TÉCNICO LISBOA

Artificial Intelligence and Decision Systems

Inteligência Artificial e Sistemas de Decisão – MEEC, Maer,

Luís Manuel Marques Custódio

North tower – ISR / IST

lmmc@isr.ist.utl.pt

Heuristic (informed) search strategies

What is missing in the uninformed search strategies?

A measure of the cost between the current node and the goal node

heuristic function, h(n): given node n returns a cost estimate of the cheapest path from n to a goal node

h(n)=0 iff *n* is a goal node

Greedy best-first search

Strategy: select node with the best heuristic value (smaller h(n))
(implemented as a priority queue on h-values)

What happens if the h(.) function provides the **exact** value of the cost path from n to a goal node?

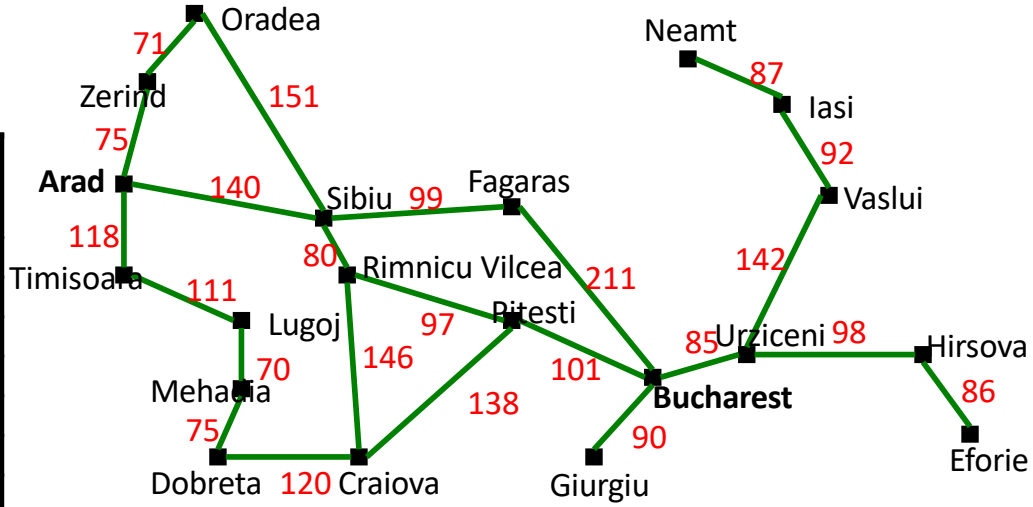
No search is needed

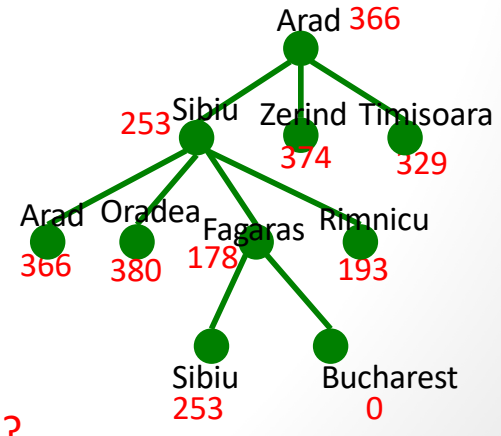
Heuristic (informed) search strategies

Greedy best-first search - select node with the best heuristic value (smaller h(n))

Example: route between Arad and Bucharest

Straight-line distances to Bucharest h(n)	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





Is it the best route?

TÉCNICO LISBOA

Heuristic (informed) search strategies

Greedy best-first search - select node with the best heuristic value (smaller h(n))

Properties**:**

- Complete, if repeated nodes not allowed
- Not optimal

- Time and space complexity: $O(b^m)$

Heuristic (informed) search strategies

Evaluation function, f(n): total (estimate) cost to go from the root node to a goal node passing by node n

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

A* search

(Uniform-cost search algorithm with a new node selection strategy)

Strategy: select node that minimizes *f(n)=g(n)+h(n)*

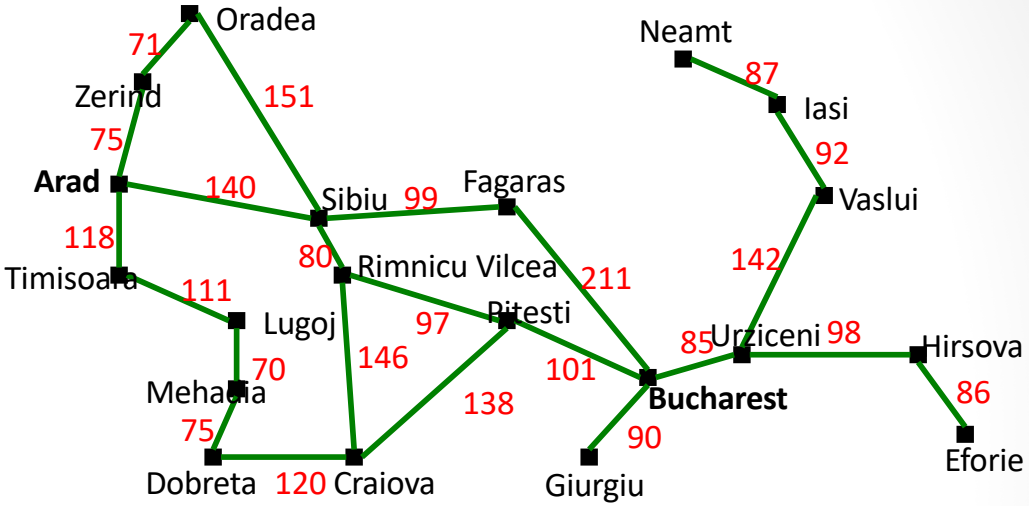
(implemented as a priority queue on f-values)

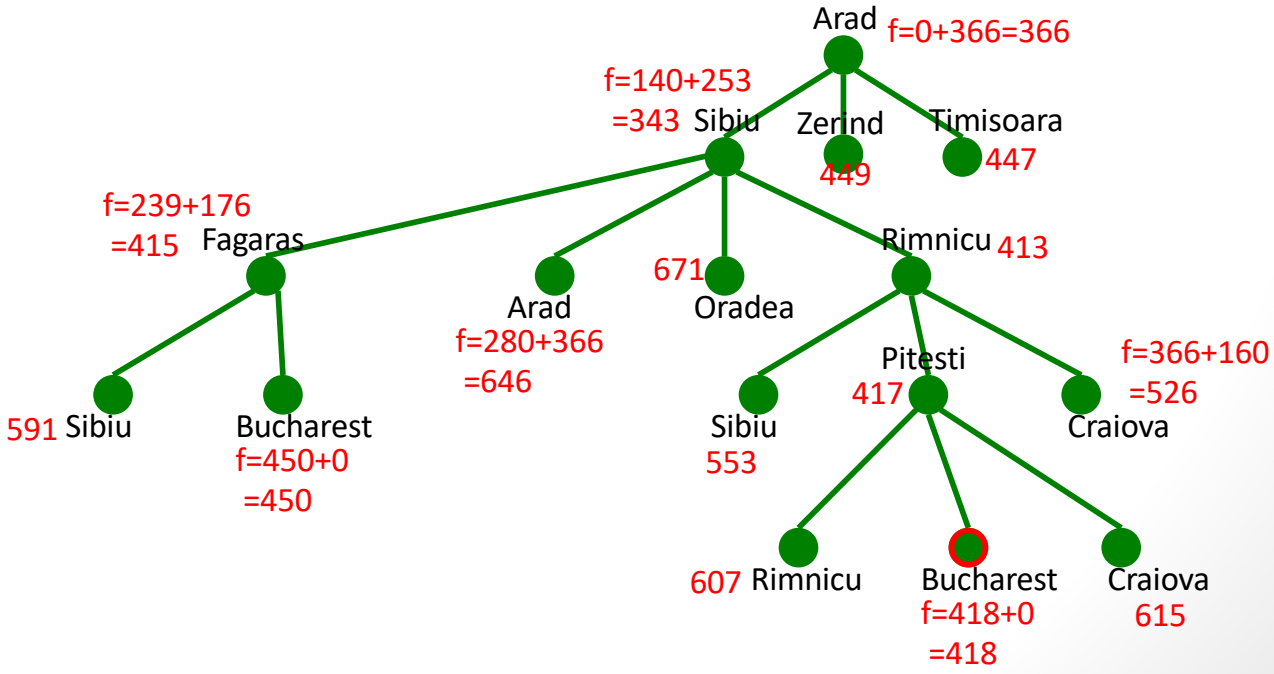
Heuristic (informed) search strategies

A* search - select node that minimizes *f(n)=g(n)+h(n)*

Example: route between Arad and Bucharest

Straight-line distances to Bucharest h(n)	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





Heuristic (informed) search strategies

A* search - select node that minimizes $f(n)=g(n)+h(n)$

admissibility: if heuristic $h(n)$ never overestimates true cost to goal

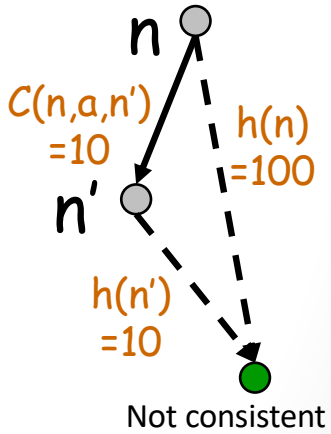
is the heuristic of the Arad-Bucharest problem admissible?

consistency: if heuristic $h(n)$ satisfies this triangular inequality:

for any node n' successor of n using action a,

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

where $c(n,a,n')$ is the step cost of action a applied to node n



is the heuristic of the Arad-Bucharest problem consistent?

Consistency implies admissibility (try to prove it!)

Properties:

- for tree-search, if h is admissible, then A* is optimal and complete
- for graph-search, if h is consistent, then A* is optimal and complete

Heuristic (informed) search strategies

A* search - select node that minimizes *f(n)=g(n)+h(n)*

Proposition:

if h(n) is admissible, the solution returned by A* in a tree-search is,

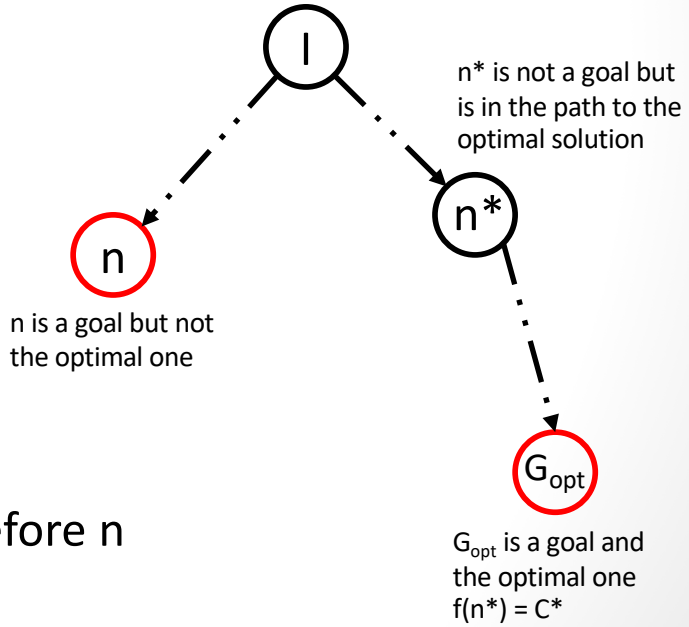
- complete, as long as step costs are strictly positive
- optimal,

$f(n) = g(n)+0$

$> C^*$ (because n is not the optimal solution)

$\geq g(n^*)+h(n^*)$ (because h(.) is an admissible heuristic)

$= f(n^*)$



So $f(n) > f(n^*)$

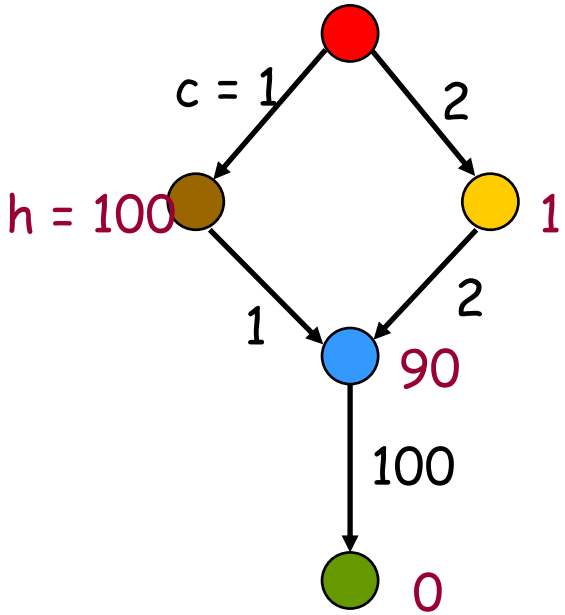
and therefore n* will always be expanded before n

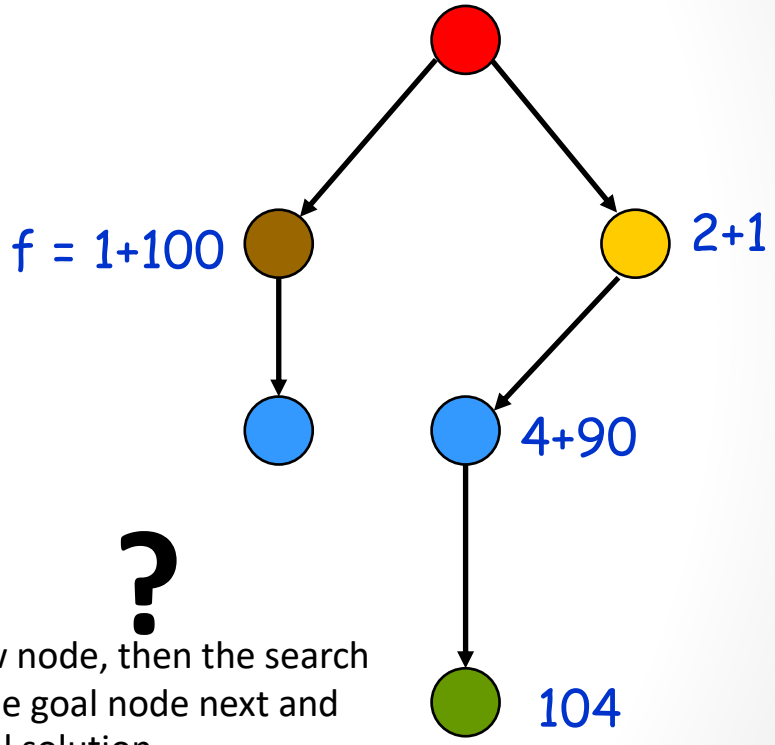
However, for graph-search optimality is not guaranteed if repeated nodes are simply discarded

Heuristic (informed) search strategies

A* search - select node that minimizes *f(n)=g(n)+h(n)*

In case of a non consistent heuristic, optimality is not guaranteed if repeated nodes are discarded





?

If we discard this new node, then the search algorithm expands the goal node next and returns a non-optimal solution

Whenever a node is generated by A*, it is first matched against OPEN and CLOSED and if a duplicate is found, **the copy with the larger *g*-value is ignored**

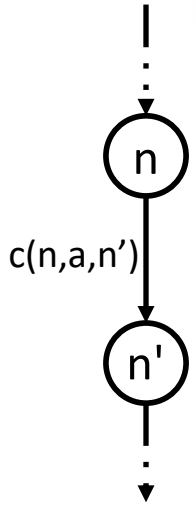
Heuristic (informed) search strategies

A* search - select node that minimizes *f(n)=g(n)+h(n)*

Proposition:

if *h(n)* is consistent (i.e. *h(n) ≤ c(n,a,n') + h(n')*),

- the values of *f(n)=g(n)+h(n)* along any path are non-decreasing,



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

So f(n') ≥ f(n)

- node expansion order in increasing values of *f(n)*
- whenever A* selects a node n for expansion, the optimal path to that node has been found
- the first solution found is optimal
- all nodes with *f(n) < C** are expanded

Heuristic (informed) search strategies

A* search - select node that minimizes *f(n)=g(n)+h(n)*

Complexity:

In general, time and space complexity of A* is exponential with the solution depth: $O(b^d)$

since it expands all nodes with f(n) < C*, and some (or all) nodes with f(n) = C*, and the number of nodes with f=C* is exponential with the solution depth (d)

except if $$\left|h(n)-h^*(n)\right| \leq O\left(\log h^*(n)\right)$$

Usually, A* runs out of memory before it runs out of time

Heuristic (informed) search strategies

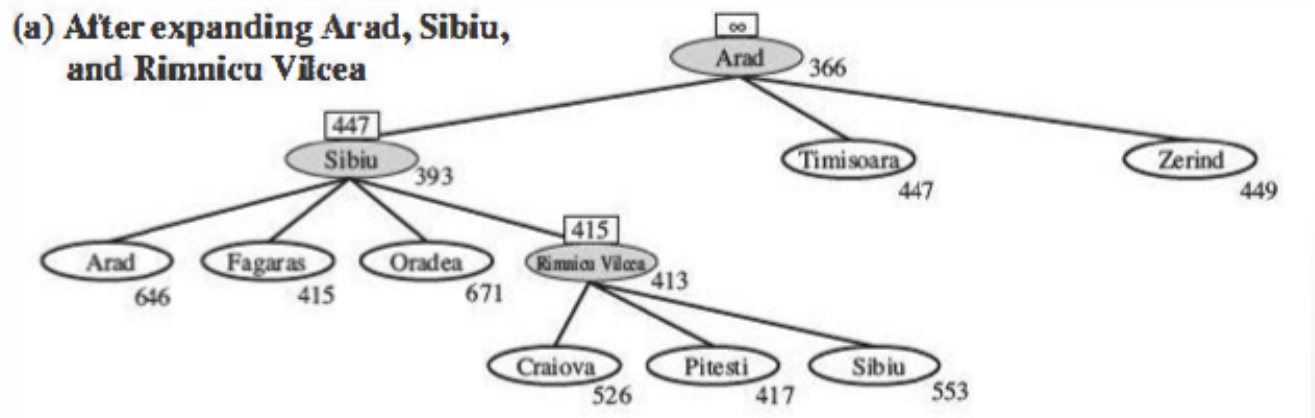
Memory-bounded A* search

- Iterative-Deepening A* (IDA*):

  Iterative-deepening search using cutoff value for f, where the cutoff is set to the smallest f that exceeded the cutoff in the previous iteration

- Recursive best-first search (RBFS):

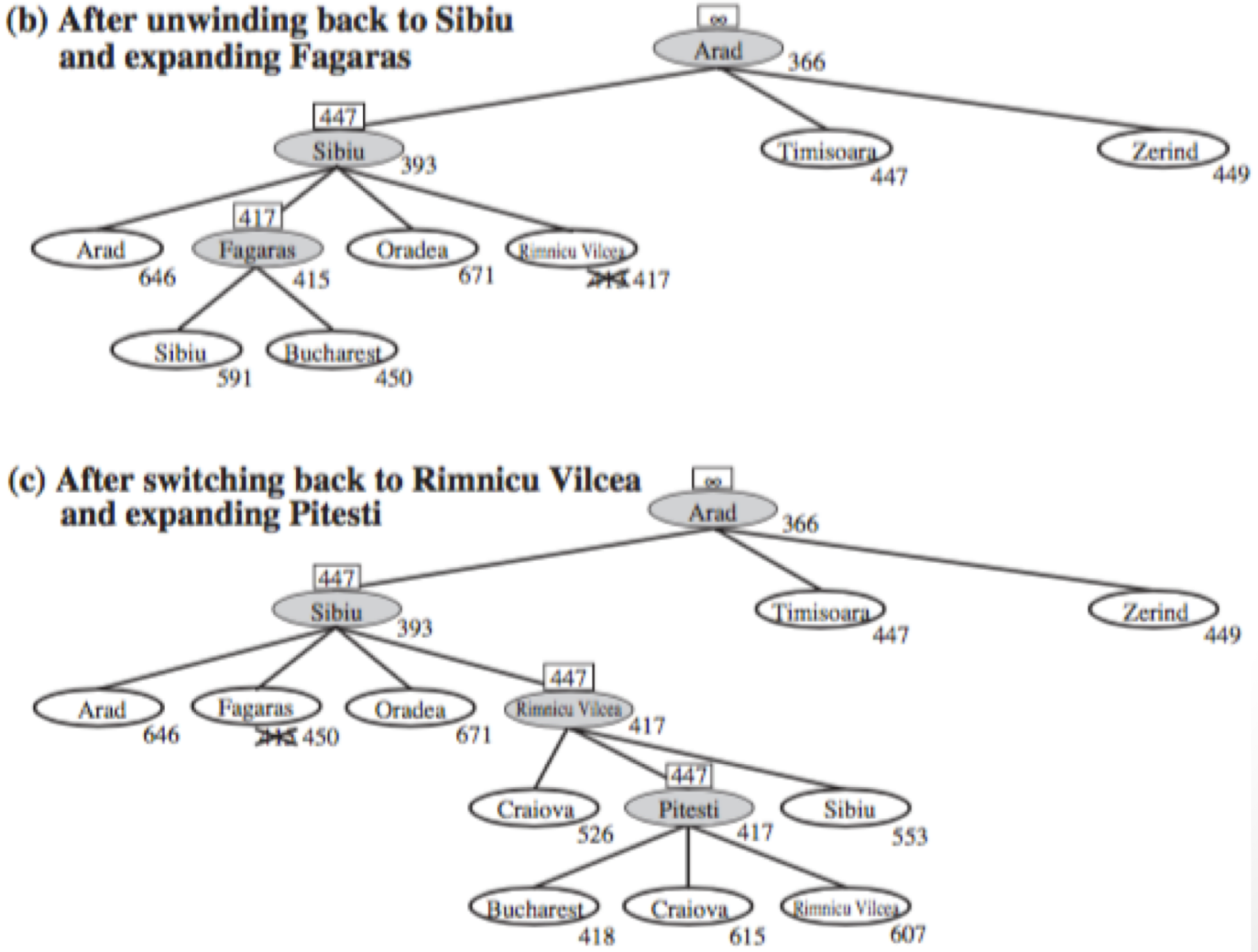
  similar to that of a recursive depth-first (backtrack) search, but rather than continuing indefinitely down the current path, it uses a *f-limit* variable to keep track of the f-value of the best alternative path available from any ancestor of the current node.



Heuristic (informed) search strategies

Memory-bounded A* search

- Recursive best-first search (RBFS)



Heuristic (informed) search strategies

Memory-bounded A* search

- Simplified memory-bounded A* (SMA*):

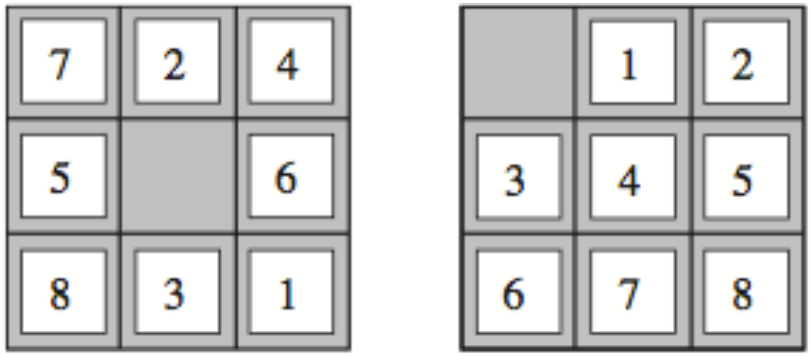
  performs A* until memory full;

  then, discard oldest worst node - the one with the highest f-value;

  and, backs up the value of the forgotten node to its parent.

Heuristic functions

Example: 8-puzzle problem



Possible heuristic functions

- *h1 = number of misplaced tiles*
- *h2 = sum of distances between current and goal position of each tile*

  (using city block distance or Manhattan distance)

Heuristic functions

effective branching factor: the value of b* that solves the equation

$N = b^* + (b^*)^2 + \ldots + (b^*)^d$

where N is the total number of generated nodes

$b^* \geq 1$ *the closer* b^* is to 1 the better

h_2 dominates h_1 iff $h_2(n) \geq h_1(n)$ for all n

h_2 never expands more nodes than h_1; so use h_2

Example: for 8-puzzle, h_2 dominates h_1

Heuristic functions

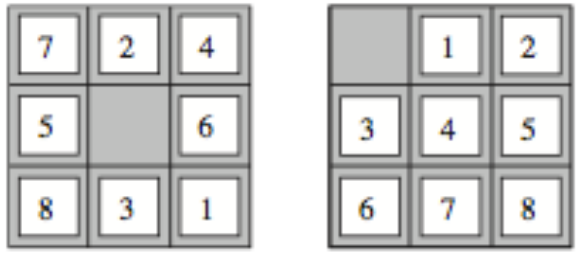
Comparing Iterative-deepening and A* for 1200 random 8-puzzle problems

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Heuristic functions

A good approach to find consistent heuristics is to consider a **relaxed problem**. i.e. set the heuristic as the true cost to goal for a relaxed version of the original problem

Example: 8-puzzle



Original problem: a tile can move from A to B if A and B are adjacent and B is blank

Candidate relaxations:

a. A tile can move from A to B ➔ $h_1(n)$
b. A tile can move from A to B if A and B are adjacent ➔ $h_2(n)$
c. A tile can move from A to B if B is blank ➔ Gaschnig's heuristic

If a collection of admissible heuristics $h_1 \ldots h_m$ is available for a problem and none of them dominates any of the others, which should we choose?

$$h(n) = \max\{h_1(n), h_2(n), \cdots, h_m(n)\}$$

Adversarial search

Competitive environments in which the agents' goals are in conflict (aka games)

Assumptions:

- turn-taking
- two-player
- zero-sum (either a draw or, if one player wins, the other looses)
- perfect information (i.e., full observability)
- deterministic

What is the difference between a search problem like 8-puzzle and a chess game?

An opponent introduces uncertainty $\Longrightarrow$ Contingency problem

What is the difference between the uncertainty in a game of chess and the uncertainty in a game of dice?

opponent will do everything to win

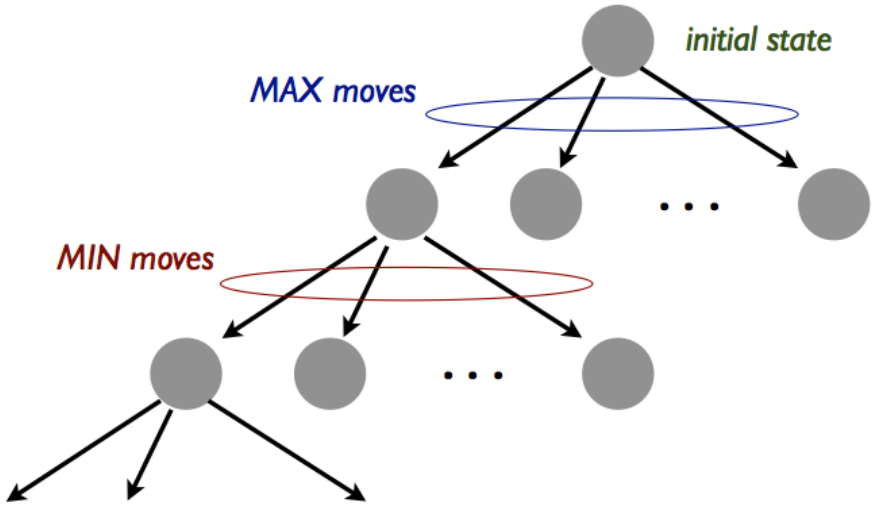
Solution: find the sequence of actions that allows the agent to obtain the maximum benefit "regardless" of the actions of the adversary

Adversarial search

Elements of a game search problem:

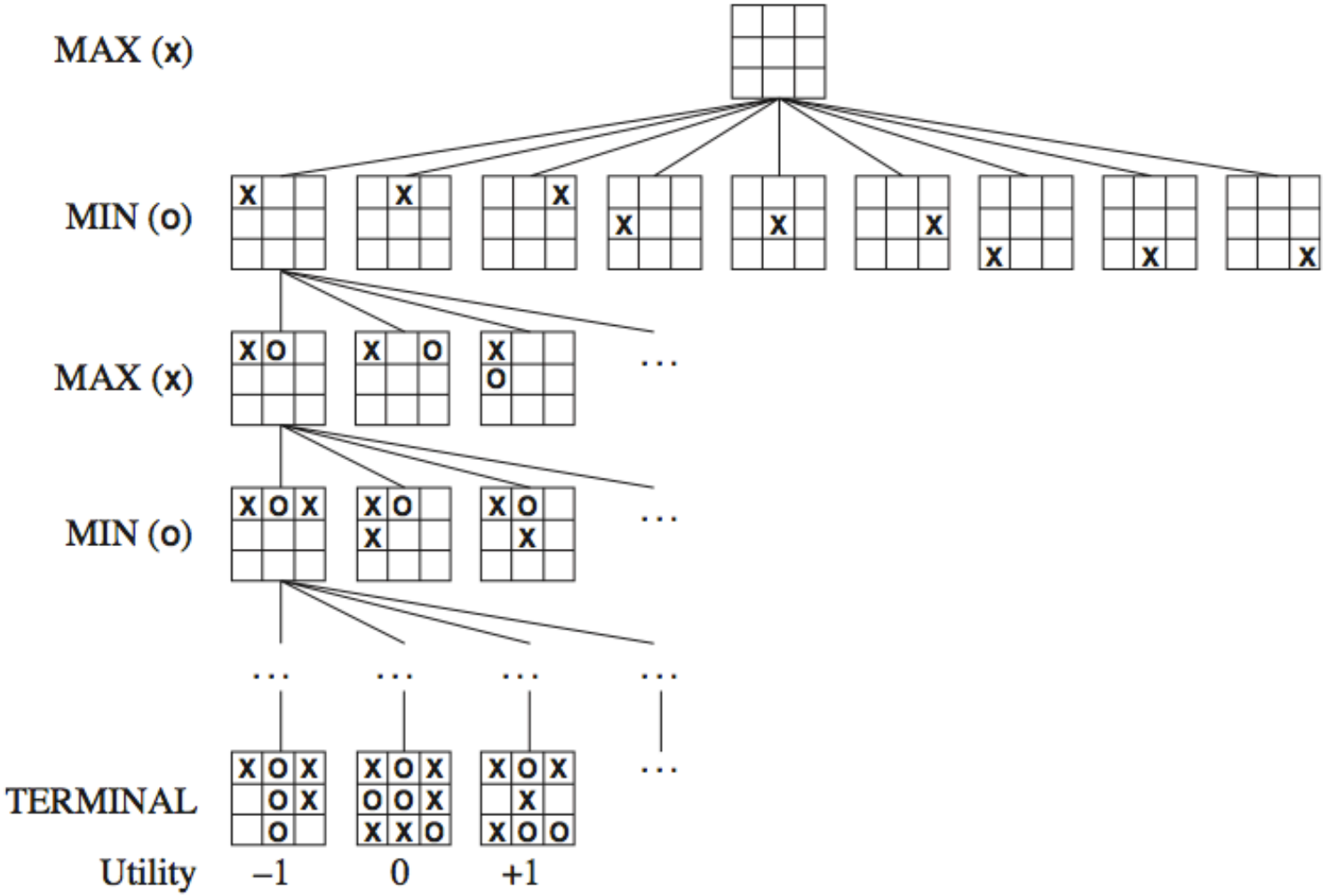
- 2 players: MAX (agent) and MIN (opponent)
- An initial state
- A set of operators (actions)
- A goal ending test function
- An evaluation function (or utility function or payoff function) - e.g., -1 if MAX looses, +1 if MAX wins, 0 in case of a draw

Game tree is formed by expanding all possible moves, for each player in turn, starting from the initial state of the game, until the end of the game

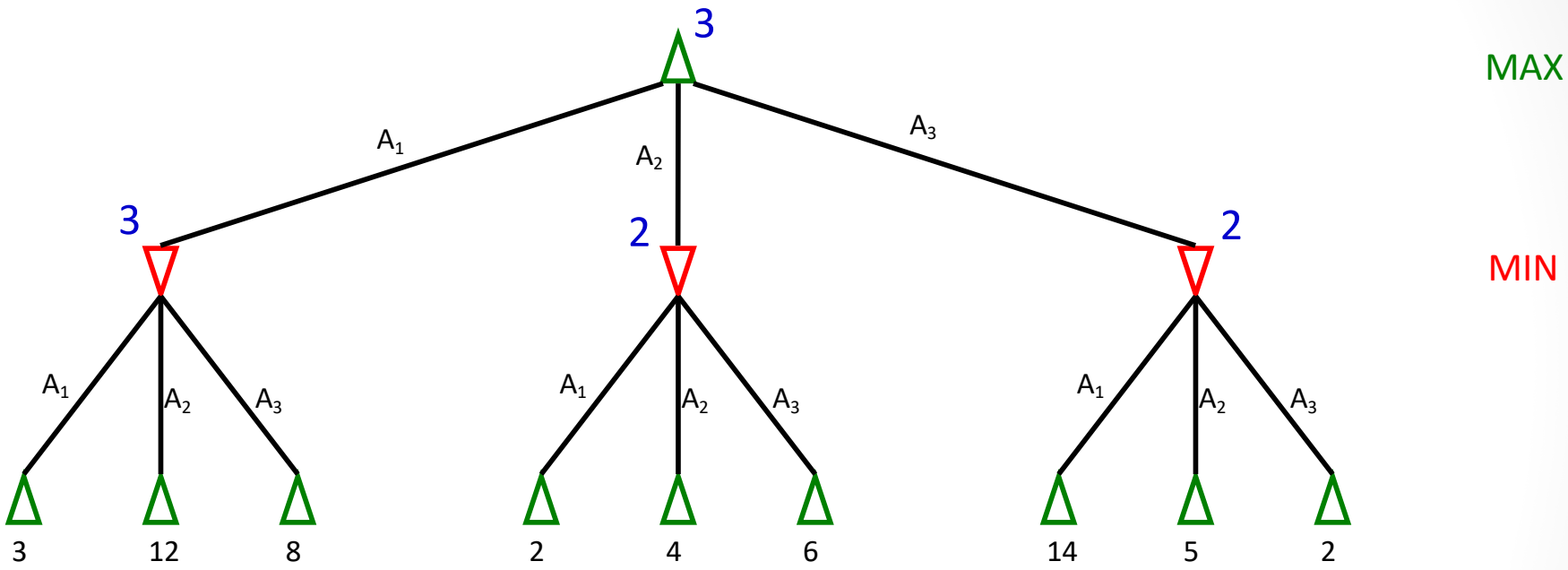


Adversarial search

Example: game tree for tic-tac-toe (less than 9! = 362 880 nodes)



MiniMax Algorithm



This procedure is the **Minimax algorithm**

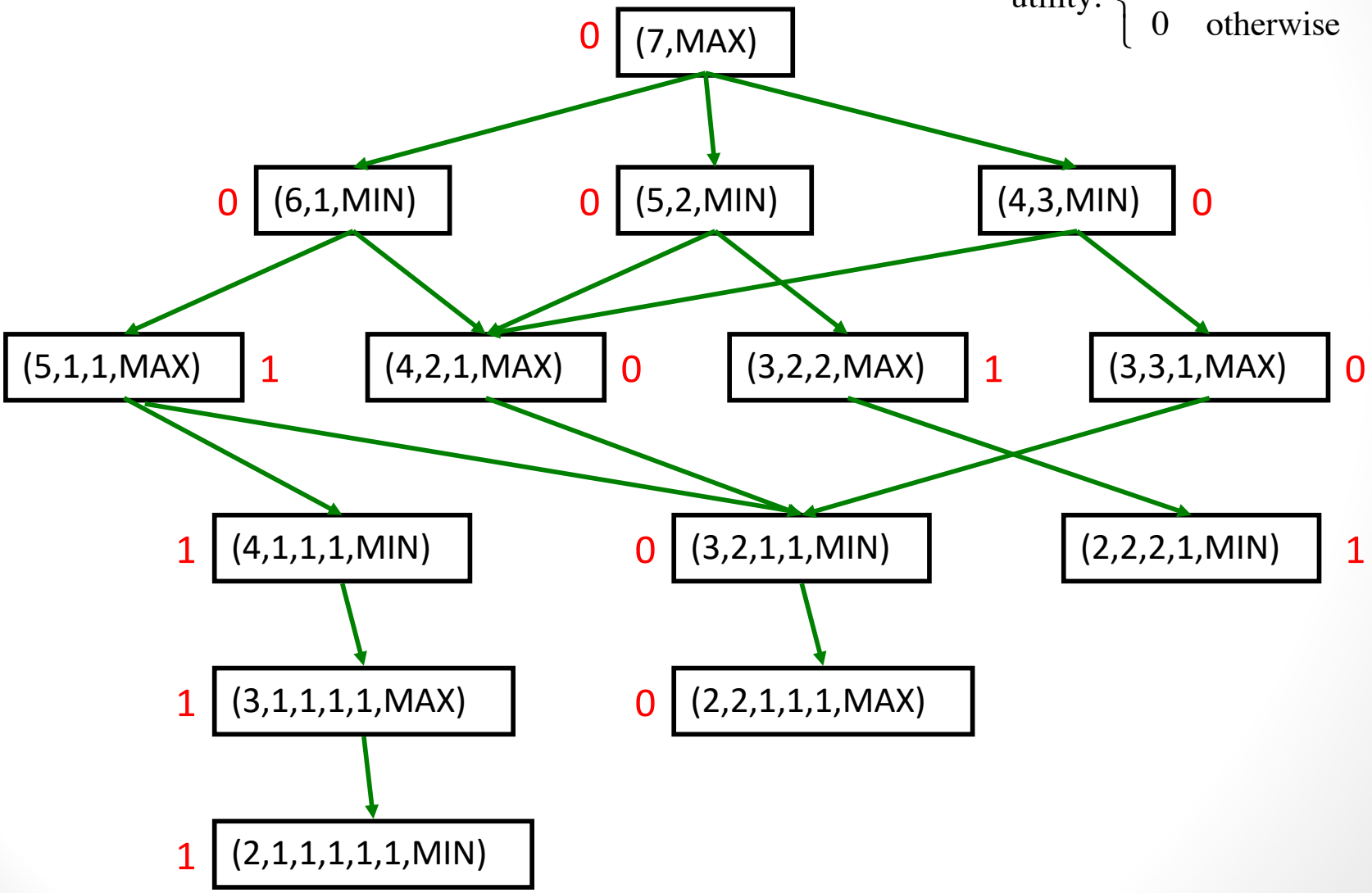
$$\text{minimax}(s) = \begin{cases} \text{utility}(s) & \text{if Terminal-test}(s) \\ \max_{a \in Actions(s)} \text{minimax}(result(s,a)) & \text{if Player(s)=MAX} \\ \min_{a \in Actions(s)} \text{minimax}(result(s,a)) & \text{if Player(s)=MIN} \end{cases}$$

The procedure can be implemented by traversing the tree in depth-first order

MiniMax Algorithm

Example: a stack with 7 coins; each player divides a stack into two unequal stacks; loses the player that can not split any stack

utility: $\begin{cases} 1 & \text{if MAX wins} \\ 0 & \text{otherwise} \end{cases}$



MiniMax Algorithm

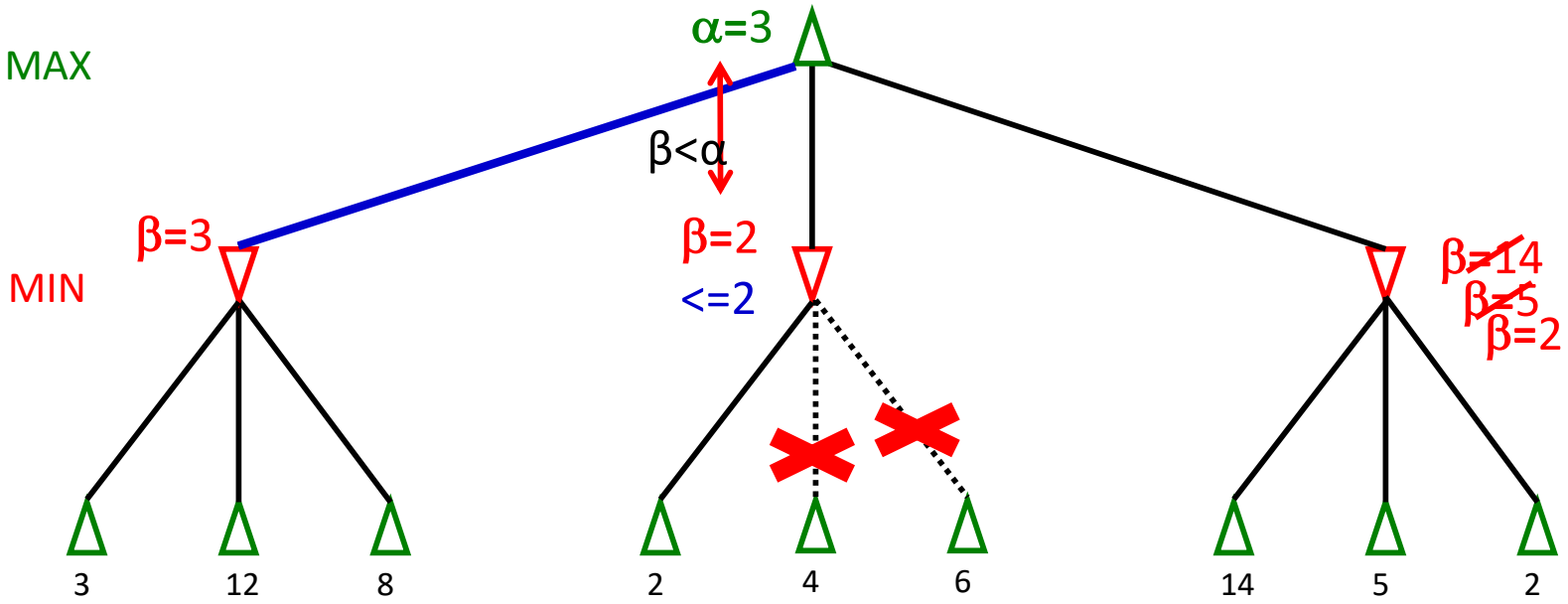
What are the weakest aspects of MiniMax?

- need to go to terminal nodes
- need to evaluate all nodes

Fortunately, the structure of the minimax tree allows to prune unnecessary parts of the tree – called **Alpha-beta pruning**

Alpha-Beta Algorithm

α is the best value found along the path **for MAX**
β is the best value found along the path **for MIN**

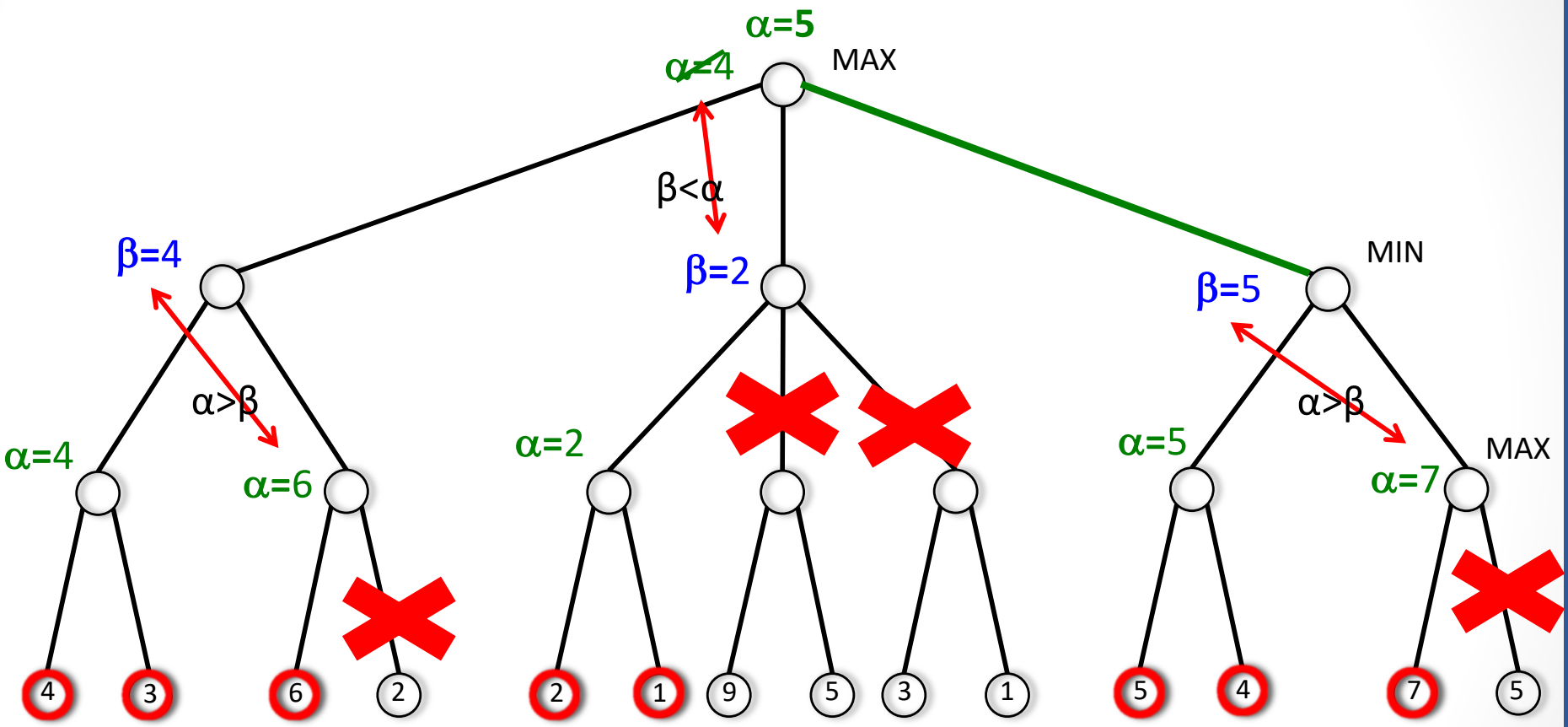


The search is stopped in the following cases:

- in a MIN node which β is smaller or equal to any α of its MAX predecessors. The value of this node is fixed to β
- in a MAX node which α greater or equal to any β of its MIN predecessors. The value of this node is fixed to α

Alpha-Beta Algorithm

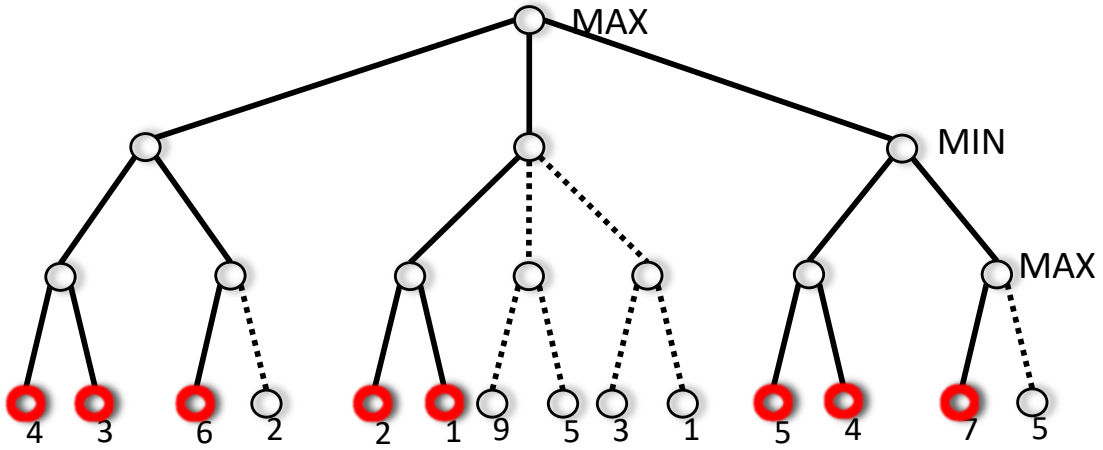
α is the best value found for MAX along the path
β is the best value found for MIN along the path



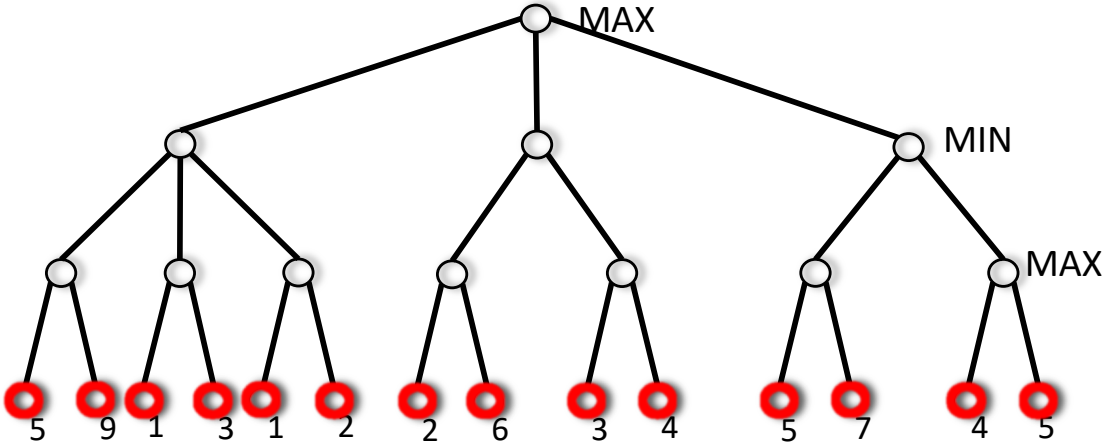
The search is stopped in the following cases:

- in a MIN node which β is smaller or equal to any α of its MAX predecessors. The value of this node is fixed to β
- in a MAX node which α greater or equal to any β of its MIN predecessors. The value of this node is fixed to α

Alpha-Beta Algorithm – pruning order



6 nodes pruned



no nodes pruned

Pruning effectiveness depends significantly on the expansion order

Alpha-Beta Algorithm

Pruning effectiveness depends significantly on the expansion order

Best case scenario: examine first the successors that are likely to be best

In this case, $O(b^{m/2})$ instead of $O(b^m)$, and *the effective branching factor is* $\sqrt{b}$,

Example: in chess, b drops from 35 to 6

If successors are examined in random order, $O(b^{3m/4})$ for moderate b

So, simple ordering functions often lead to improvements within the random and the best-case ordering

In non-trivial games it is intractable to expand the full game tree (except when close to the end of the game)

So, the search has to be cut off at some point

The Utility() function has to be replaced by an heuristic evaluation

TÉCNICO LISBOA

Cutting off the game tree

The simplest approach is based on depth (i.e., depth-limited search): blind cutoff

Quiescent state: unlikely to have wild swings in value in near future (peaceful states)

- e.g., states where advantageous moves can be made are not quiescent
- Quiescence search: do not cut off on non quiescent states

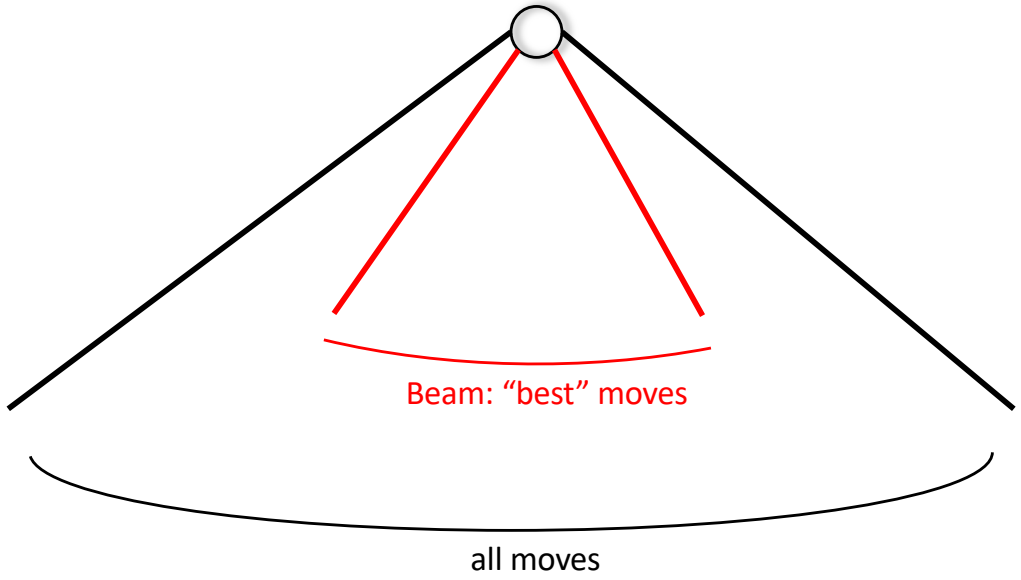
Horizon effect: arises when the agent faces an opponent move that causes serious damage and is unavoidable, but it was pushed over the maximum depth (just delaying the damage)

- Can be mitigated by singular extensions: a “clearly good” move; once discovered it is remembered
- when search reaches cutoff limit, algorithm checks if the singular extension is legal
- if it is the move is considered and the tree is expanded beyond the depth limit

Cutting off the game tree

Forward pruning: do not expand bad moves

- e.g., beam search, limit the number of expansions to a “beam” of the n best moves, instead of all possible moves



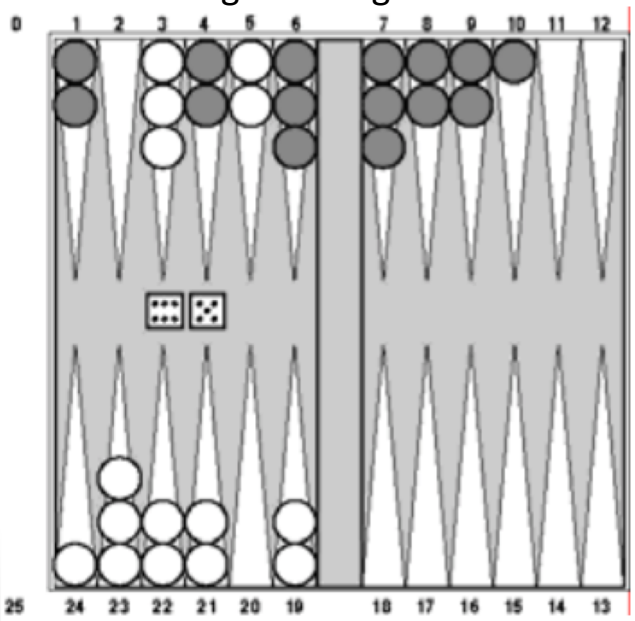
Stochastic games

Minimax (and alpha-beta) can be extended to non-deterministic games

Chance nodes:

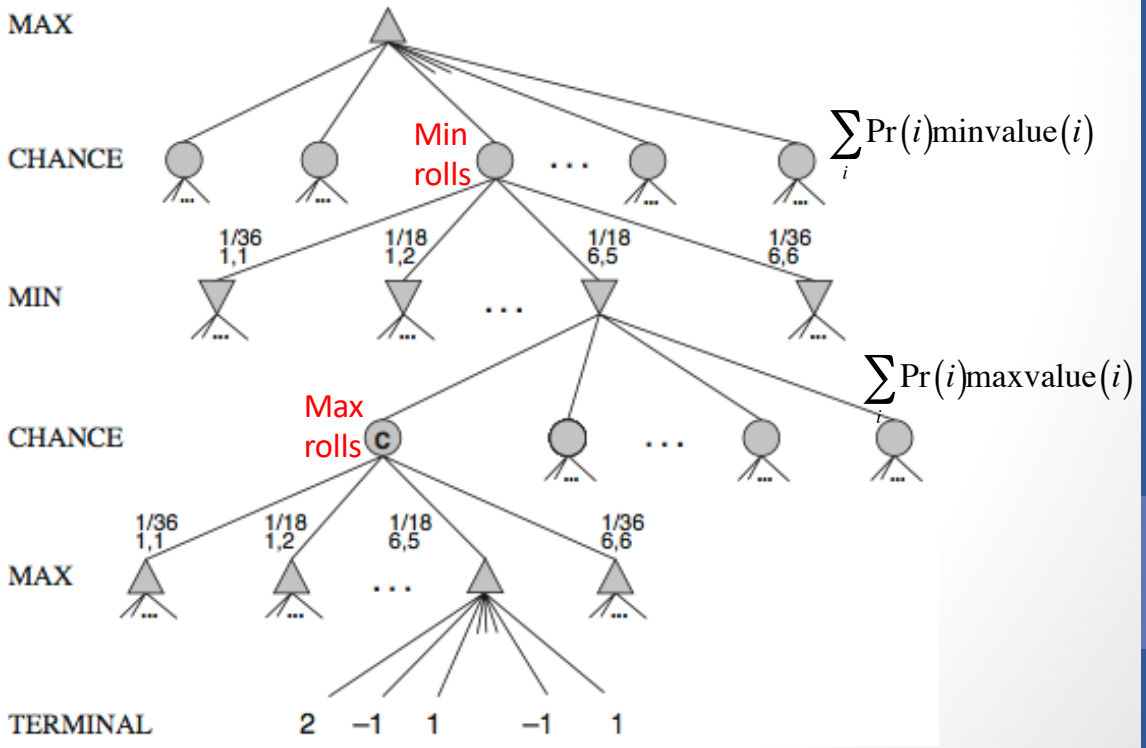
- Represent probabilistic outcomes
- Backup of values by computing the expected value over the child nodes

Backgammon game



White possible moves

- 5-10, 5-11
- 5-11, 19-24
- 5-10, 10-16
- 5-11, 11-16



Deep Blue (IBM): chess player

Defeated world champion Gary Kasparov in 1997

Hardware:

- 30 IBM RS/6000 processors
- 480 custom VLSI dedicated chess processors
- Examined up to 30 billion positions per move (about depth 14; up to depth 40 with singular extensions)

Software:

- Evaluation function with 8000 features
- Opening book with 4000 positions
- Database with 700 000 grandmaster games
- Solution for all possible 5-piece endgames (and many 6-piece ones)