Foundations of Information Processing

Search problems and playing games

Considerations about the nature of problems

Consideration 1:

Are all tasks/problems deterministic?

Consideration 2:

How can search techniques be used for solving deterministic and nondeterministic problems?

Consideration 3:

How can be played algorithmically?

Consideration 4:

Is it possible to play optimally?

Nondeterministic algorithms and playing games

An algorithm is defined to be deterministic. Thus, the solution must be unambiguous, and in each step of an algorithm it must be known exactly what to do and what is the next step.

However, all problems are not deterministic by their nature.

Problems can be solved as **search problems**.

Games are good examples of **nondeterministic** problems where available actions are well known and defined, but how to use them is not known at the beginning of a game.

Search problems and playing games

”When solving problems,
dig at the roots instead of just hacking at the leaves.”
(Anthony J. D'Angelo, an American writer)

- Deterministic and nondeterministic search problems.
- Basic methods:
  - Breadth-first search (BFS).
  - Depth-first search (DFS).
- Heuristics: ”sure information” + rules of thumb.
- Game playing strategies: how to win the game?

Source (partly/modified from): J. Boberg, Johdatus tietojenkäsittelytieteeseen, Turun yliopisto, 2010 (in Finnish)

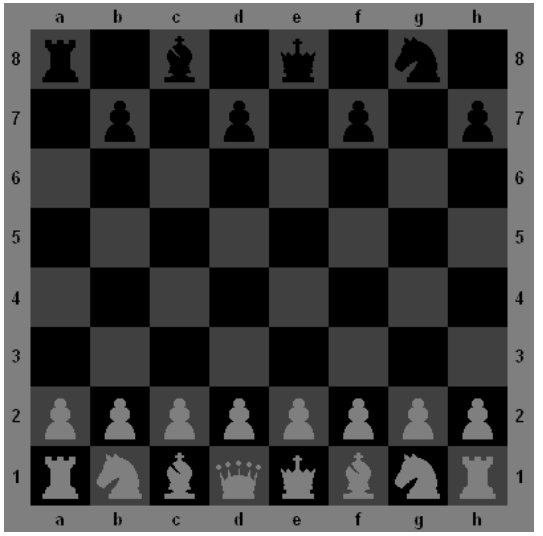
Deterministic problems and nondeterministic problems

- Deterministic problems:
  - During the execution, it is exactly known at each step how to continue to the next step.
  - The same input always causes the same choices in the execution.
- Many problems are nondeterministic due to their nature:
  - In search problems, a set of available actions is known beforehand.
  - It is unclear where to find the solution, i.e., to what direction to move in the search.

Nondeterministic problems

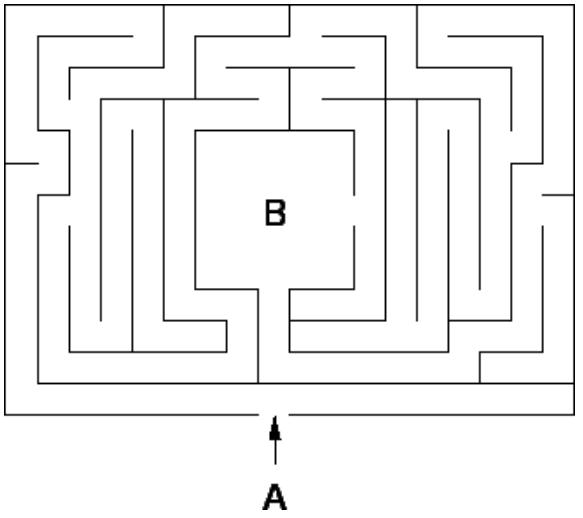
Example: chess

- Allowed moves are well known due to the exact rules of the game.
- The order of experimenting with these moves to achieve the goal (= to win the game) is unclear.



Example: labyrinth

- Available directions of moving: south, west, north, or east.
- It is not known in what order to try the directions in the maze in order to find the most optimal path.

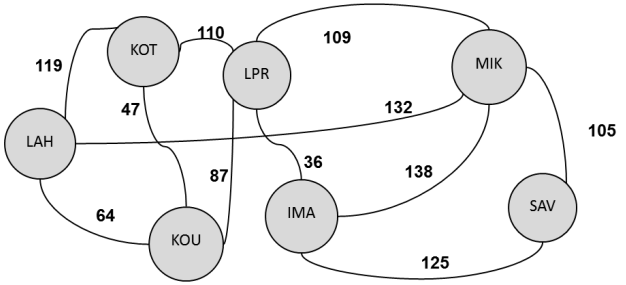


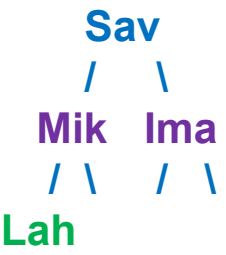
Search problems: states

- Nondeterministic problems can be described as search problems where the solution can be found in a set of different options (states).
- A given search problem can be represented as states as follows:
  - A set of the states: S
  - The start state: $s_0 \in S$
  - The set of the goal states: $F \subseteq S$.
  - Moves: t: $S \rightarrow S$, $s_i \rightarrow s_j$
- Solving the problem begins from the start state s_0. At each state there is a set of allowed moves which direct to a new state $s_{k-1} \rightarrow s_k$.

Search problems: search trees and solution paths

- **State space** = a graph which contains all possible states and moves connecting them.
  - **A graph** consists of a set of *nodes* (vertices) and a set of *edges* (links) connecting them.
  - There is an edge between two nodes, if there is a relation between them (binary relation).





- **Search tree** = a tree to store those moves which are investigated while trying to find the goal state.
  - Example: a route from **Sav**onlinna (start state) to **Lah**ti (goal state).
- **Solution paths** = a path in the search tree from the root (start state) to that leaf node where there is the goal state:
  - Best solution path = shortest solution path.
    - Sav-Mik-Lah, is this the shortest one?
  - Best goal state = the state which leads to the best possible situation in terms of who made the search. For example, to find the shortest route or to win the game.

Finite search problems

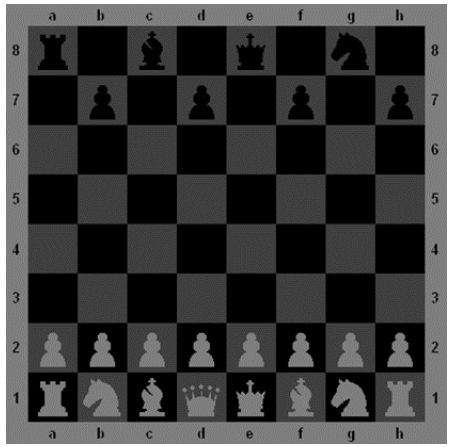
- The search space is finite.
- The full graph can be constructed during the search.
- Finding the solution, sooner or later is sure.
- The problem to be solved is to find the fastest or best solution.
- Example: tic-tac-toe (noughts and crosses, Xs and Os)
  - A game board of a 3x3 grid.
  - Two players:
    - The one marks X and the other O in turns.
  - A finite number of moves.
  - The length of playing the game can be defined.
  - The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.

1	2	3
4	5	6
7	8	9

O	O	X
	X	
X		

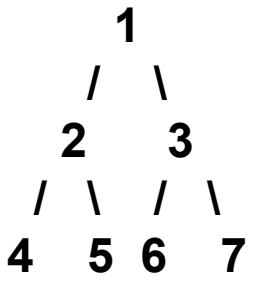
Infinite search problems

- The search space is infinite.
- The full graph cannot be constructed.
- The solution cannot be necessarily found ever, if a “wrong” search strategy has been selected.
- Alternatively, the solution is found, but its computation is typically not feasible.
- Example: chess
  - The size of a game board is limited.
  - The number of moves has not been limited.
  - Many different types of moves available.
  - The length of playing the game cannot be defined.



Breadth-first search (BFS)

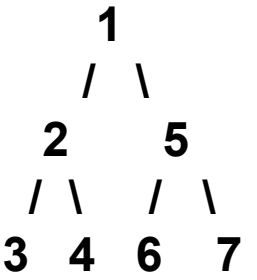
- Starting at the tree root, explore all nodes at the present depth (breadth) prior to moving on to the nodes at the next depth level.
  - Example: the order of progress has been numbered, starting from the root node (start node) marked as “1”.
- The states are stored as a queue (first in, first out, FIFO).
  - Data structures are considered later in the course.



- Properties:
  - Complete: all options are explored (maybe slow).
  - Sure: the solution is found due to the completeness.
  - Optimal: the best solution is found in the finite search.
    - The full graphs is explored if needed.
    - Must be able to recognize the *goodness* of each solution as compared to other solutions.
  - For example, when *the goal state* has been found at a certain depth it must be checked whether other nodes at the same level would not produce *a better solution*, e.g., no shorter route to Lahti via some other towns (the nodes at the same level) can be found.

Depth-first search (DFS)

- Starting at the root node, explore explores as far as possible along each branch (depth) before backtracking.
  - Example: it is decided to explore the left branch first as deep as possible.
  - *Backtracking*, if no solution in the last node (leaf node), for example, if no “Lahti” in Node 3 then explore Node 4.



- The states are stored as a stack (last in, first out, LIFO).
- Properties:
  - Incomplete: not all options explored necessarily (could be fast).
  - Unsure: an infinite search possible.
    - For example, the left branch continues “forever”.
  - Suboptimal = the optimal solution is not guaranteed by the search.
    - Even a finite search may not find the best solution, because typically the search is stopped as soon as the first acceptable solution has been found.

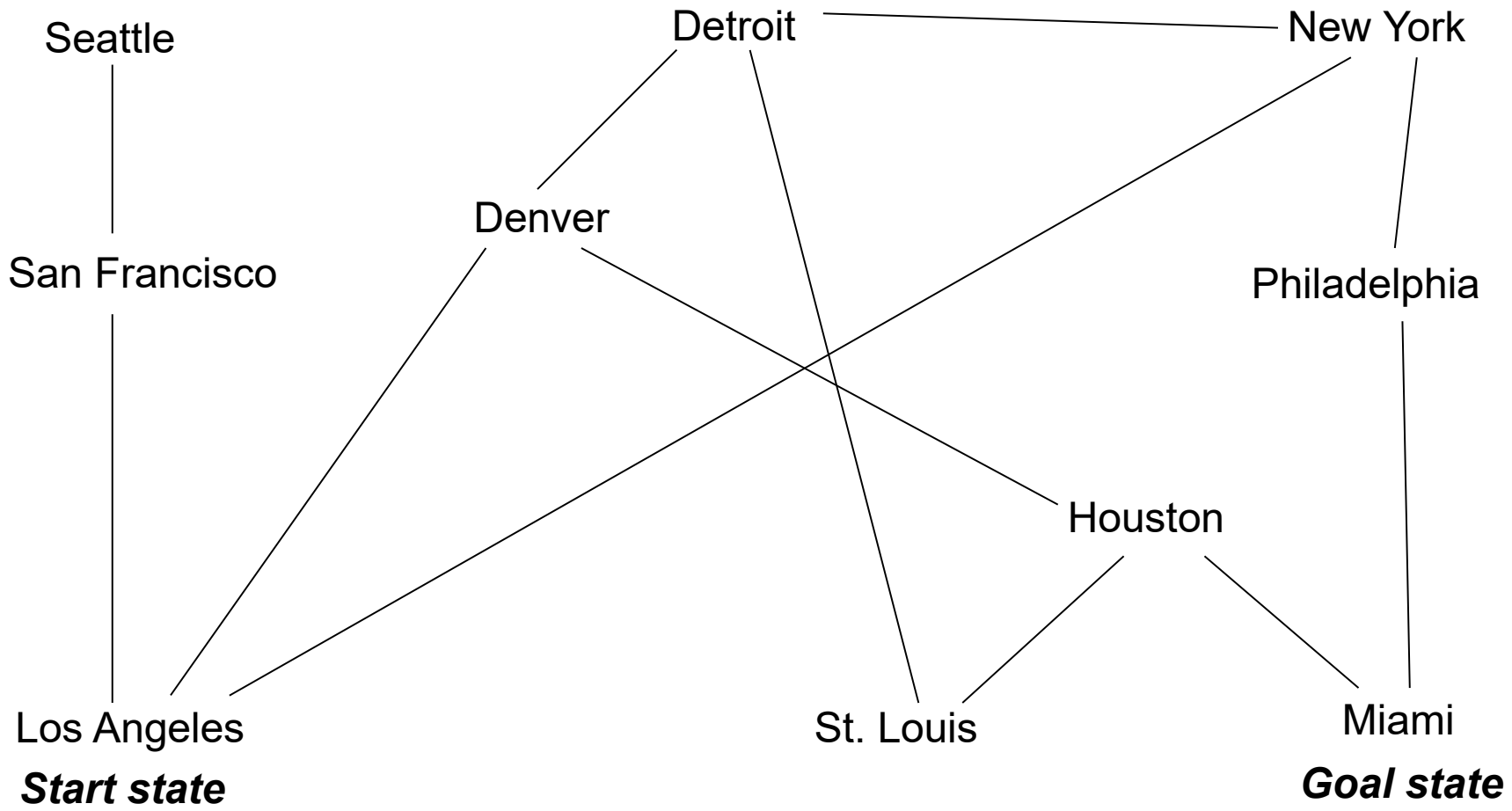
Heuristics.

- A set of rules in which the unexplored states $s_1 ... s_k$ are stored in memory/explored in the search:
  - The order in which the different moves are examined.
  - In practice, the goodness value needed for each node.
- Meaning of heuristics:
  - Insignificant in the breadth-first search:
    - At each stage of the search, all states that can be accessed from this state are examined.
  - Crucial in the depth-first search:
    - Only the **most important state** that can be reached from this state is examined next as long as it is possible:
      - Only when the main direction is not available anymore are other directions explored (backtracking in algorithm strategies).
- Next, let us consider examples.

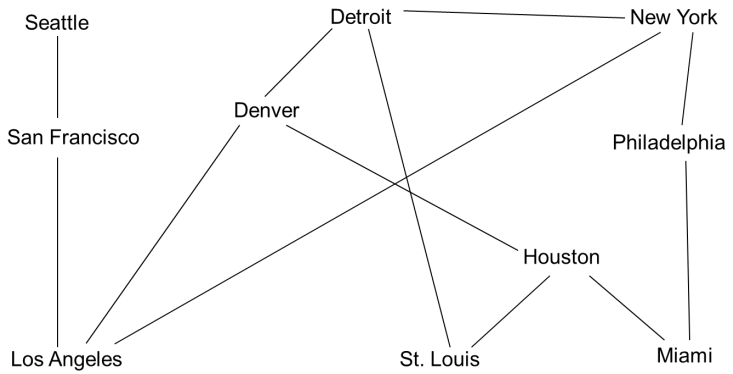
Example of the finite search problem: the minimum number of stopovers from Los Angeles to Miami

What **heuristics** to explore the states? Cardinal directions?:

- Head to **north first,** and then approximately **clockwise** to the next flight.



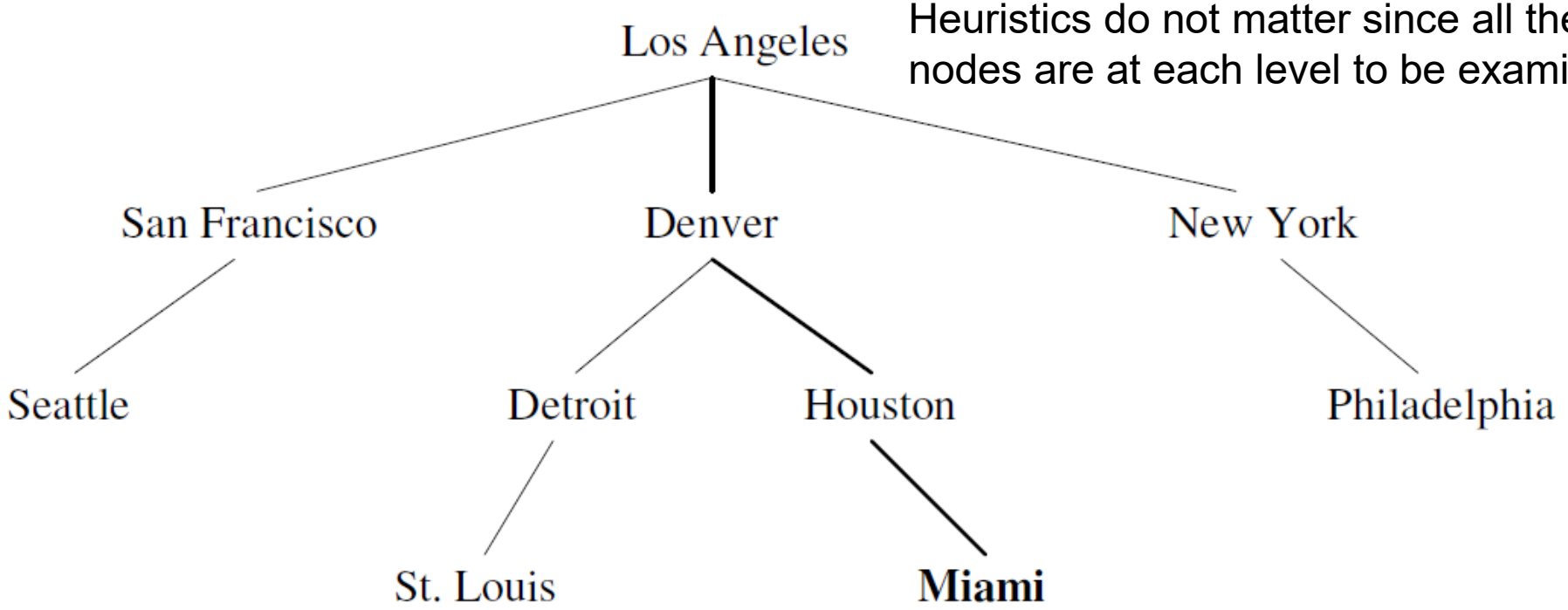
Breadth-first search



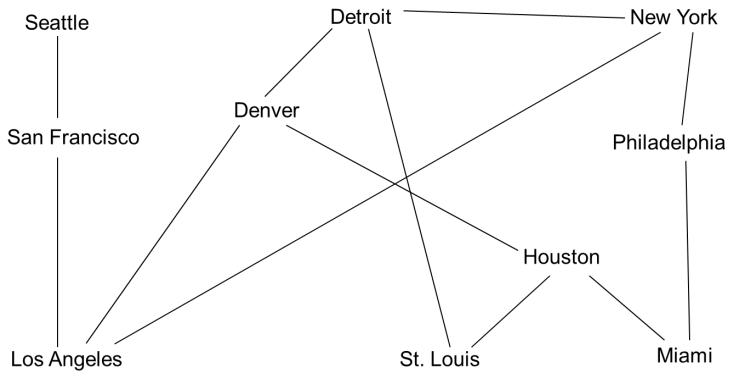
The queue development:
Los Angeles
San Francisco, Denver, New York
Denver, New York, Seattle
New York, Seattle, Detroit, Houston
Seattle, Detroit, Houston, Philadelphia
Detroit, Houston, Philadelphia (empty)
Houston, Philadelphia, St. Louis
Philadelphia, St. Louis, **Miami**

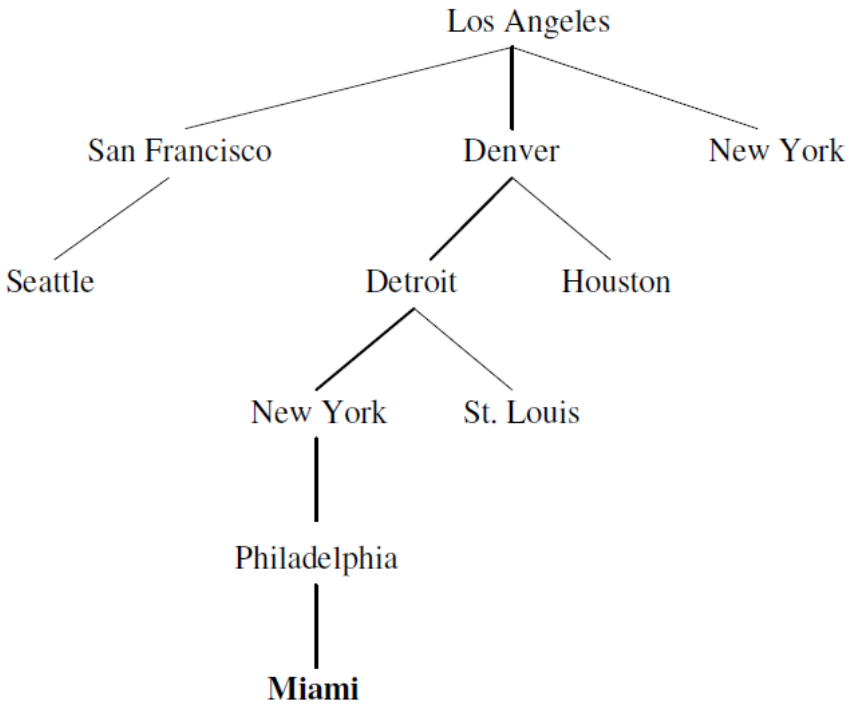
By continuing, the route via Philadelphia would be also found (also two stopovers).

Heuristics do not matter since all the nodes are at each level to be examined.



Depth-first search





The stack development:
Los Angeles
San Francisco, Denver, New York
Seattle, Denver, New York
Denver, New York
Detroit, Houston, New York
New York, St. Louis, Houston, New York
Philadelphia, St. Louis, Houston, New York
Miami, St. Louis, Houston, New York

Backtracking is done only if there is a dead end, as in the case of Seattle.

Heuristics matter: by selecting New York in the beginning the more optimal route would have been found, but there is no backtracking from Denver since there is no dead end. Later the same happens by traveling via Houston, instead of Detroit.

⇒ **Combination of the breadth and depth-first search algorithms (A* search)**:
All nodes at the same breadth level are examined, and using *a goodness estimate the best node* is selected where to continue further (to the next depth level).

Playing games

- Nondetermitistic search problem:
  - Available moves are known, but the order of moves is unclear, and depends on the moves of the opponent.
  - The search space is finite in practice, but often large.
  - For example, in chess there are several options to move, also repetitions.
- Two players (parties) zero-sum games:
  - Players move in turn.
  - The one's win is the other one's loss.
  - Own moves are affected by the opponent's moves.
  - The breadth and depth-first search algorithms do not consider the opponent $\Rightarrow$ **heuristics** needed.

Heuristic function

- A heuristic function estimates **how good** is **a move** in a game from the current state to the new state.
- The next move is selected based on this **heuristic function**:
  - Compute the value of the function for all possible next moves from the current state to know the success of each move.
  - Select that move which leads to the best state (best heuristic value).
- The heuristic search tends to find the shortest path in the search space from the current state of the game to the solution.
- The principle of the greedy algorithm is applied.
  - Select the best move at the current moment (local optimum).

Heuristic game algorithm

Let f: S$\rightarrow$R be a heuristic goodness function

MODULE play

Let s be the start state

REPEAT

Let s$\rightarrow s_i$ (i=1...k)
possible moves

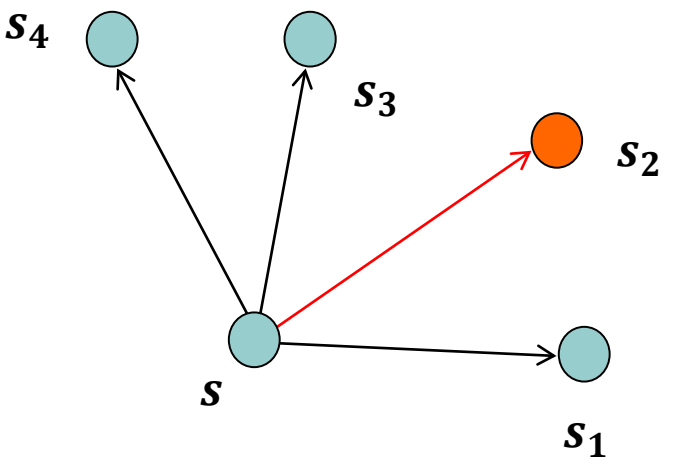
Select the move to the state s_i,
where f(s_i) has the largest value

Wait until the opponent has moved

Denote the new state by s

UNTIL the win OR the loss OR the draw

ENDMODULE



Tic-tac-toe

- Two players write in turn in the 3x3 grid their own mark (X or O).
- The winner is the first one who can get the lines of three own marks horizontally, vertically, or diagonally.
- Let us number the grid as shown in the figure.
- Let us consider the game from a point of view of X.

1	2	3
4	5	6
7	8	9

- Let us define for X a heuristic function $f_x(s)$ =

  The number of lines of three for X

  when in the state *s* the empty spaces are filled with Xs

  –

  The number of lines of three for O

  when in the state *s* the empty spaces are filled with Os.

- Thus, the function is *the difference* of the lines of three of X and the lines of three of O in case of X or O getting all next moves.

Tic-tac-toe: X starts

- X marks a square where the value of the heuristic function is the largest (maximum).
- The new state is named as the number of the square which X selects.
- The goal is to find such a new state where $f_x(s)$ is maximized.
- If X starts, the square in the middle (5) is the best choice
  - since $f_x(5) = 8-4 = 4$ (8 straight lines for X and 4 for O)
- The other moves lead to the worse situation:
  - $f_x(1) = 8-5 = 3$
  - $f_x(2) = 8-6 = 2$
  - No need to compute more since the values for the other squares are same (2 or 3) due to symmetry.
- The new state is as shown in the figure.

1	2	3
4	5	6
7	8	9

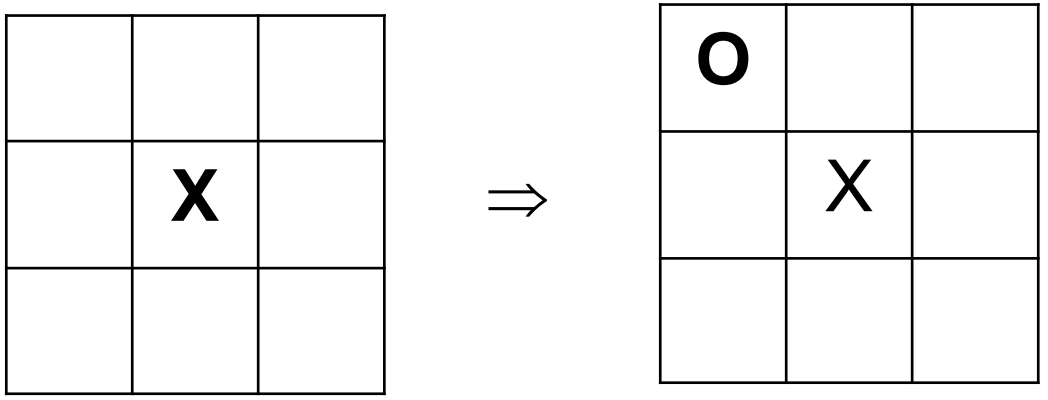
	X	

Tic-tac-toe: O makes the first move

- Let us assume that O uses the same heuristic function.
- Since f_x estimates the goodness from the point of view of X, O tries to minimize its value:
  - $f_x(1) = f_x(3) = f_x(7) = f_x(9) = 5\text{-}4 = 1$
  - $f_x(2) = f_x(4) = f_x(6) = f_x(8) = 6\text{-}4 = 2$

1	2	3
4	5	6
7	8	9

$\Rightarrow$ It wise to O to mark some corner square
let us select the left upper corner as follows:



Tic-tac-toe: X responds to O

- Next, X computes what is the best move:

  $f_x(3) = f_x(7) = f_x(9) = 5\text{-}2 = 3$

  $f_x(2) = f_x(4) = f_x(6) = f_x(8) = 2$

1	2	3
4	5	6
7	8	9

$\Rightarrow$ Let X select the upper right corner (this maximum was computed first):

O		
	X	

$\Rightarrow$

O		**X**
	X	

Tic-tac-toe: the function betrays O

- Now O should absolutely select the lower left corner (state s_7) not to lose the game.
- The used function cannot show this unambiguously due to similar values:
  - 2 for s_2 and s_4, and
  - 1 for s_6, s_7, s_8, ja s_9.
- When selecting the minimum, the states s_6, s_7, s_8, and s_9 are equal choices according to the heuristic function:

$\Rightarrow$ A better game strategy is needed!

1	2	3
4	5	6
7	8	9

x3r 4	o3r 2	f_2 2
O	**O**	X
	X	

x3r 4	o3r 2	f_4 2
O		X
O	X	

x3r 3	o3r 2	f_8 1
O		X
	X	
	O	

x3r 3	o3r 2	f_9 1
O		X
	X	
		O

x3r 3	o3r 2	f_6 1
O		X
	X	**O**

x3r 3	o3r 2	f_7 1
O		X
	X	
O		

Improving the heuristic algorithm for playing

- There are two options to improve the algorithm:
  - Find a better heuristic function.
  - Check more rounds of moves than the next move only:
    - If I (the first player) will move like this

      => what the opponent (the second player) will do

      => what I will do after that move,

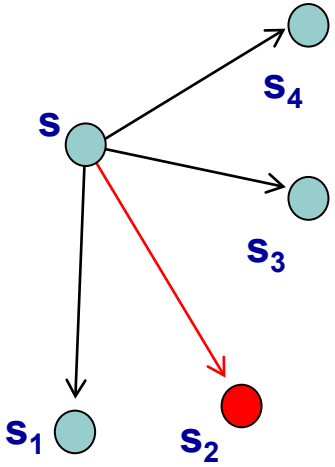
      => etc.
- It can be very challenging to find a better heuristic function.
- Thus, the latter option of checking more moves requires less creativity so let us consider it first.
- This approach is called as the **minmax technique**.

Minmax technique

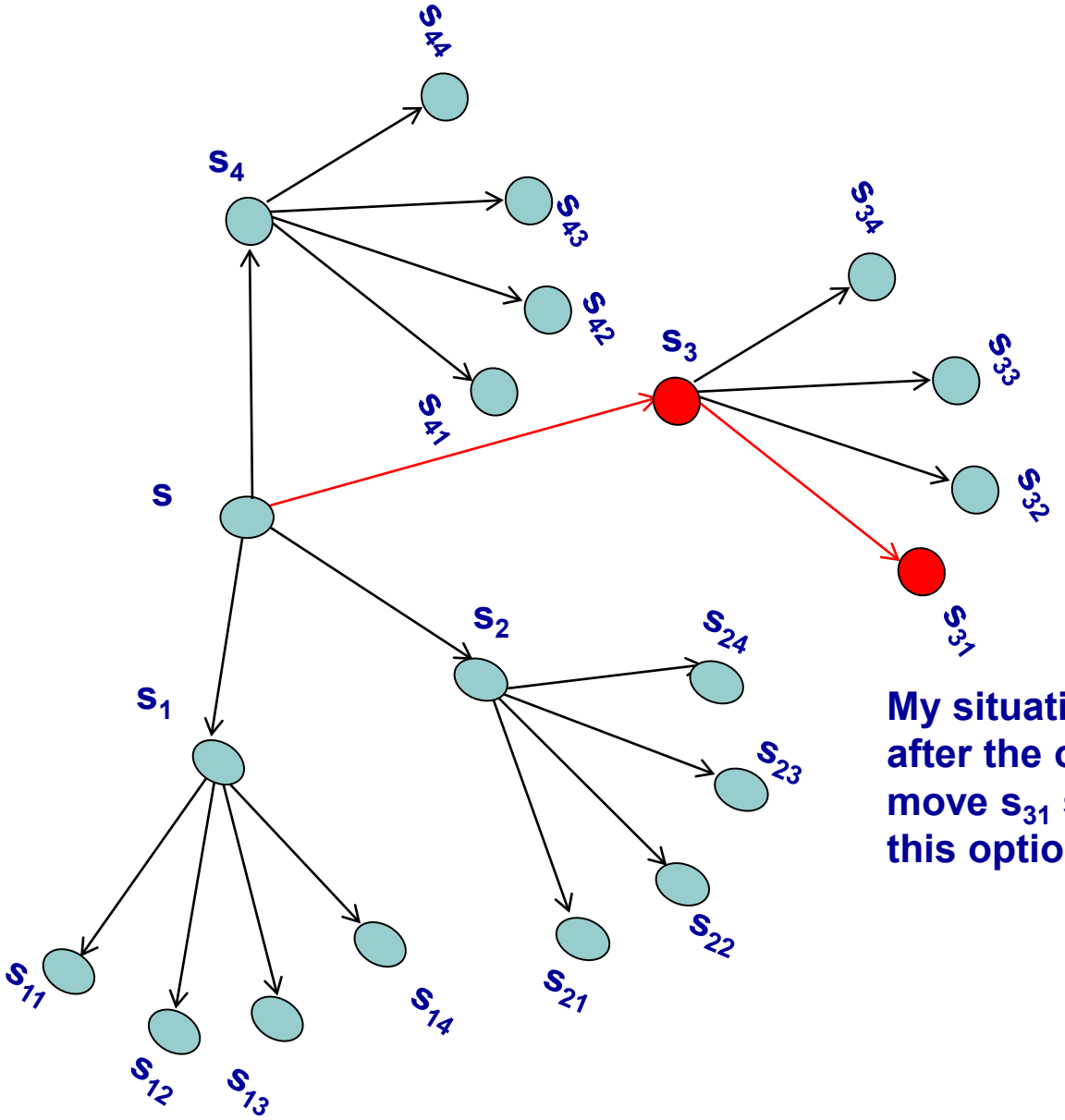
- Minmax is also called as min-max, minimax, MM, or saddle point, or maximin.
- Let us assume that the opponent plays as well as you.
- Let us compute events in the game after two rounds of moves, not just one round:

  ⇨ I have moved +
  the opponent has moved
  = what is the best move to do now?
- Thus, I make that move which is best for me after the opponent has made his/her move.
- In practice:
  - The values of the heuristic functions are computed to all the opponent's possible moves, and
  - that move is selected which minimizes the success of the opponent.

The previous algorithm:
MODULE play



The new algorithm using the minmax technique:
MODULE playbetter



My moves:

$s_1..s_4$

Opponent's moves:

$s_{11}..s_{44}$

My situation is best after the opponent's move s_{31} so I select this option!

Minmax for Tic-tac-toe

O		X
	X	

- O should move to s_7 not to lose.
- O has the options of the move s_i to squares 2, 4, 6, 7, 8, and 9.

- The used heuristic function estimates the success from the X's point of view

  => O tries **to minimize** the **maximum of** the **function** f(x) denoted as **g(s)** in each state.
- Since all options for O must be studied
  - let us compute the cases where O moves to the squares 2, 4, 6, 7, 8, and 9,
  - and then X makes the corresponding moves s_{ij}.

Minmax for Tic-tac-toe: how to compute g(s)?

O	O	X
	X	

1	2	3
4	5	6
7	8	9

For example, after O has moved to the square 2 then X computes the best move based on the heuristic function:

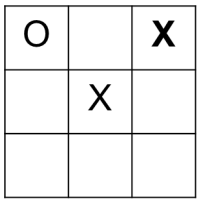
- X can move to the following squares j: 4, 6, 7, 8, 9.
- X selects the move s_{2j} which is the most beneficial so that g (s_2) is the maximum of the values of f.

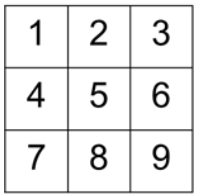
Move s_2:
$f(s_{24}) = 4–1 = 3$
$f(s_{26}) = 4–2 = 2$
$f(s_{27}) = 4–0 = 4$ maximum => $g(s_2) = 4$
$f(s_{28}) = 4–1 = 3$
$f(s_{29}) = 4–1 = 3$

The best move for X is s_{27}

O	O	X
	X	
X		

Tic-tac-toe: what is the best move for O?



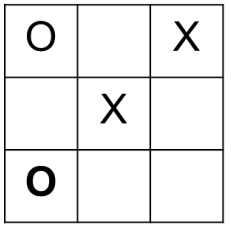


s2:
f(s24) = 4–1 = 3
f(s26) = 4–2 = 2
f(s27) = 4–0 = 4 maximum => g(s2) = 4
f(s28) = 4–1 = 3
f(s29) = 4–1 = 3

s4:
f(s42) = 4–2 = 2
f(s46) = 4–2 = 2
f(s47) = 4–0 = 4 maximum => g(s4) = 4
f(s48) = 4–1 = 3
f(s49) = 4–1 = 3

s6:
f(s62) = 3–2 = 1
f(s64) = 3–1 = 2
f(s67) = 3–0 = 3 maximum => g(s6) = 3
f(s68) = 3–1 = 2
f(s69) = 3–1 = 2

s7:
f(s72) = 3–2 = 1
f(s74) = 3–1 = 2
f(s76) = 3–2 = 1
f(s78) = 3–1 = 2
f(s79) = 3–1 = 2 maximum => **g(s7) = 2**



The minimum value of the function g values.

s8:
f(s82) = 3–2 = 1
f(s84) = 3–1 = 2
f(s86) = 3–2 = 1
f(s87) = 3–0 = 3 maximum => g(s8) = 3
f(s89) = 3–1 = 2

s9:
f(s92) = 3–2 = 1
f(s94) = 3–1 = 2
f(s96) = 3–2 = 1
f(s97) = 3–0 = 3 maximum => g(s9) = 3
f(s98) = 3–1 = 2

Minmax does not always work

- The minmax technique works, but it is still not optimal:
  - It does not consider those moves which cause to lose the game immediately.
  - It would also be wise to consider more than just two rounds of moves.
- Possible solutions:
  1. Compute cases about further rounds of moves:
     - $minmax^n$ technique.
     - It is being used in chess.
     - More computations are needed.
  2. Improve the used heuristic function as follows:
     - $f'_x(s) = \infty$, if X has in the state s a line of three.
     - $f'_x(s) = -\infty$, if O:lla in the state s a line of three.
     - $f'_x(s) = f_x(s)$, otherwise.
- In general, it is challenging to develop a good heuristic function to estimate how good are the moves.

Summary

- Deterministic and nondeterministic problems can be solved using search algorithms.
- Games belong to difficult nondeterministic problems which makes them interesting to the humans.
- Games are usually played by using search algorithms and heuristic functions which estimate how available choices to play the game affect the success to win the game.