Heuristic Search

- INTRODUCTION
- HILL-CLIMBING
- DYNAMIC PROGRAMMING
- THE BEST-FIRST SEARCH ALGORITHM
- ADMISSIBILITY, MONOTONICITY, AND INFORMEDNESS
- USING HEURISTICS IN GAMES.

Heuristic

- Heuristic – the study of the methods and rules of discovery and invention
- State Space Heuristics – Formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution
- Apply Heuristics When:
  - A problem is ambiguous and may not have an EXACT solution
    - Eg: Medical diagnosis
  - The computational cost of finding an exact solution is prohibitive.
    - Eg: Chess – where the number of possible states increases exponentially or factorially with the depth of the search.
- Heuristics guide the search along the most “promising” path through space – Informed Search.
- Eliminates unpromising states and their descendants from consideration and finds an acceptable solution

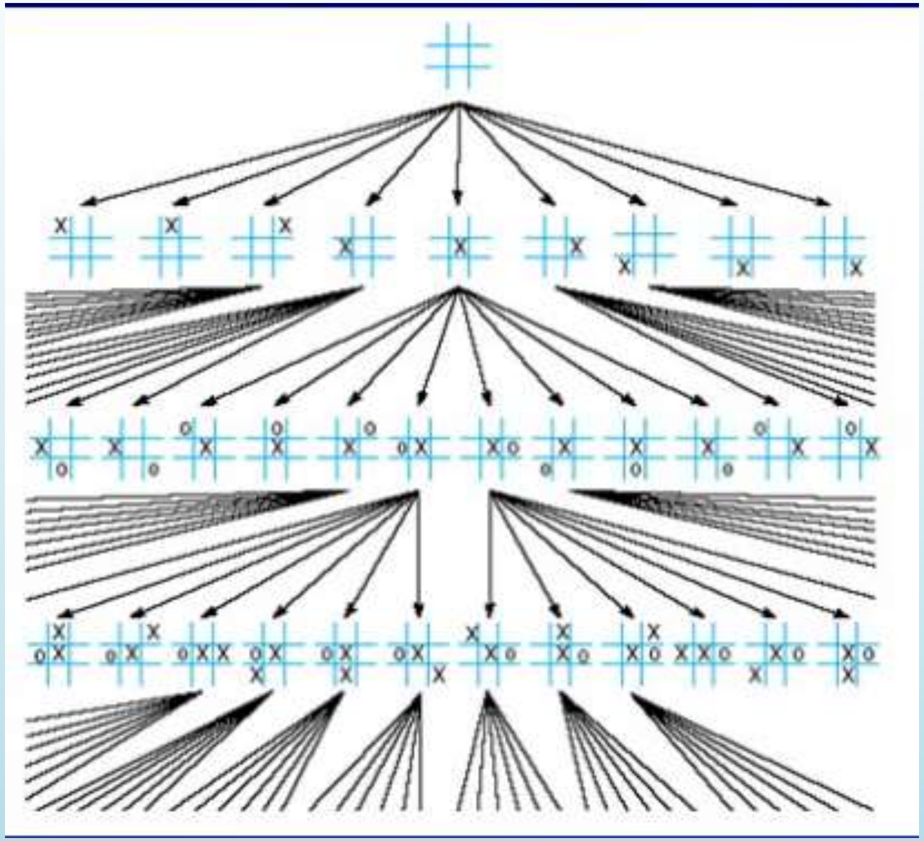
Inherent limitation of heuristic search

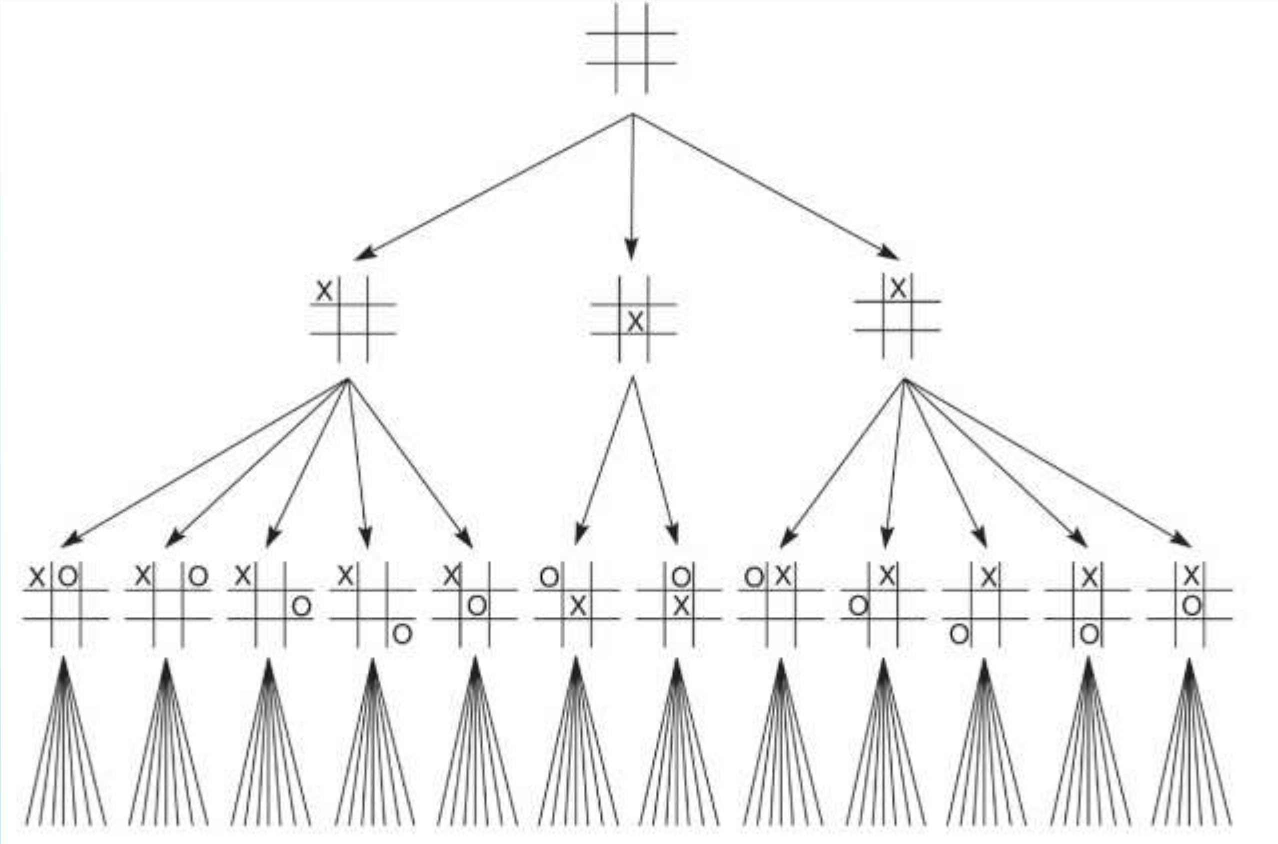
- Heuristic is only an informed guess of the next step to be taken in solving a problem
- Heuristics use limited information, so they are seldom able to predict the exact behavior of the state space farther along in the search
- A heuristic can lead a search algorithm to a suboptimal solution or fail to find any solution at all

Importance of heuristics

- It is not feasible to examine every inference that can be made in a mathematics domain
- Heuristic search is often the only practical answer
- Reduce complex information to a simple and manageable set of choices
- Help people turn an intention into a realized action
- Provide quick and relatively inexpensive feedback to designers

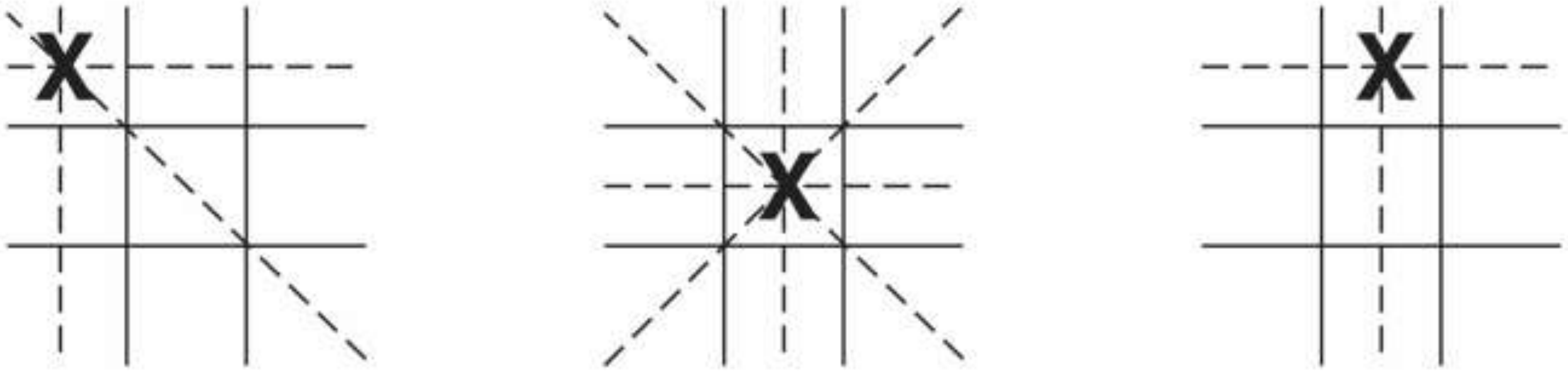
TIC-TAC-TOE





- Total number of states that need to be considered in an exhaustive search at 9x8x7x....or 9!
- Symmetry reductions on the second level further reduce the number of paths through the space to 12x7!

The “most wins” heuristic applied to the first children in tic-tac-toe.

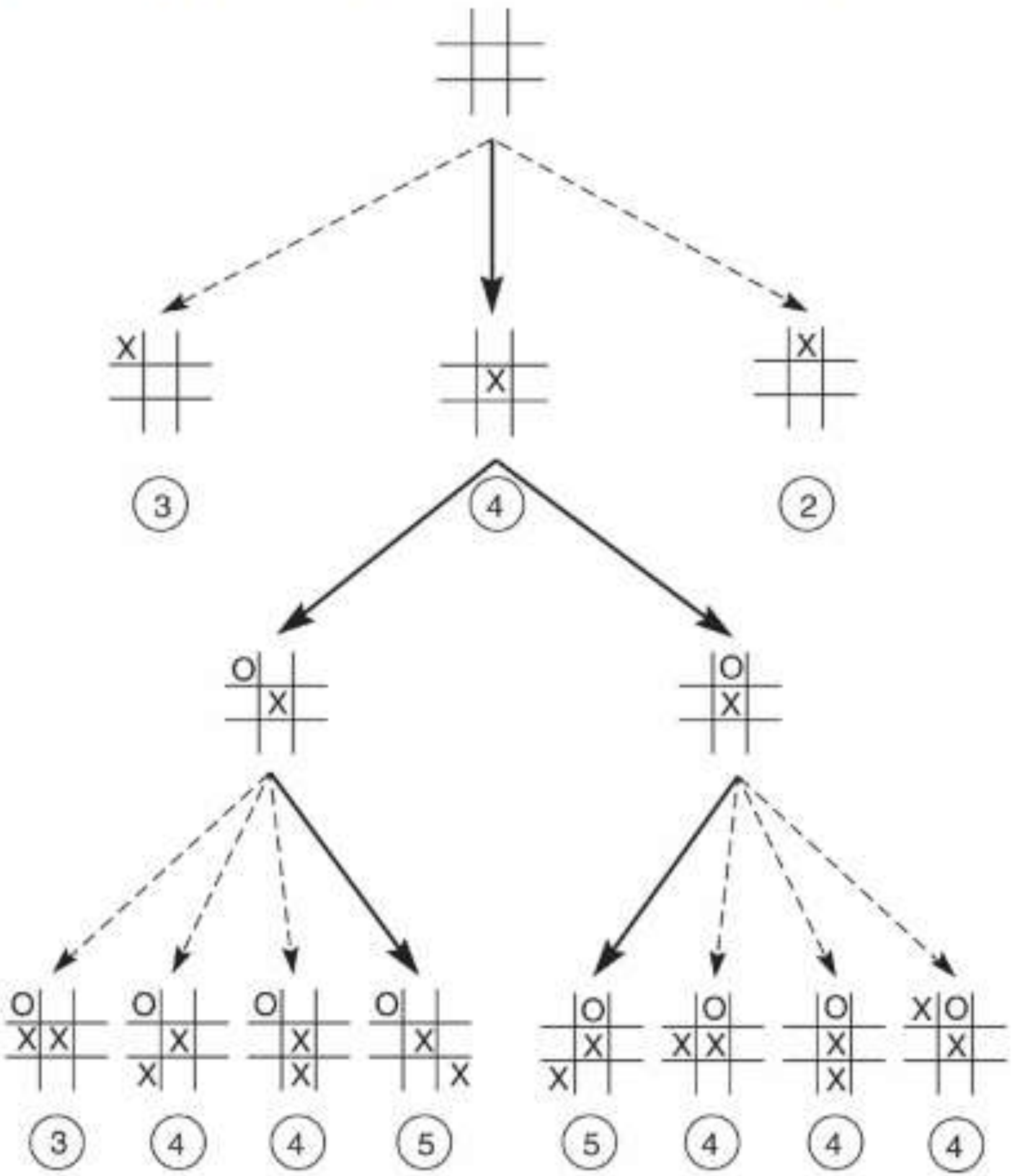


Three wins through a corner square

Four wins through the center square

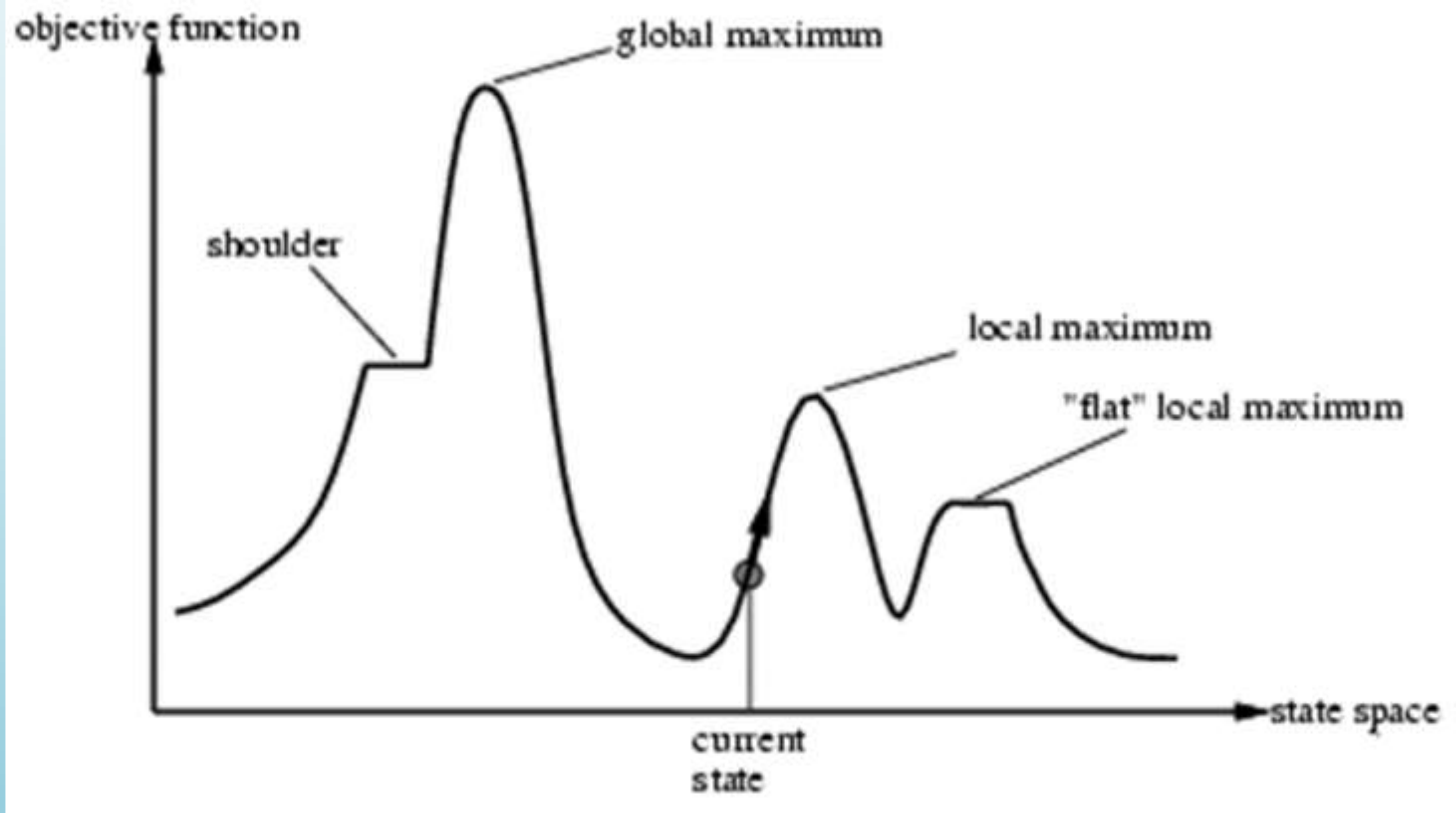
Two wins through a side square

Heuristically reduced state space for tic-tac-toe.

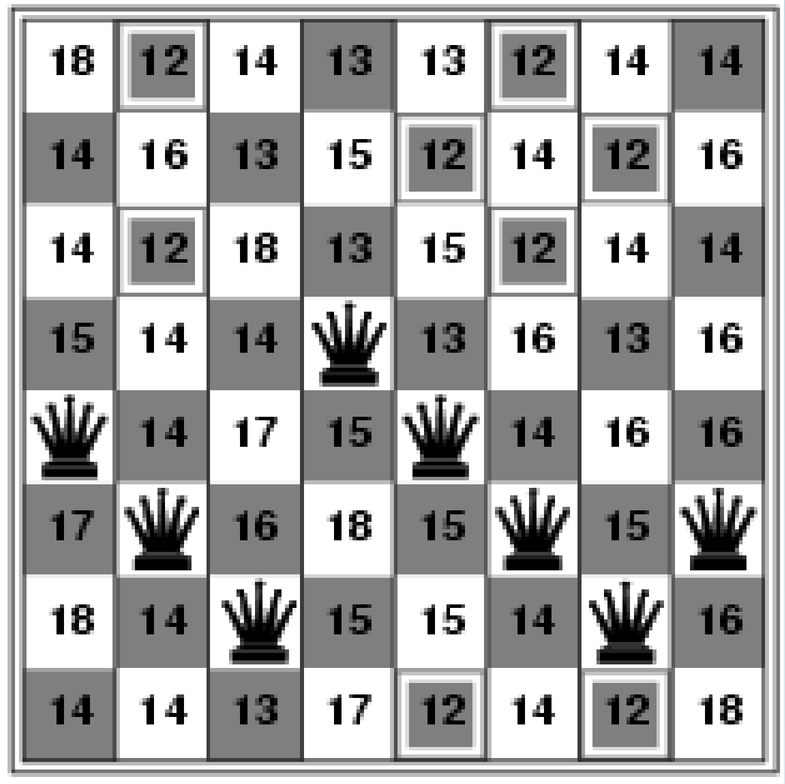


Hill climbing

- Hill climbing strategies expand the current state in the search and evaluate its children
- The best child is selected for further expansion; neither its siblings nor its parent are retained
- Search halts when it reaches a state that is better than any of its children
- Go uphill along the steepest possible path until it can go no farther
- Keeps no history, hence the algorithm cannot recover from failures of its strategy

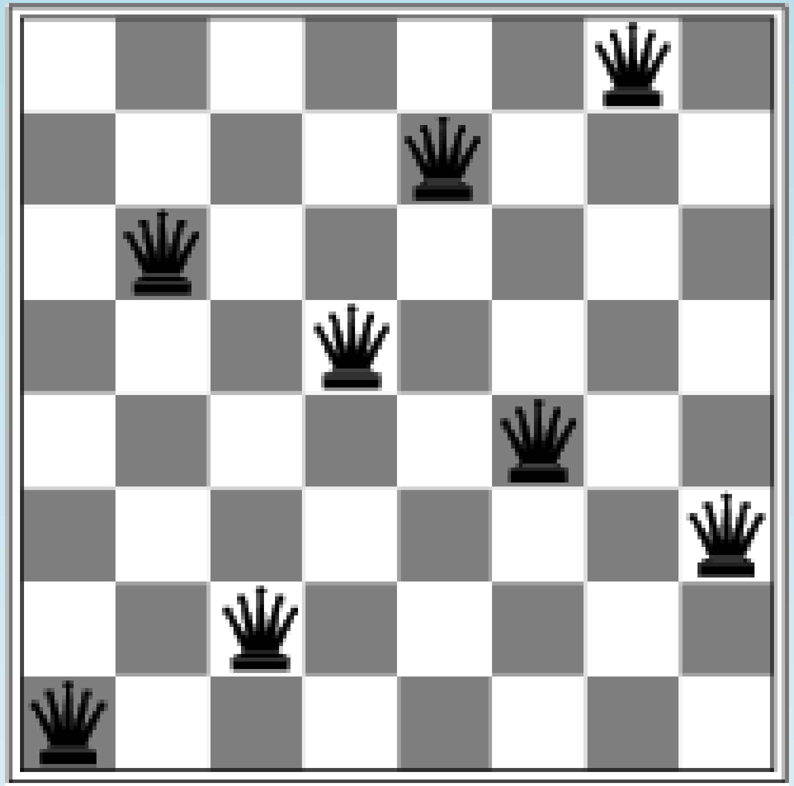


Hill-climbing search: 8-Queens problem



- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Hill-climbing search: 8-Queens problem



- A local minimum with $h = 1$

Dynamic Programming

- Sometimes called the forward-backward. or, when using probabilities, the Viterbi algorithm.
- DP keeps track of and reuses subproblems already searched and solved within the solution of the larger problem.
- Used in
  - Optimal Global Alignment
  - Minimum Edit Distance Between two strings

Optimal Global Alignment

- Small Example
  - String #1
    - BAADDCABDDA
  - String #2
    - BBADCBA
- Rules
  - Cannot change order of respective elements
  - Can have spaces between elements
- Possible solutions
  - How do we figure out optimal solution?

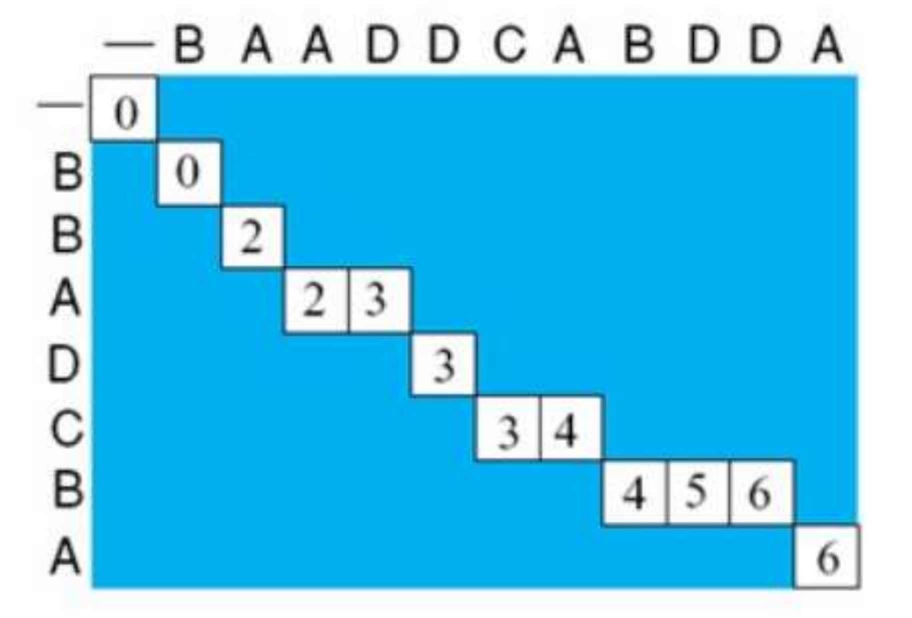
OPTIMAL GLOBAL ALIGNMENT

	—	B	A	A	D	D	C	A	B	D	D	A
—	0	1	2	3	4	5	6	7	8	9	10	11
B	1	0										
B	2											
A	3											
D	4											
C	5											
B	6											
A	7											

The forward stage

- Fill the array from the upper left corner
- The value of (x,y) is a function
    - min cost((x-1,y), (x-1,y-1), (x, y-1))
- If there is a match: Add 0 to (x-1,y-1)
- If there is no match: Add 2 to (x-1,y-1)
- If we shift: Add 1 to the previous column
- If we insert a character: Add 1 to the previous row

	—	B	A	A	D	D	C	A	B	D	D	A
—	0	1	2	3	4	5	6	7	8	9	10	11
B	1	0	1	2	3	4	5	6	7	8	9	10
B	2	1	2	3	4	5	6	7	6	7	8	9
A	3	2	1	2	3	4	5	6	7	8	9	8
D	4	3	2	3	2	3	4	5	6	7	8	9
C	5	4	3	4	3	4	3	4	5	6	7	8
B	6	5	6	5	4	5	4	5	4	5	6	7
A	7	6	5	4	5	6	5	4	5	6	7	6



The backward stage

- Once the matrix is filled
- From the best alignment count, we produce a specific alignment of characters
- Begin at the lower right-hand corner
- Move back through the matrix
- Each step, select one of the immediate state's predecessors (previous diagonal, row, or column)➔ Choose the minimum

BAADDCABDDA
BBADC B A

Min Edit Distance Spell Checker

- What words from our dictionary best approximate a word we do not recognize (misspelled word)
- We need to know the “distance” between two words Minimum edit distance
- The number of insertions, deletions and replacements to turn the **source word** into the **target word**

	—	e	x	e	c	u	t	i	o	n
—	0	1	2	3	4	5	6	7	8	9
i	1	**2**	3	4	5	6	7	8	9	10
n	2	3	**4**	5	6	7	8	9	10	11
t	3	4	**5**	6	7	8	9	10	11	12
e	4	5	6	**5**	**6**	7	8	9	10	11
n	5	6	7	6	7	**8**	9	10	11	12
t	6	7	8	7	8	9	**8**	9	10	11
i	7	8	9	8	9	10	9	**8**	9	10
o	8	9	10	9	10	11	10	9	**8**	9
n	9	10	11	10	11	12	11	10	9	**8**

cost of (x,y) is the minimum of

- Cost of (x-1,y) + insertion
- Cost of (x-1,y-1) + replacement
- Cost of (x, y-1) + deletion

intention – source word

execution – target word

Our “cost”

1 for a character insertion or deletion

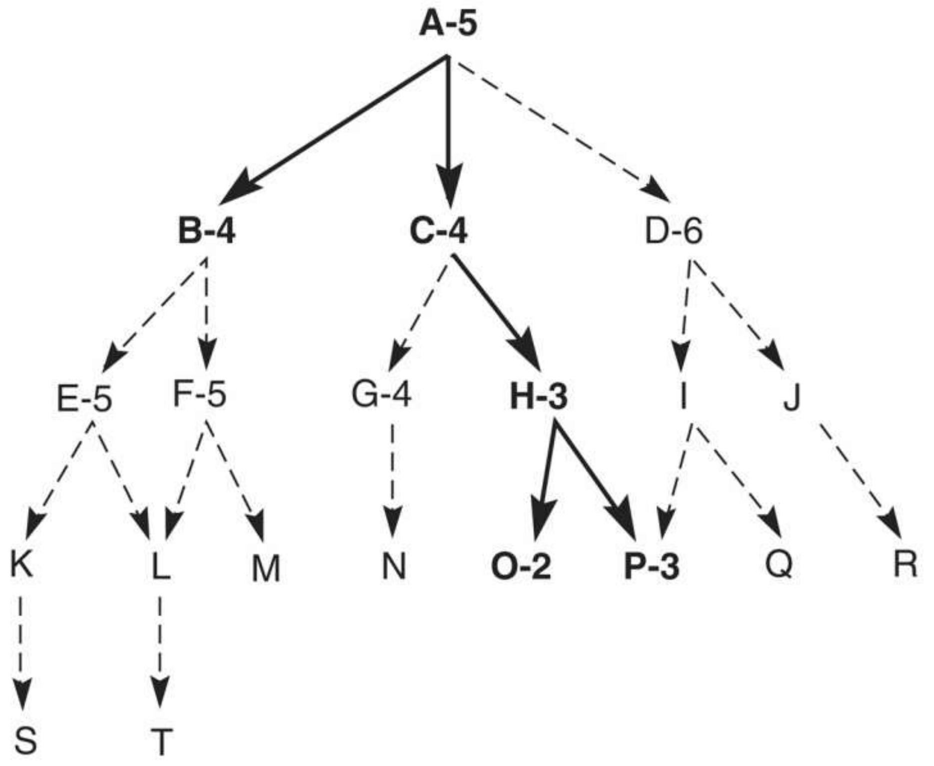
2 for a replacement (deletion + insertion)

intention	
ntention	delete i, cost 1
etention	replace n with e, cost 2
exention	replace t with x, cost 2
exenution	insert u, cost 1
execution	replace n with c, cost 2

Best-First Search Algorithm

- Hill climbing tends to become stuck at local maxima
- If they reach a state that has a better evaluation than any of its children, the algorithm halts
- Hill climbing can be used effectively if the evaluation function is sufficiently informative to avoid local maxima and infinite paths
- Heuristic search requires a more flexible algorithm: this is provided by best-first search, where, with a priority queue, recovery from local maxima is possible

Heuristic search of a hypothetical state space.



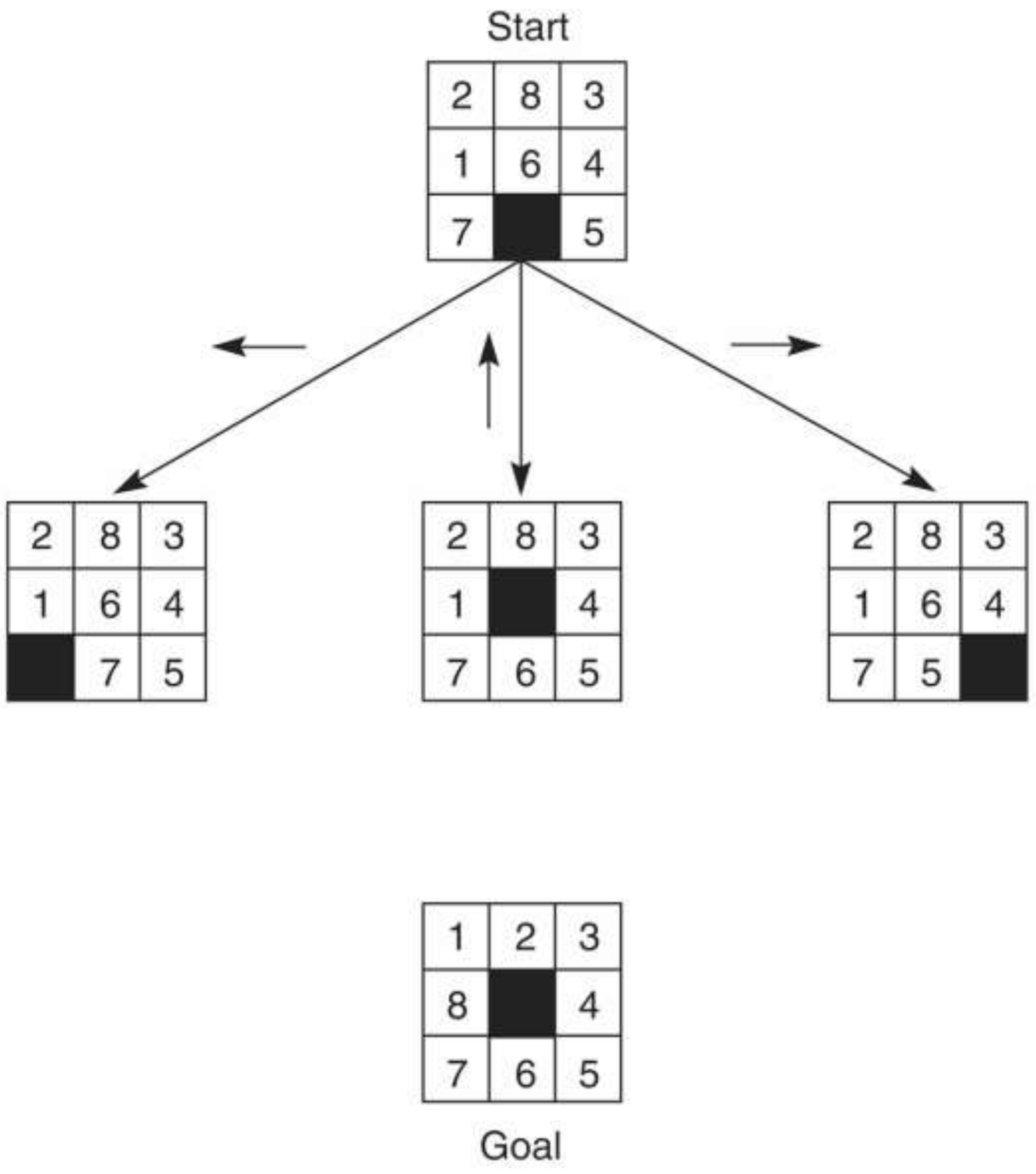
1. **open = [A5]; closed = [ ]**
2. **evaluate A5; open = [B4,C4,D6]; closed = [A5]**
3. **evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]**
4. **evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]**
5. **evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]**
6. **evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]**
7. **evaluate P3; the solution is found!**

```
function best_first_search;

begin
   open := [Start];
   closed := [ ];
   while open ≠ [ ] do
      begin
         remove the leftmost state from open, call it X;
         if X = goal then return the path from Start to X
         else begin
               generate children of X;
               for each child of X do
               case
                  the child is not on open or closed:
                     begin
                        assign the child a heuristic value;
                        add the child to open
                     end;
                  the child is already on open:
                     if the child was reached by a shorter path
                     then give the state on open the shorter path
                  the child is already on closed:
                     if the child was reached by a shorter path then
                        begin
                           remove the state from closed;
                           add the child to open
                        end;
               end;
               put X on closed;
               re-order states on open by heuristic merit (best leftmost)
            end;
return FAIL
```

- The best-first search algorithm always select the most promising state on open for further expansion
- it is using a heuristic that may prove erroneous, it does not abandon all the other states but maintains then on open
- In the event a heuristic leads the search down a path that proves incorrect, the algorithm will eventually retrieve some previously generated, “next best” state from open and shift its focus to another part of the space
- In best-first search, as in depth-first and breadth-first search algorithms, the open list allows backtracking from paths that fail to produce a goal

The start state, first set of moves, and goal state for an 8-puzzle instance.



Informed (Heuristic) Search Strategies

- ***Informed Search*** – a strategy that uses problem-specific knowledge beyond the definition of the problem itself
- ***Best-First Search*** – an algorithm in which a node is selected for expansion based on an evaluation function f(n)
  - Traditionally the node with the lowest evaluation function is selected
  - Choose the node that *appears* to be the best

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"

  →Expand most desirable unexpanded node
- Implementation:

  Order the nodes in decreasing order of desirability
- Special cases:
  - greedy best-first search
  - A^* search

Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)

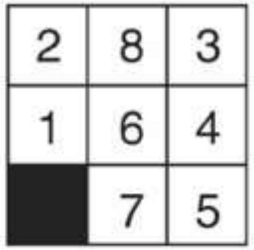
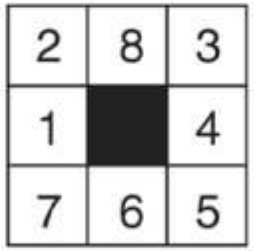
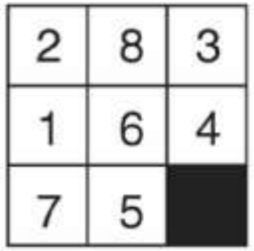
  = estimate of cost from n to *goal*

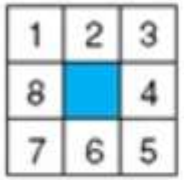
- e.g., $h_{SLD}(n)$ = straight-line distance from n to goal
- Greedy best-first search expands the node that appears to be closest to goal

A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

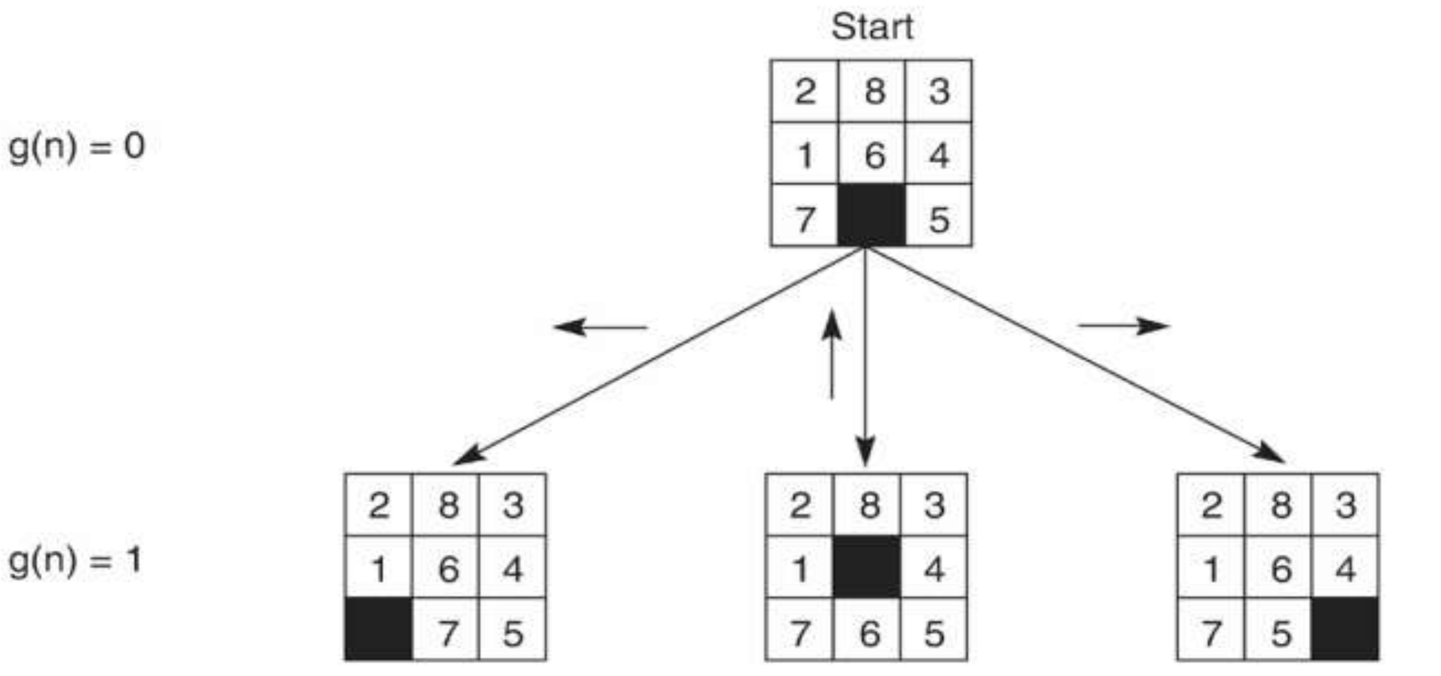
Three heuristics applied to states in the 8-puzzle.

	5	6	0
	3	4	0
	5	6	0
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals



Goal

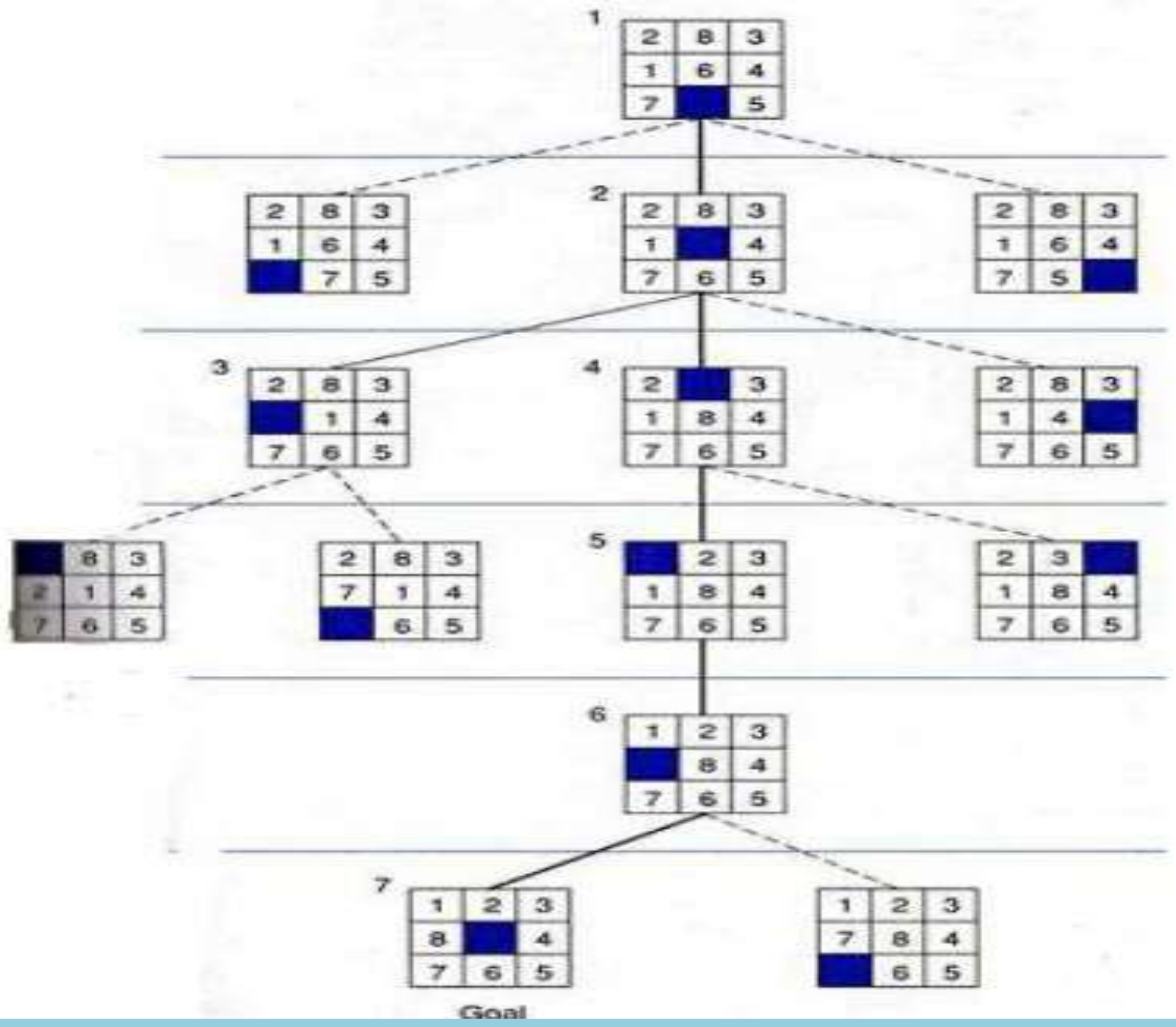
The heuristic **f** applied to states in the 8-puzzle.



Values of f(n) for each state, **6** **4** **6**

where:

$f(n) = g(n) + h(n)$,
$g(n) =$ actual distance from n to the start state, and
$h(n) =$ number of tiles out of place.


1
2
3
4
5
6
7
Goal

Admissibility, Monotonicity & Informedness

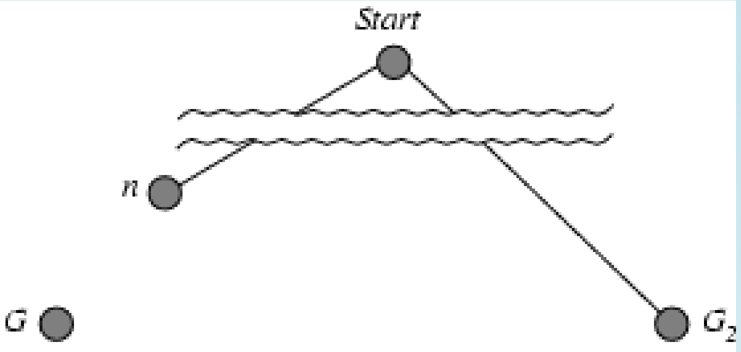
Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n,

  $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

Optimality of A^*

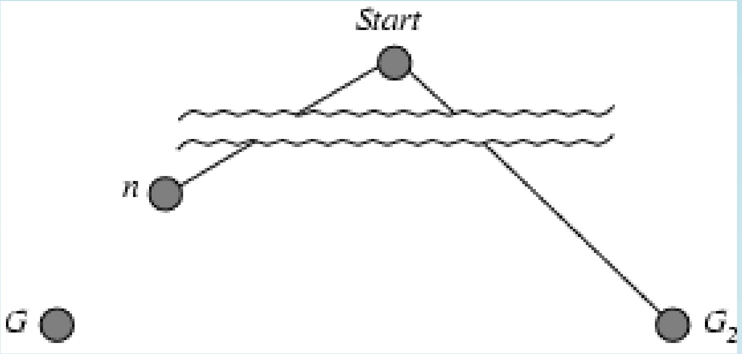
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A*

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



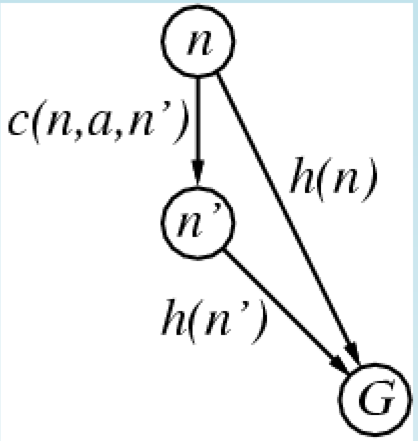
- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

Consistent heuristics / Monotonicity

- A heuristic is consistent/monotone if for every node n, every successor n' of n generated by any action a,

$$h(n) \le c(n,a,n') + h(n')$$



- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) \\ &= f(n) \end{aligned}$$

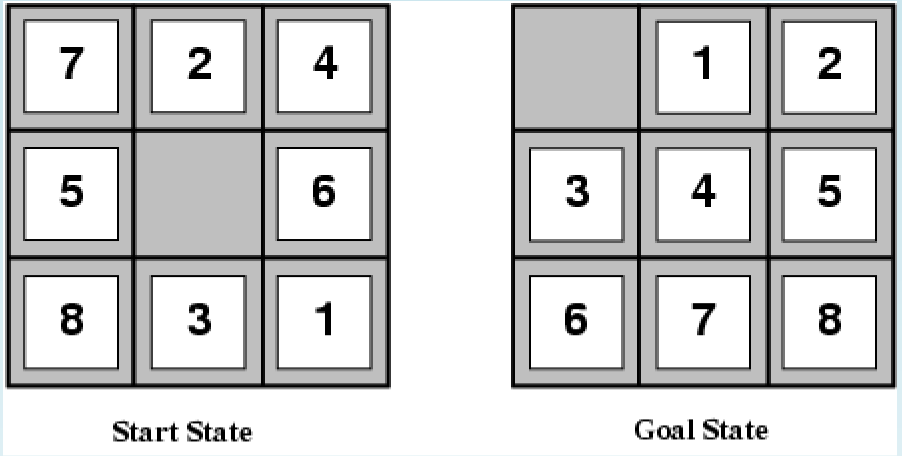
- i.e., $f(n)$ is non-decreasing along any path.
- Theorem: If $h(n)$ is consistent, A* using GRAPH-SEARCH is optimal

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



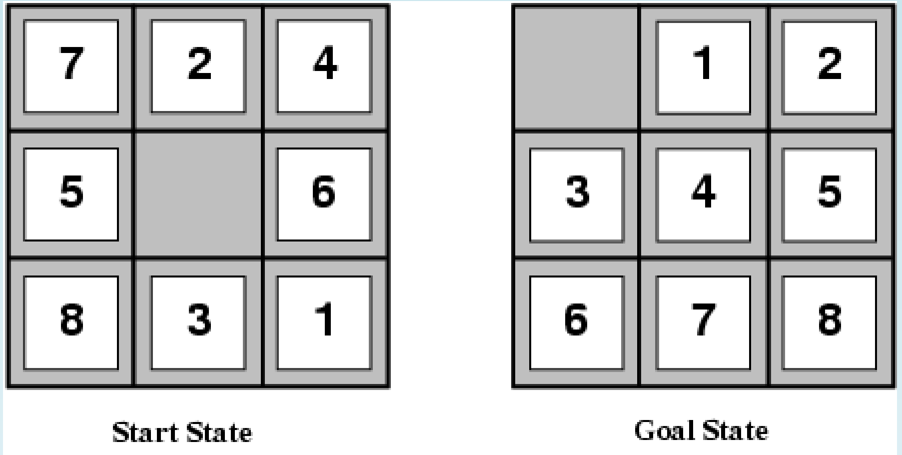
- $h_1(S) = ?$
- $h_2(S) = ?$

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance/ Informedness

- If $h_2(n) \geq h_1(n)$ for all n (both admissible),

  then h_2 dominates h_1 $\rightarrow h_2$ is better for search

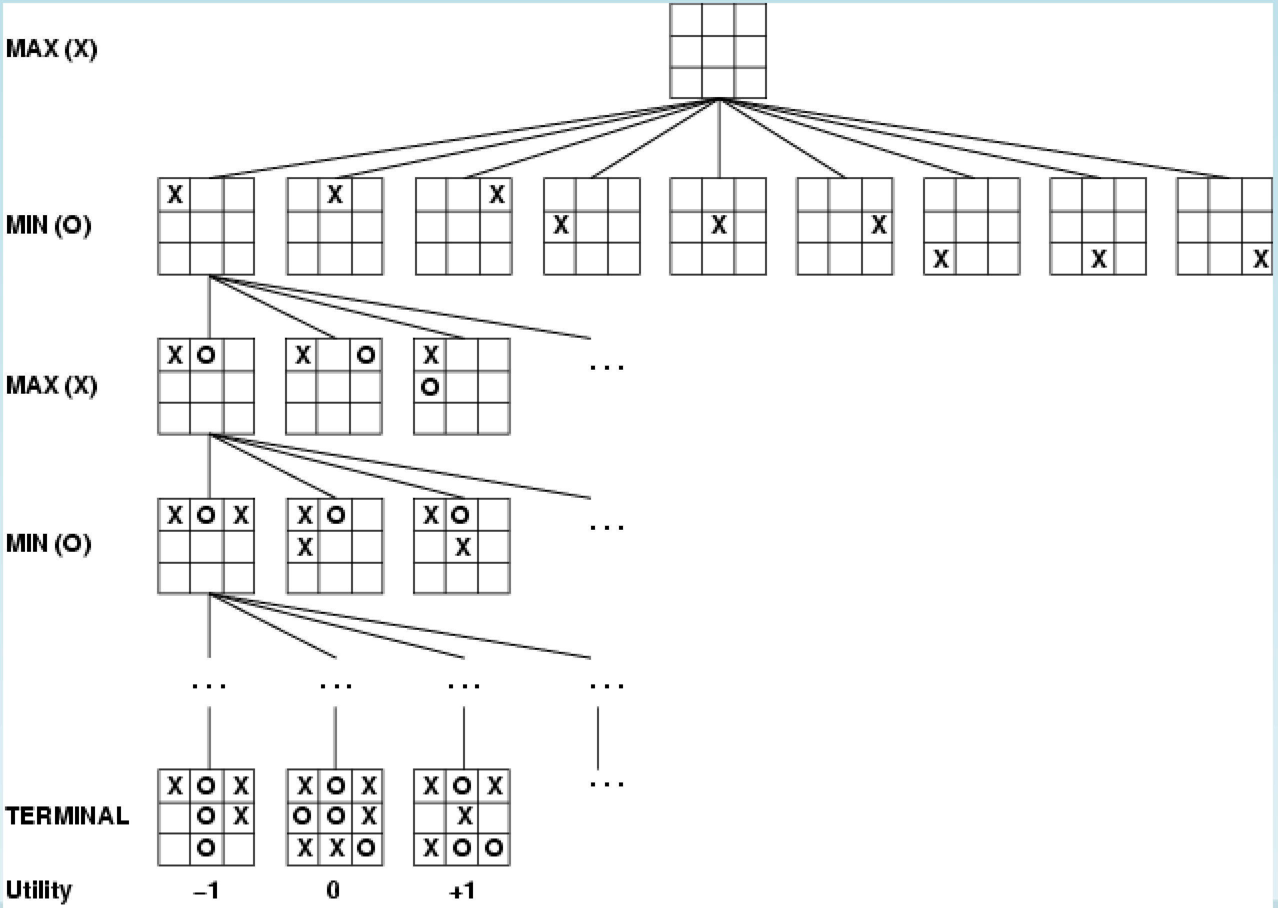
- Typical search costs (average number of nodes expanded):
  - $d=12$, IDS = 3,644,035 nodes
    $A^*(h_1)$ = 227 nodes
    $A^*(h_2)$ = 73 nodes
  - $d=24$ IDS = too many nodes
    $A^*(h_1)$ = 39,135 nodes
    $A^*(h_2)$ = 1,641 nodes

Adversarial Search

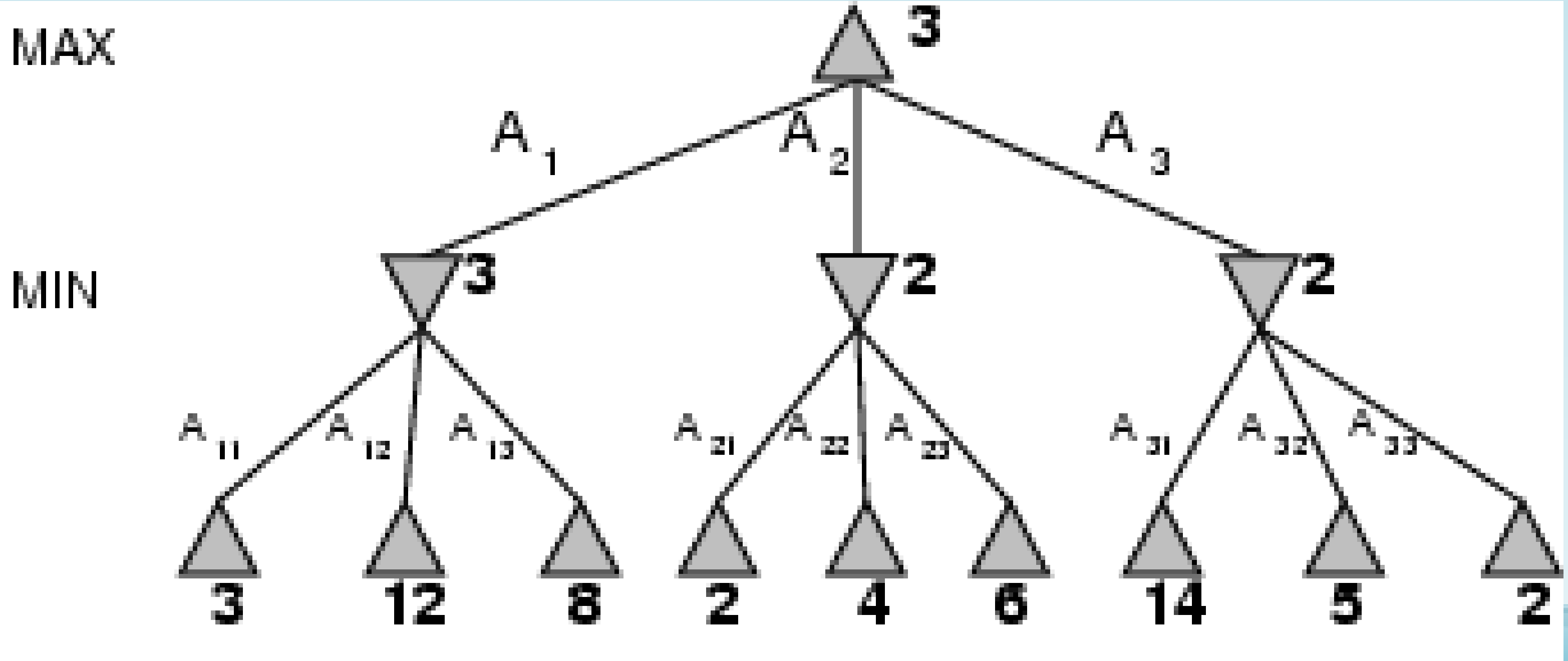
Games vs. search problems

- "Unpredictable" opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Game tree (2-player, deterministic, turns, zero-sum)



Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:
- 

Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*

    $v \leftarrow$ MAX-VALUE(*state*)
    return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*) **returns** *a utility value*

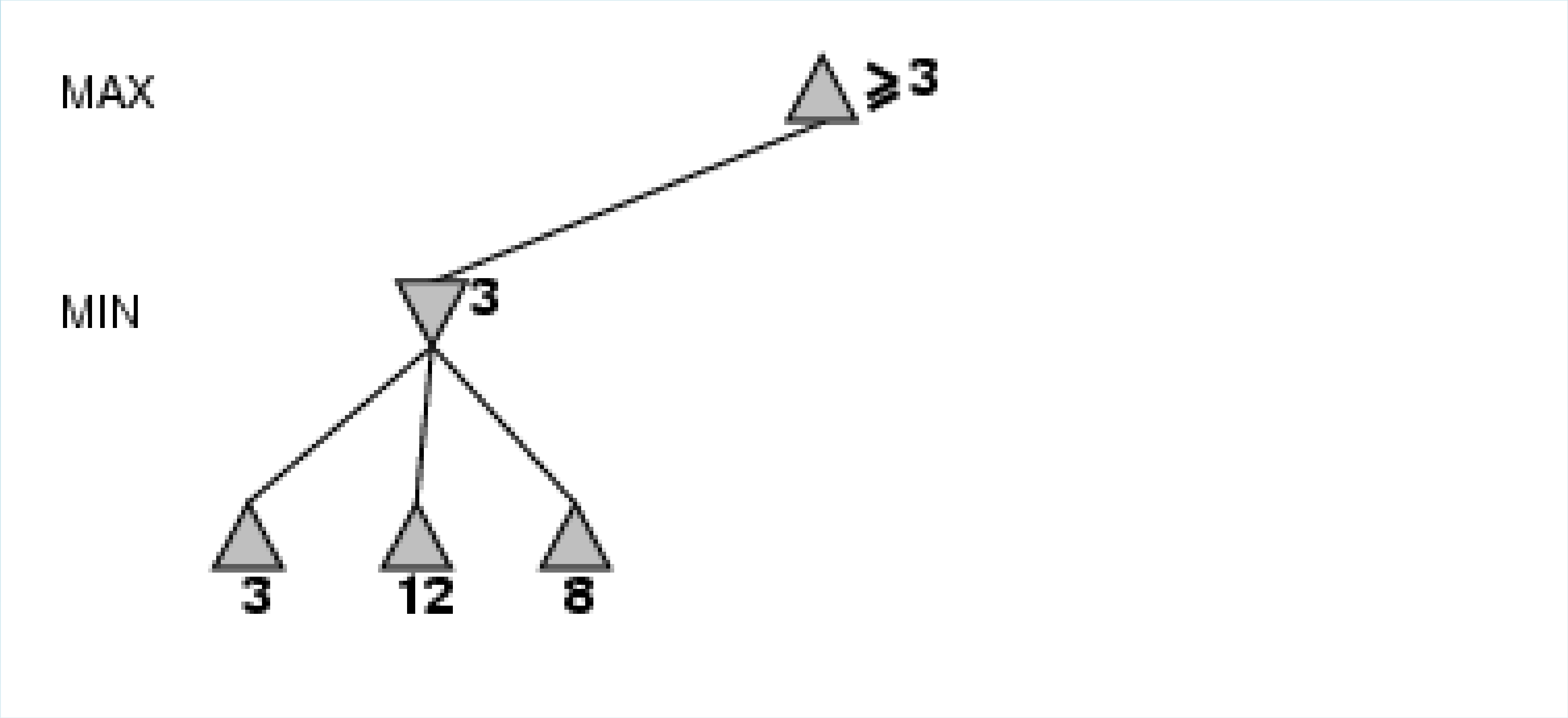
    if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
    $v \leftarrow -\infty$
    for a, s in SUCCESSORS(*state*) **do**
        $v \leftarrow$ MAX(v, MIN-VALUE(s))
    return v

---

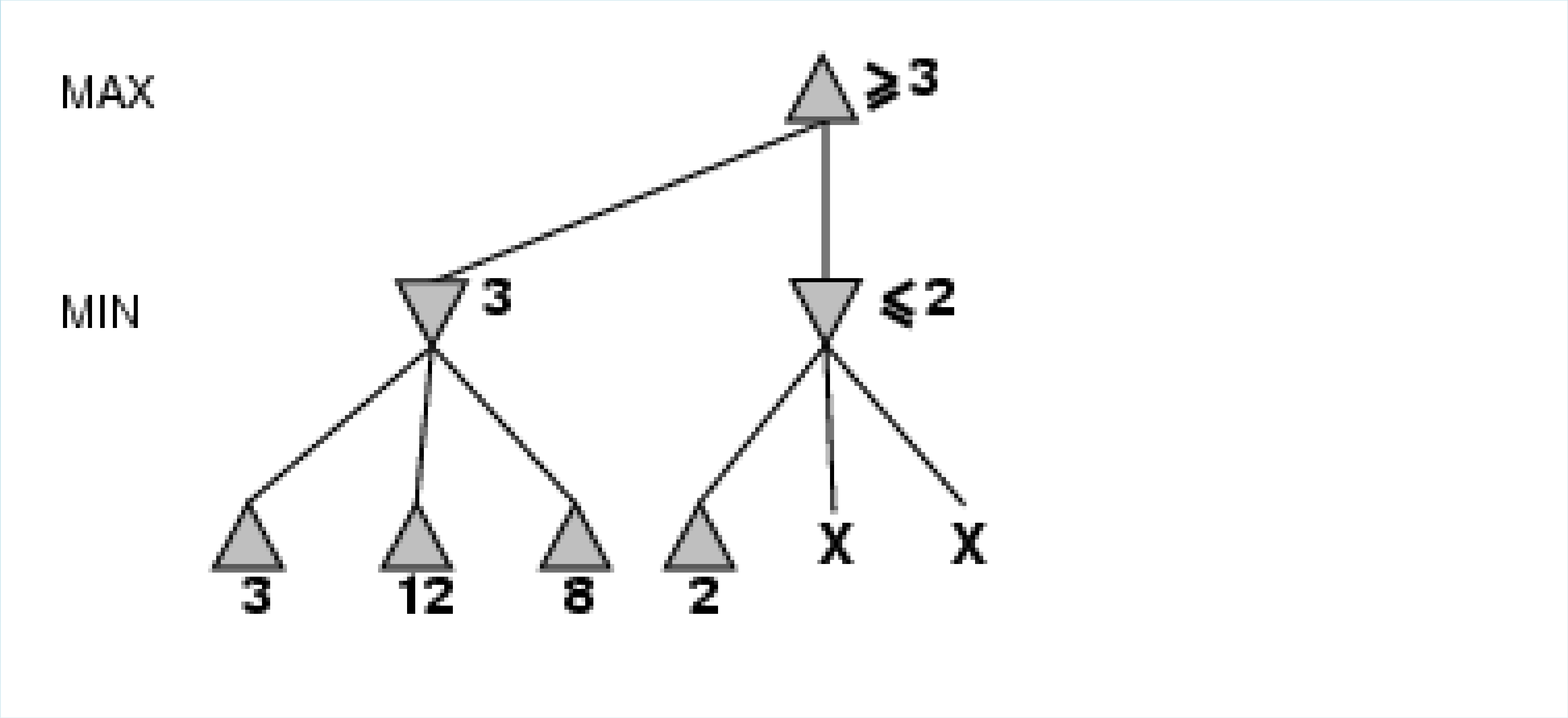
function MIN-VALUE(*state*) **returns** *a utility value*

    if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
    $v \leftarrow \infty$
    for a, s in SUCCESSORS(*state*) **do**
        $v \leftarrow$ MIN(v, MAX-VALUE(s))
    return v

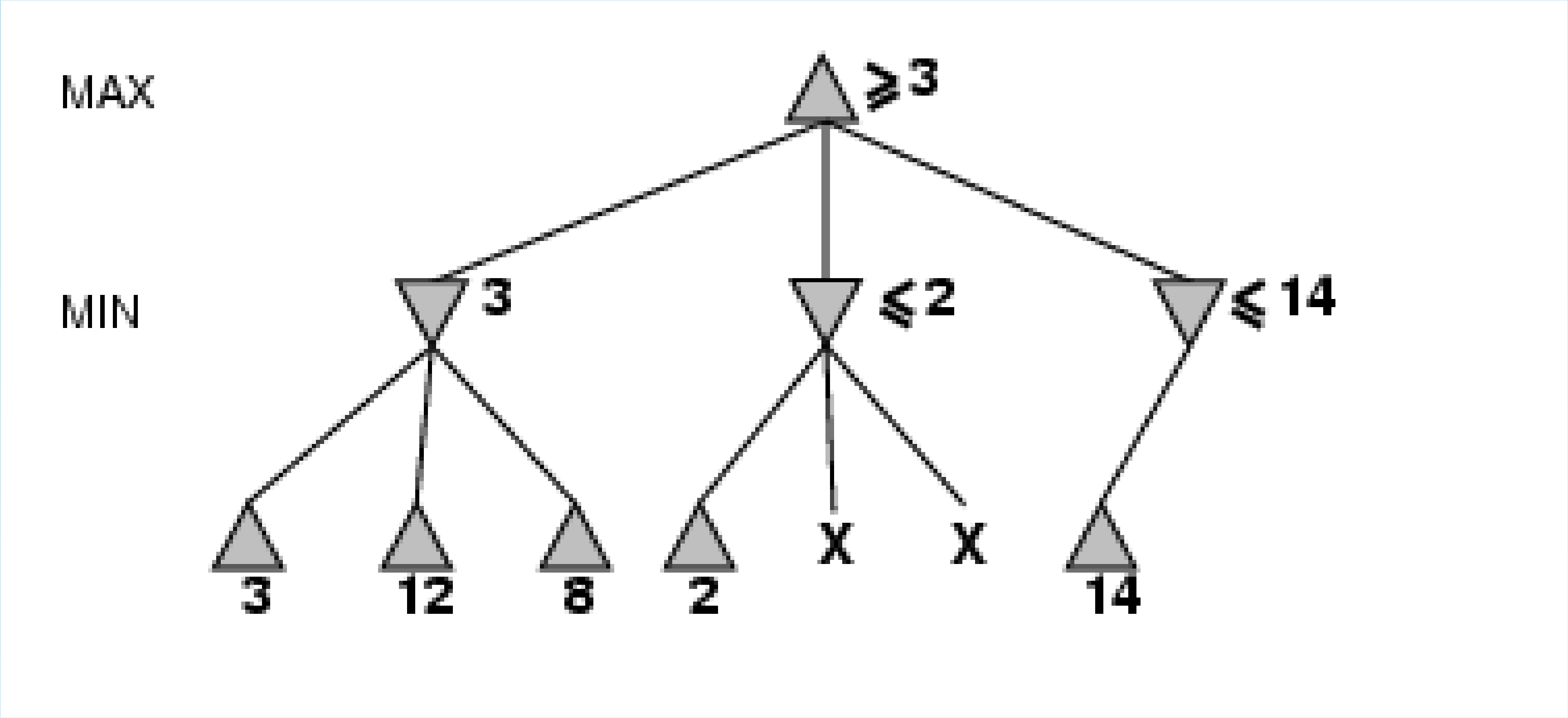
α-β pruning example



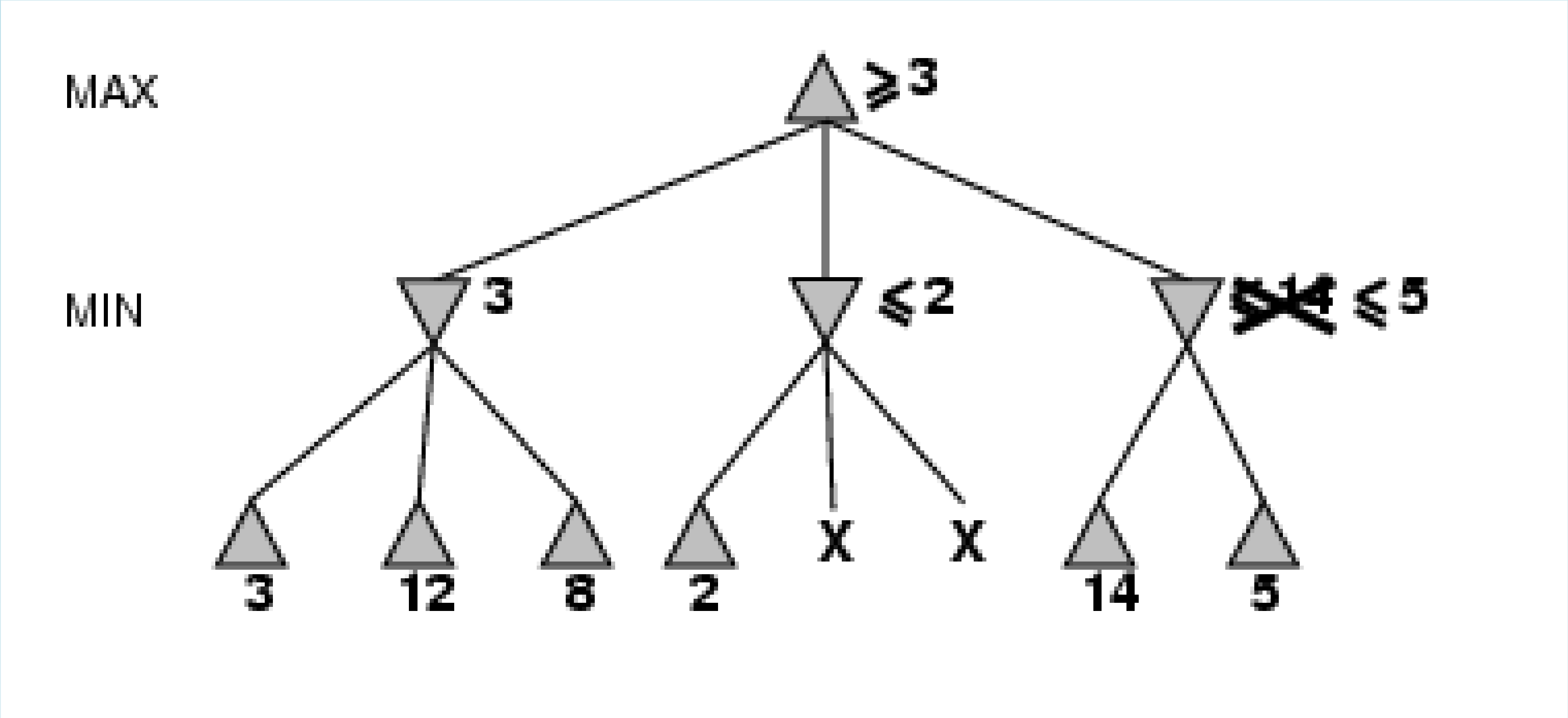
α-β pruning example



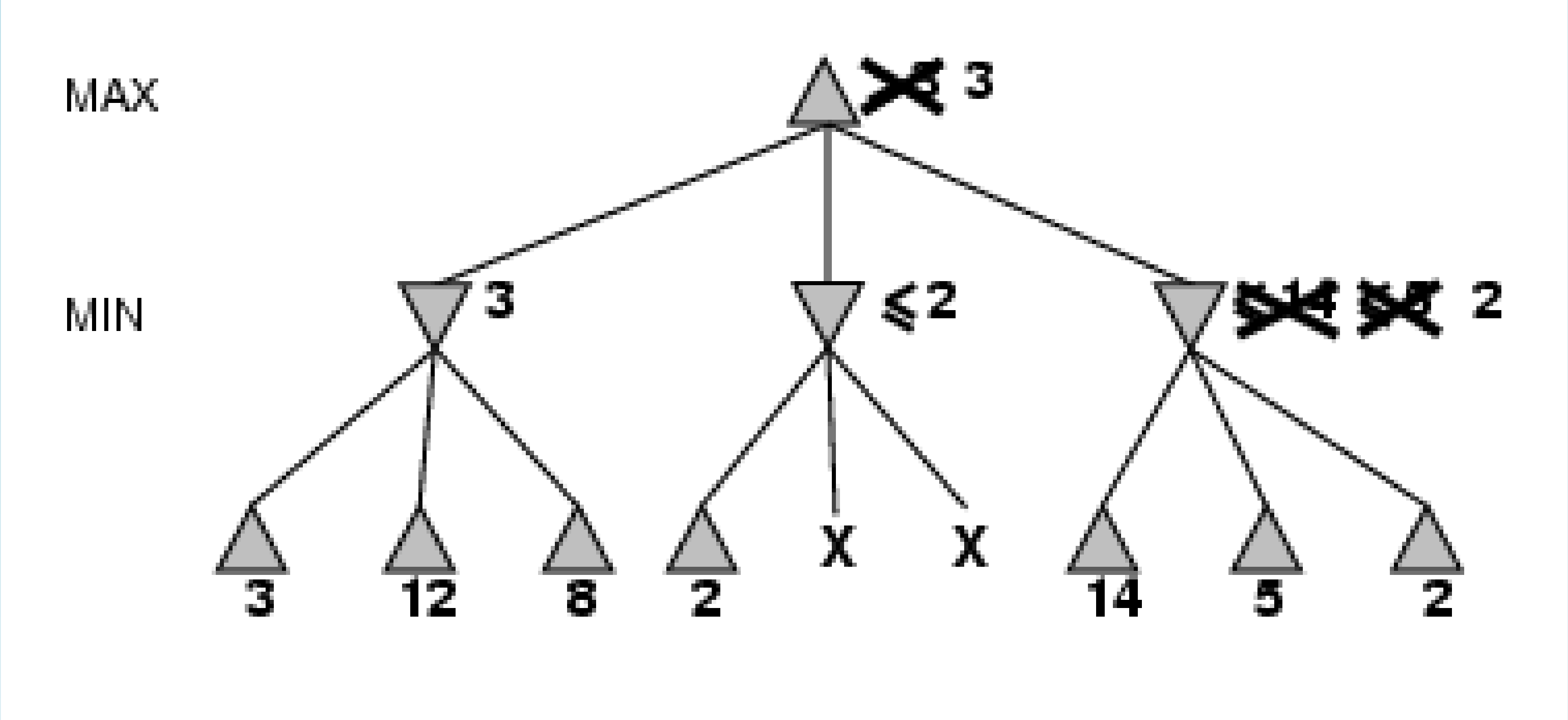
α-β pruning example



α-β pruning example



α-β pruning example



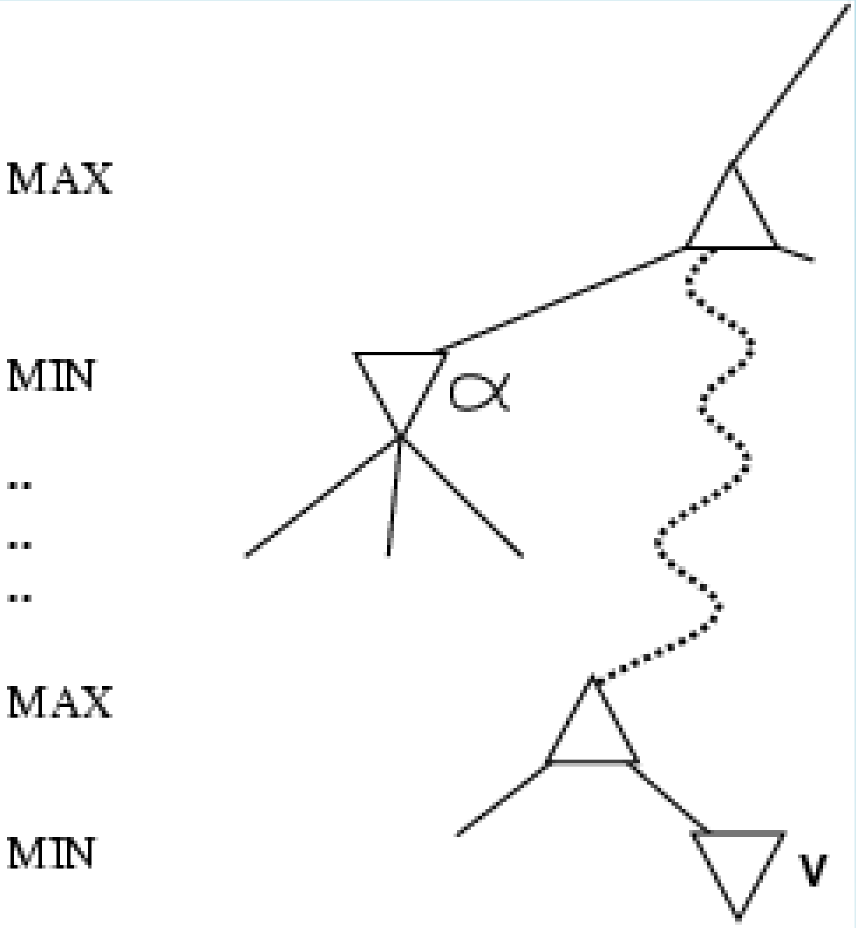
Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$

  → doubles depth of search

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid it
- → prune that branch
- Define β similarly for *min*



The α-β algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Evaluation functions

- For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

Does it work in practice?

$b^m = 10^6$, $b=35 \rightarrow m=4$

4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov