

# CS4618 Artificial Intelligence I

Today: Heuristics for  $A^*$  Search  
Adversarial Search ( $\cong$  Games)

Thomas Jansen

October 10<sup>th</sup>



# Plans for Today

- 1 Introduction  
Overview
- 2 Generating Heuristic Functions  
Relaxations  
Subproblems and Learning
- 3 Games  
Motivation
- 4 Optimal Game Play  
Optimal Decisions  
Alpha-Beta-Pruning
- 5 Summary  
Summary & Take Home Message

# Generating Heuristic Functions

**Remember**  $A^*$  search  
 $\hat{=}$  best-first search guided by **evaluation function**  $f = g + h$   
( $g$  = path cost,  $h$  = heuristic)  
**optimal** with consistent heuristic (triangle inequality)  
prefer **dominating** heuristics (yielding larger values)

**Observation** good heuristic functions make a huge difference

How do we find good heuristic functions?

- constructing heuristics from relaxations
- constructing heuristics from subproblems
- learning heuristics from experience

# Relaxations

## Definition (Relaxation)

For a given problem a **relaxation** of the problem is created by removing restrictions on the actions.

What happens in the state space?

**Observation** edges are added

$\rightsquigarrow$  state space of relaxation is a super-graph

$\hat{=}$  state space of problem is a sub-graph

**Observation** solutions of problems are solutions for relaxations  
**but** cost of solution in relaxation may be **reduced**

**Consequence** values of solutions of relaxations are **admissible** heuristics

**Observation** values of exact solution obey triangle inequality

**Consequence** values of solutions of relaxations are **consistent** heuristics

## Heuristics from Relaxations

**Remember** values of solutions of relaxations are **consistent** heuristics

Does it make sense to use such heuristics?

**Observation** **necessary** compute optimal value for relaxation  
much more efficiently than for original problem

**Example** modifications of 8-puzzle

**Remember** **rule for 8-puzzle**

action can move tile from A to B if and only if

- ① A is adjacent to B and
- ② B is empty.

No other tile is moved.

## Relaxations of the 8-Puzzle

Remember rule for 8-puzzle

action can move tile from A to B if and only if

① A is adjacent to B and

② B is empty.

No other tile is moved.

Relaxation 1 action can move tile from A to B  
in any case

tile in B is moved either to A or the empty space  
avoiding its correct position

Relaxation 2 action can move tile from A to B if and only if  
A is adjacent to B

tile in B is moved either to A or the empty space  
avoiding its correct position

Relaxation 3 action can move tile from A to B if and only if  
B is empty

## Relaxations 1 and 2

**Relaxation 1** action can move tile from A to B in any case  
tile in B is moved either to A or the empty space  
avoiding its correct position

### Observation

- optimal value  $\hat{=}$  number of misplaced tiles
- optimal value can be easily computed ✓
- Relaxation 1 generates  $h_1$

**Relaxation 2** action can move tile from A to B if and only if  
A is adjacent to B  
tile in B is moved either to A or the empty space  
avoiding its correct position

### Observation

- optimal value  $\hat{=}$  sum of Manhattan distances over all tiles
- optimal value can be easily computed ✓
- Relaxation 2 generates  $h_2$

## Relaxation 3

**Relaxation 3** action can move tile from A to B if and only if B is empty

How can we compute  $h_3$ ?

**Algorithm** If blank is correctly placed  
then move any incorrectly placed tile to blank  
else move incorrectly placed tile to its correct location.

**Observation**  $\forall v: h_1(v) \leq h_3(v) \leq h_1(v) + 1$

Is  $h_3(v) > h_2(v)$  possible?

Consider

$$v = \begin{array}{|c|c|c|} \hline & 1 & 2 \\ \hline 3 & 7 & 5 \\ \hline 6 & 4 & 8 \\ \hline \end{array}$$

$$h_1(v) = 2$$

$$h_2(v) = 2$$

$$h_3(v) = 3$$

**Remark** Relaxation 3 generates **Gaschnig's heuristic**  
J. Gaschnig (1979): Performance Measurement and Analysis  
of Certain Search Algorithms. PhD Thesis, CMU.

## On Having Many Heuristics

**Remember** for 8-puzzle we have three heuristics

- $h_1$ : number of misplaced tiles
- $h_2$ : sum of Manhattan distances
- $h_3$ : Gaschnig's heuristic

How can we choose a best?

**Remember**  $\forall v: h_3(v) \geq h_1(v)$   
 $h_3$  **better** than  $h_1$  since it **dominates** it

**Idea** arbitrary heuristics  $h_1, h_2, \dots, h_n$  can be **combined**  
 $h(v) := \max\{h_1(v), h_2(v), \dots, h_n(v)\}$

### Observations

- $h$  dominates  $h_1, h_2, \dots, h_n$
- all  $h_i$  admissible  $\Rightarrow h$  admissible


Is  $h$  also consistent?

## On the Consistency of $h$

**Remember** consistent heuristics  $h_1, h_2, \dots, h_n$   
 $h(v) := \max\{h_1(v), h_2(v), \dots, h_n(v)\}$

**Assume**  $h$  **not** consistent  
 $\exists v, a: h(v) > \text{cost}(a) + h(v')$  with  $v' = \text{transition-model}(v, a)$

### Observations

- $\exists i, j: h(v) = h_i(v), h(v') = h_j(v')$  by definition of  $h$
- $i \neq j$  since  $h_i$  consistent
- $h_i(v) \geq h_j(v), h_j(v') \geq h_i(v')$  by definition of  $h$
- $h_i(v) \leq \text{cost}(a) + h_i(v'), h_j(v) \leq \text{cost}(a) + h_j(v')$  since  $h_i$  and  $h_j$  consistent
- $h_i(v') > h_j(v')$  since  $h_i(v) \leq \text{cost}(a) + h_i(v')$  and  $h_i(v) > \text{cost}(a) + h_j(v')$  

**Thus**  $h$  consistent 

## Heuristics From Subproblems

**Idea** define subproblem ( $\approx$  smaller problem)  
use cost of solving the subproblem as heuristic

**Example** for 8-puzzle any initial state like

*	2	4
*		*
*	3	1

and goal state

	1	2
3	4	*
*	*	*

**Idea** in **preprocessing** compute solutions for each subproblem  
and store in database, used for computing  $h$

**Observation** preprocessing **costly**  
**worthwhile** if many instances solved afterwards

## Combining Heuristics from Subproblems

Consider for 8-puzzle any initial state like

*	2	4
*		*
*	3	1

and goal state

	1	2
3	4	*
*	*	*

Alternative for 8-puzzle any initial state like

*	8	6
*		*
*	7	5

and goal state

	*	*
*	*	5
6	7	8

Can we combine these?



## Combining 1–4 Subproblem Heuristic with 5–8 Subproblem Heuristic

**Remember** heuristic based on all 1–4 initial states  
 heuristic based on all 5–8 initial states

Can these two be combined?

### Observations

- **incorrect** ( $\hat{=}$  not admissible) if taking into account all moves (since moving considered tiles also moves others)
- **correct** if only moves of considered tiles are counted  
 $\rightsquigarrow$  very good heuristics

# Learning Heuristics

**An Outlook** on 'self-improving' agents

**Consider** agent solving problem with A\* search with heuristic  $h = 0$

**Observation** solution yields exact value for initial state  
and a few states 'on the way'

**Idea** store these in database as heuristic information  
for subsequent runs

**Problem** only **very few** states with solutions encountered  
unlikely to be helpful

**Improvement** use **some way** of extending knowledge  
using values for **similar** states

# Search

## We know

- uninformed search
  - BFS
  - DFS
- informed search
  - Greedy Best First Search
  - A\* Search (and variants)

**Now** search for a special kind of application 'games'  
adversarial search

## 'Classical' Game Theory

Consider **two-player zero-sum games**.

Game can be described by **pay-off matrix**.

**Example: Rock Paper Scissors**

	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0

**row player** aims at maximising.

**column player** aims at minimising.

**Strategies**    **pure**  $\hat{=}$  deterministic

**mixed**  $\hat{=}$  randomised

## Set-Up and Notation

**Central** two-player zero-sum games  
**but** with emphasis on alternating **moves**

### Notation

- two **players**: Max (has first move), Min
- **initial state**:  $S_0$  (game set up)
- **actions**: set of legal moves
- **transition model**:  $\text{Result}(s, a)$
- **terminal test** indicates end of game
- **utility function**:  $\text{Utility}(s, p)$  (of state  $s$  for player  $p$ )

**Remark** relaxation of 'zero-sum game'  
 $\forall$  terminal states  $s$ :  $\text{Utility}(s, \text{Max}) + \text{Utility}(s, \text{Min})$  equal  
 (not necessarily equal to 0)



# Games and Search

## Observation

- **initial state:**  $S_0$  (game set up)
  - **actions:** set of legal moves
  - **transition model:**  $\text{Result}(s, a)$
  - **terminal test** for end of game
- }  $\hat{=}$  given like in a search problem

Thus **game tree**  
in the same way as search tree

# Game Tree

Max

Min

Max

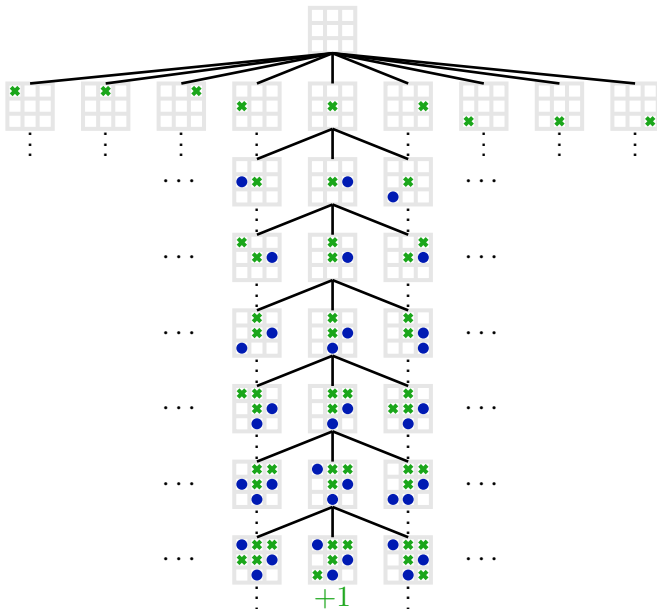
Min

Max

Min

Max

Min



# Minimax Search

**Remember**    **Max** wants to maximise  
                   **Min** wants to minimise  
                   the final outcome

**Definition**    for an arbitrary state  $s$

$$\text{minimax}(s) = \begin{cases} \text{utility}(s) & \text{if terminal-test}(s) \\ \max_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s, a)) & \text{if player}(s) = \text{Max} \\ \min_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s, a)) & \text{if player}(s) = \text{Min} \end{cases}$$

**Example**    Take( $n, k$ )

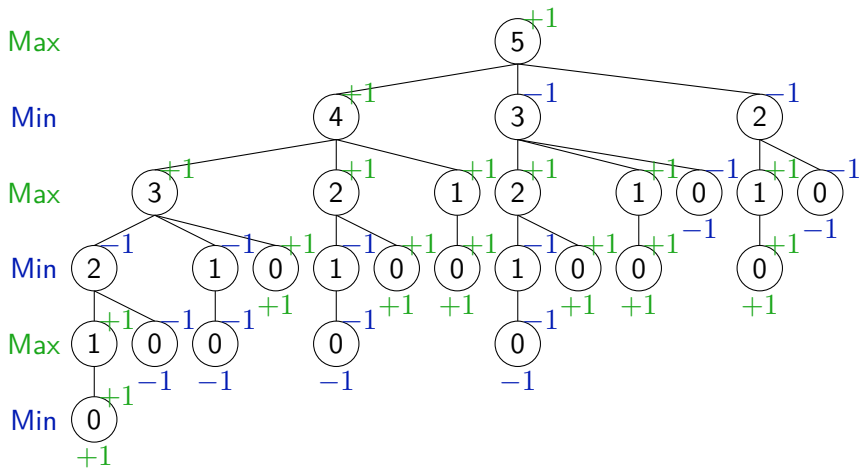
Start with  $n$ . ( $s_0 = n$ )

Each player takes  $i \in \{1, 2, \dots, k\}$

leading from state  $s = j$  to  $j - i$  with  $j - i \geq 0$ .

First player to reach 0 wins.

## Take(5, 3)



# Minimax Algorithm

## Computing Values

function **max-value**( $s$ ) returns utility value

1. if terminal-test( $s$ ) then return utility( $s$ )
2.  $v = -\infty$
3. for each  $a$  in actions( $s$ ) do
4.      $v = \max\{v, \text{min-value}(\text{result}(s, a))\}$
5. return  $v$

---

function **min-value**( $s$ ) returns utility value

1. if terminal-test( $s$ ) then return utility( $s$ )
2.  $v = -\infty$
3. for each  $a$  in actions( $s$ ) do
4.      $v = \min\{v, \text{max-value}(\text{result}(s, a))\}$
5. return  $v$

**Decision** for Max

return  $\arg \max_{a \in \text{actions}(s)} \text{min-value}(\text{result}(s, a))$

# About Minimax Search

## Observations

- minimax algorithm guarantees optimal choice of Max if Min plays optimally
- explores complete game tree
  - ↪ for depth  $m$  and  $b$  actions per state
    - size  $b^m$  ( $\hat{=}$  time complexity)
    - depth  $m$  ( $\hat{=}$  space complexity)

time complexity **completely impractical**

- can be generalised to multi-player games
  - use vector  $(u_1, u_2, \dots, u_k)$  for  $k$  players at each node
  - let player  $i$  select node that maximises  $i$ -th component

## Remarks

- in multi-player games other effects important (**cooperation**)
- in games which are **not** zero-sum other strategies may be optimal even for only two players

# Improving on Minimax Search

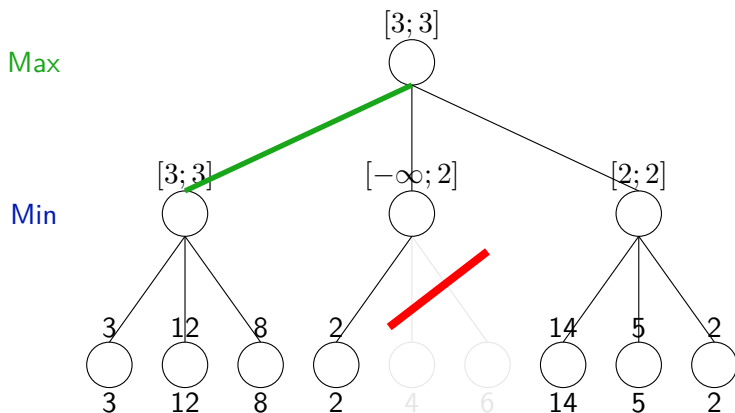
**Remember** minimax search **impractical** due to time complexity

**Goal** **improve** run time  
without sacrificing optimality

**Idea** **pruning**  
≡ do not consider sub-trees that do not influence search result

**Method** **Alpha-Beta Pruning** uses  
lower bound  $\alpha$  for what Max can achieve  
upper bound  $\beta$  for what Min can achieve  
update  $\alpha$  and  $\beta$  whenever possible  
prune max-nodes  $< \alpha$   
prune min-nodes  $> \beta$

# Example Alpha-Beta Pruning



Observation   move ordering matters

# Move Ordering

**Remember** move ordering matters  $\hat{=}$  determines pruning options  
**best** is 'optimal move first'

**Observation** **not really helpful**  
 if optimal move known no need for game tree evaluation

**Heuristic** try **suspected** best move first

## Facts

- **best case** best move first  $\rightsquigarrow \Theta(b^{m/2})$  explored nodes  
 $\hat{=}$  doubles explorable depth
- **worst case** worst move first  $\rightsquigarrow b^m$  explored nodes
- **average case** random move ordering  
 $\rightsquigarrow \Theta(b^{(3/4)m})$  explored nodes
- **empirical** simple heuristics often work well  
 e. g., ordering 'captures, threats, forward, backward' for chess  
 up to factor 2 optimal

# Summary & Take Home Message

## Things to remember

- heuristics for  $A^*$ : relaxation, subproblems, combinations
- games as a special case of search problems
- game tree and minimax-search
- alpha-beta pruning

## Take Home Message

- Heuristics are crucial for the performance of  $A^*$  search.
- Coming up with good heuristics is hard, being inventive helps.
- Standard ways for this are often useful.
- Games are a fascinating and motivating area for AI applications.
- Many games are a AI success story.
- If feasible 'brute force' is better than 'intelligence'.