$
- `Process-Solution` returns `Solution`

## Branch and Bound

- Uses  $f(n) = g(n) + h(n)$

- Single call to Solve with high (Upper) Bound
- Process-Solution: updates Bound to Solution Cost minus 1 & calls Solve(Rest, New-Bound)

# Properties of Algorithms

- **Completeness:** whether guaranteed to find solution
- **Optimality:** whether solution guaranteed optimal
- **Time Complexity:** how time increases with size
- **Space Complexity:** how space increases with size

	DFS	BrFS	ID	A*	HC	IDA*	B&B
Complete	No	Yes	Yes	Yes	No	Yes	Yes
Optimal	No	Yes*	Yes	Yes	No	Yes	Yes
Time	$\infty$	$b^d$	$b^d$	$b^d$	$\infty$	$b^d$	$b^D$
Space	$b \cdot d$	$b^d$	$b \cdot d$	$b^d$	$b$	$b \cdot d$	$b \cdot d$

-- Parameters:  $d$  is solution depth;  $b$  is branching factor

-- BrFS optimal when costs are uniform

-- A\*/IDA\* optimal when  $h$  is **admissible**;  $h \leq h^*$

## A\*: Additional Properties

- A\* stores in memory **all nodes visited**
  - Nodes either in **Open** (search frontier) or **Closed**
  - When nodes expanded, children looked up in **Open** and **Closed** lists
  - Duplicates prevented and no node expanded more than once
- A\* is **optimal** in another sense: no other algorithm expands less nodes than A\* with same heuristic function (*this doesn't mean that A\* is always fastest*)
- A\* expands 'less' nodes with **more informed heuristic**,  $h_2$  more informed than  $h_1$  if  $0 < h_1 < h_2 \leq h^*$

## Practical Issues: Search in Large Spaces

- Exponential-memory algorithms like A\* **not feasible** for large problems
- **Time and memory** requirements can be lowered significantly by multiplying heuristic term  $h(n)$  by a constant  $W > 1$  (WA\*)
- Solutions **no longer optimal** but at most  $W$  times from optimal
- For large problems, only feasible optimal algorithms are **linear-Memory** algorithms such as IDA\* and B&B
- Linear-memory algorithms often use **too little memory** and may visit fragments of search space many times
- It's common to extend IDA\* in practice with so-called **transposition tables**
- Optimal solutions have been reported to problems with **huge state spaces** such 24-puzzle, Rubik's cube, and Sokoban (Korf, Schaeffer); e.g.  $|S| > 10^{25}$

- Key issues: heuristics, representation, symmetries, use of memory, branching rules, . . .

# Heuristics: where they come from?

- General idea: heuristic functions obtained as **optimal cost functions** of relaxed problems
- Examples:
  - *Manhattan distance in N-puzzle*
  - *Euclidean Distance in Routing Finding*
  - *Spanning Tree in Traveling Salesman Problem*
  - *Shortest Path in Job Shop Scheduling*
- Yet
  - how to get and solve suitable relaxations?
  - how to get heuristics automatically?

We'll get back to this in Planning . . .

# Summary

- AI modeling and problem solving; three main ingredients
  - **representation languages** for describing problems
  - **mathematical models** for making sense of problems
  - **algorithms** for solving models
- So far we've focused on **State Models** and **Algorithms** for solving them
- No **general problem solving** yet, but specialized solvers for problems that map into state models
- **Next: Planning**
  - incorporation of languages for representing problems
  - automatic extraction of heuristics

# HW 1: Optimal Solver for 15-puzzle

- Algorithm: IDA\*,
- Heuristic: Sum of Manh. Distances (don't count blank)
- Implementation: need to expand  $> 10^6$  nodes/second
- Hand in Code & Results (Time, Cost, Expanded Nodes)
- Run over benchmarks in Korf, AIJ Vol 27, pp 106-7, 1985
- Experiment with A\*, WA\*, and WIDA\*

N	Initial State $s_0$	$h(s_0)$	$h^*(s_0)$	Nodes
1	14 13 15 7 11 12 9 5 6 0 2 1 4 8 10 3	41	57	276M
2	13 5 4 10 9 12 8 14 2 3 7 1 0 15 11 6	43	55	15M
3	14 7 8 2 13 11 10 4 9 12 5 0 3 6 1 15	41	49	565M
4	5 12 10 7 15 11 14 0 8 2 1 13 3 4 9 6	42	56	62M
...				
14	7 6 8 1 11 5 14 10 3 4 9 13 15 2 0 12	41	59	1369M
...				
88	15 2 12 11 14 13 9 5 1 3 8 7 0 10 6 4	43	65	6009M
...				

-- Nodes: Number of generated nodes (in Millions)

- Goal state is 0 1 2 . . . 15, where first four numbers, refer to first row, second four numbers to second row, etc.

# Selected Bibliography for Current Research

- R. Korf. Finding Optimal Solutions to the Twenty-Four Puzzle. Proc. of AAI-96, pp 1202--1207
- J Culberson and J Schaeffer. Pattern Databases , Computational Intelligence, Vol. 14, No. 4, pp. 318-334, 1998.
- R. Korf Finding optimal solutions to Rubik's Cube using pattern databases, Proc. AAI-97, pp. 700-705.
- R. Holte and I. Hernadvolgyi, A Space-Time Tradeoff for Memory-Based Heuristics, Proceedings AAI-99, pp 704--709.
- A Junghanns and J Schaeffer. Domain-Dependent Single-Agent Search Enhancements , IJCAI, pp. 570-575, 1999.
- Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing General Single-Agent Search Methods Using Domain Knowledge, Artificial Intelligence. 2001.
- A. Felner and R. Korf. Disjoint pattern database heuristics, AIJ, Vol. 134, No. 1-2, Jan. 2002, pp. 9-22.
- R. Korf. Recent progress on the design and analysis of admissible heuristic functions, Proceedings AAI-2000, pp. 1165-1170.