Artificial Intelligence

CPSC 481

AI as Knowledge Representation and Search:

Heuristic Search for Problem Solving

Lecture Overview

- Heuristics in problem solving
- Heuristic search algorithms
  - Hill-climbing,
  - Simulated annealing,
  - Best-first search and A*,
  - Constraint satisfaction,
  - Mini-max for game playing
  - Advanced game playing
- Performance evaluation of heuristic search
  - Criteria for evaluation of heuristics
  - Complexity and efficiency issues
  - AI-complete and AI-hard

n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000
16	20922789888000
17	355687428096000
18	6402373705728000
19	121645100408832000
20	2432902008176640000
25	1.551121004×10^{25}
50	3.041409320×10^{64}
70	$1.197857167\times10^{100}$
100	$9.332621544\times10^{157}$
450	$1.733368733\times10^{1000}$
1000	$4.023872601\times10^{2567}$

Search space for Tic-Tac-Toe: 9!

Search space for chess: 64!

Search space for Go! game: 361!

10^{10} ms = 115.7 days

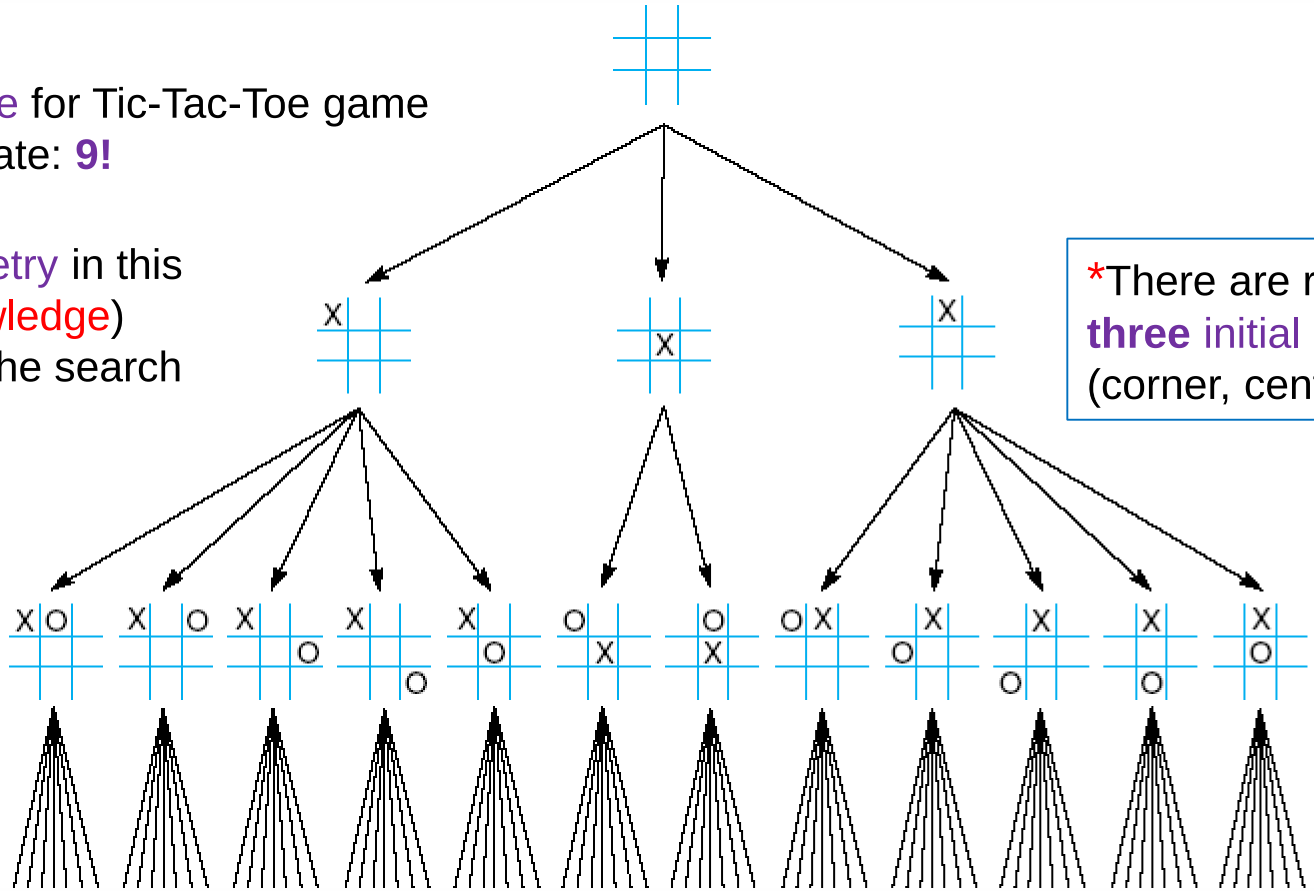
Heuristics

- Main problems of BFS and DFS
  - Brute force approach leading to combinatorial explosion
    - No utilization of information/strategies to make it efficient
- Many straightforward search algorithms can be improved by utilizing information and *intelligent strategy*, called "***heuristics***"
  - **Heuristics** is the study of the methods and rules of discovery and invention (Eureka "I have found it!"), commonly used by human.
  - Improvement can be made by intelligent strategies based on utilization of information or knowledge.
- Heuristic is necessary when there is no perfect solution.
  - E.g., playing chess or go game, many optimization problems, most AI problems.

First Three Levels of the Tic-Tac-Toe State Space Reduced by Symmetry

Search space for Tic-Tac-Toe game at the first state: **9!**

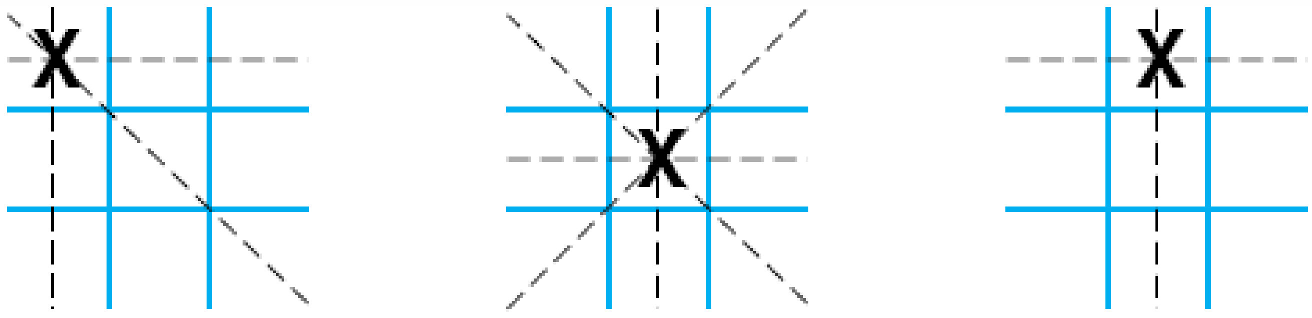
***But** symmetry in this game (knowledge) decreases the search space.



*There are really **three** initial moves (corner, center, side).

What heuristics can we use to determine better move?

The “most wins” Heuristic Applied to the First Children in Tic-Tac-Toe

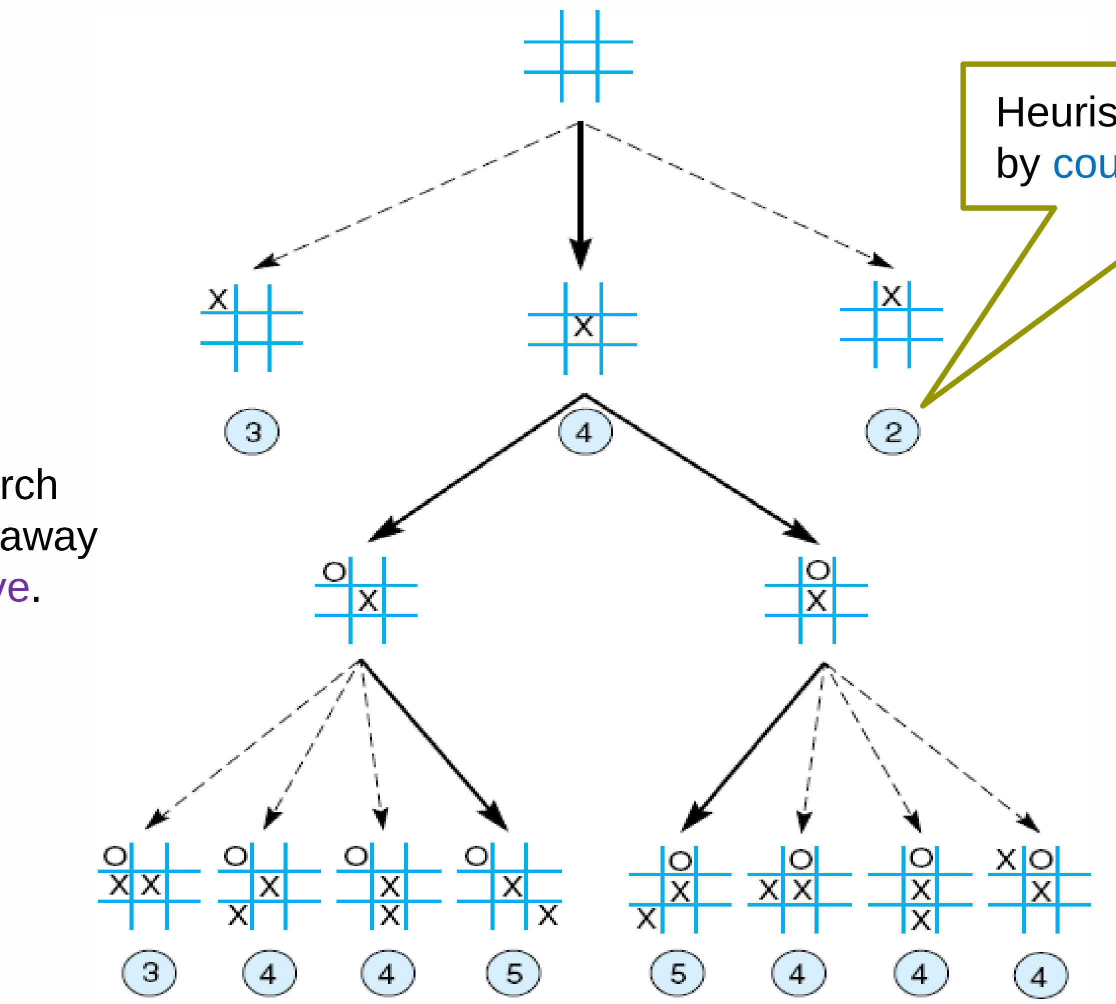


Three wins through a corner square

Four wins through the center square

Two wins through a side square

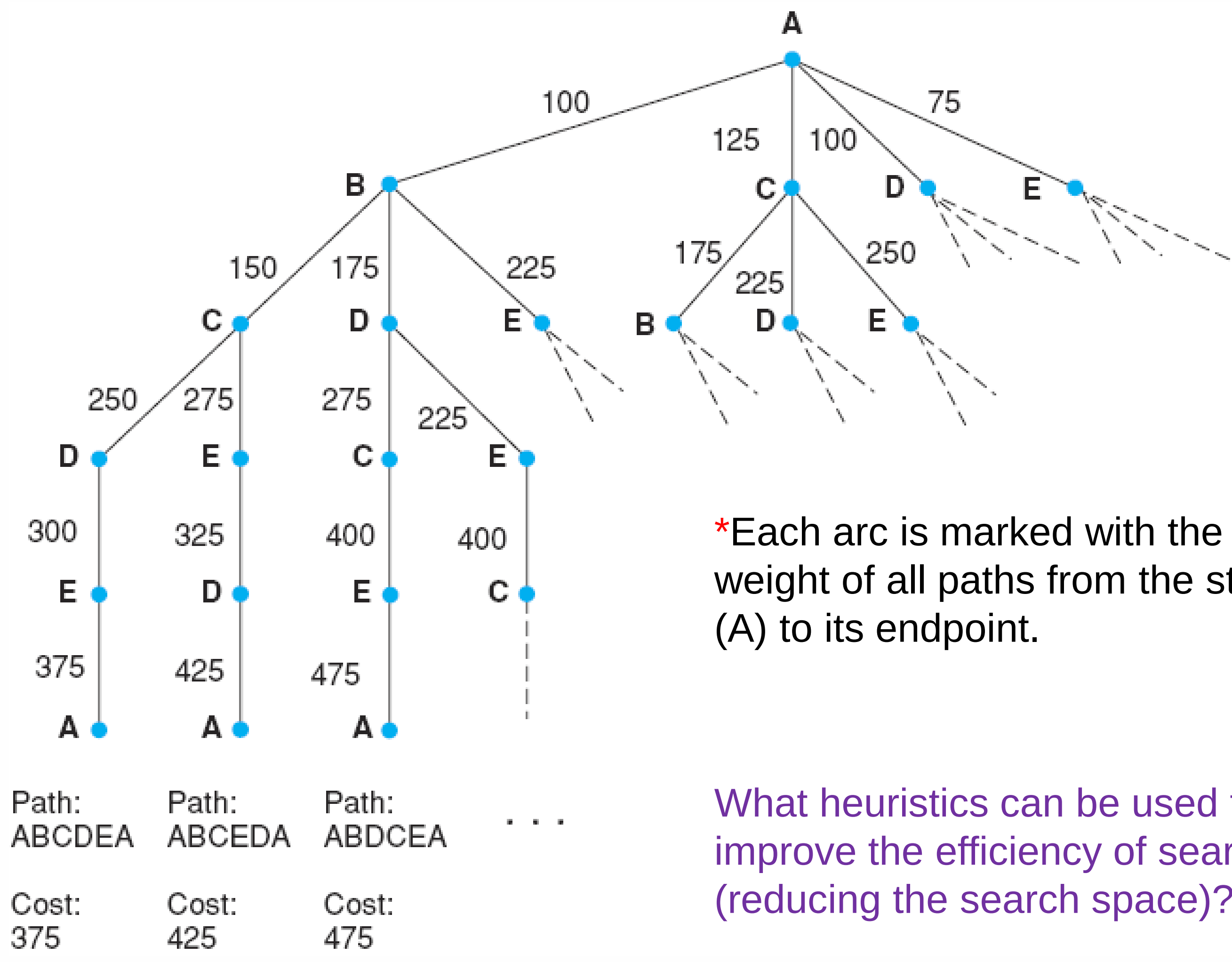
Heuristically Reduced State Space for Tic-Tac-Toe



Heuristic value calculated by counting possible wins

2/3 of all the search space is pruned away with the first move.

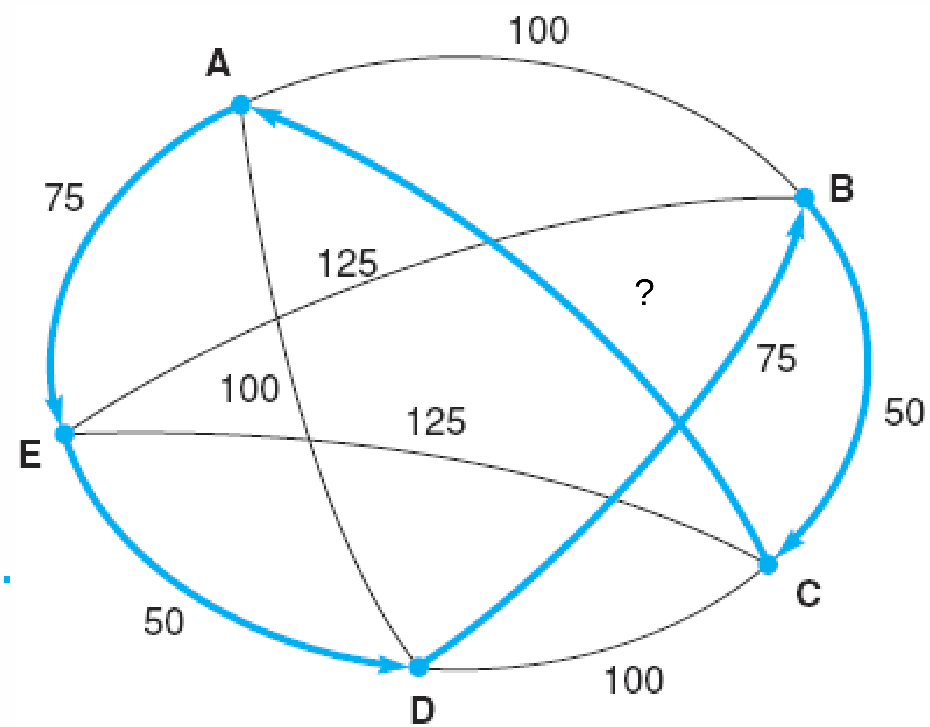
Search Space for the Travelling Salesman Problem



*Each arc is marked with the total weight of all paths from the start node (A) to its endpoint.

What heuristics can be used to improve the efficiency of search (reducing the search space)?

Looking for the Shortest Path with the Nearest Neighbor Path



*Neighbor path is in **bold.**

Is it optimal solution?

***Note**: if (A,C) is 125, this path (A, E, D, B, C, A), at a cost of **375**, is the shortest path. If (A, C) is 300, then the cost is 550, not the shortest. The comparatively high cost of arc (C, A) defeated the heuristic (**nearest neighbour**) but it is better than most random paths.

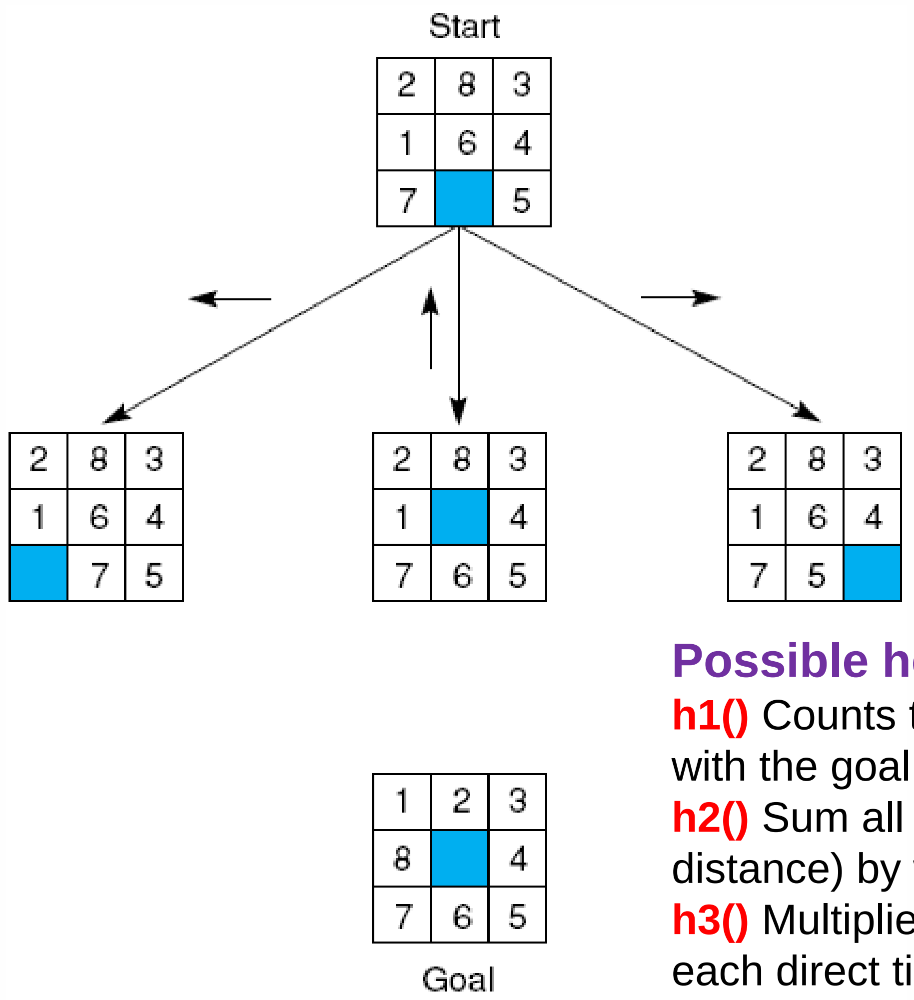
How to Define Heuristics

- General approaches to define heuristics
  - Identify the goal of the problem.
  - Collect and analyze available information and use background/domain knowledge to achieve the goal.
  - Identify all possible states and eliminate unnecessary states using the information to reduce the search space if possible.
  - Define a method(s) to quantitatively evaluate each state (heuristics).
  - Formulate the heuristics into a function (heuristic function).
- Try some examples
  - Tic-tac-toe, 8-puzzle, Chess, Go
  - Auto pilot for car, Mars rover, Robot vacuum

How to Define a Heuristic Function

- A heuristic function, ***f(n) = g(n) + h(n)***
  - **g(n)**: the measure (distance or cost) from the start to current state, n,
    - E.g., count 0 for the beginning state and is incremented by 1 for each level of the search.
  - **h(n)**: a heuristic ***estimate*** of the measure from state **n** to a **goal**.
- **h()** guides search toward heuristically promising states while **g()** **prevents** search from indefinitely fruitless path.
  - If two states are the same or nearly the same, it is generally preferable to examine the state that is **nearest** to the root state of the graph (initial state) since it will give a greater probability of being on the shortest path to the goal.
- **Cost vs. Reward**
  - If **f()** is a cost function, the smaller the better.
  - If **f()** is a reward function, the larger the better.

Some Heuristics for the 8-puzzle Game



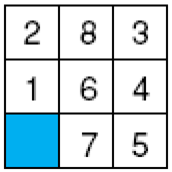
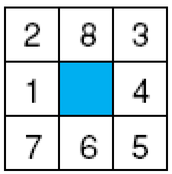
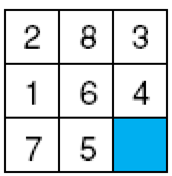
Possible heuristics h():

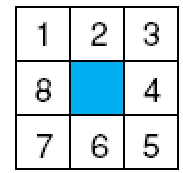
h1() Counts the tiles out of place compared with the goal state in each state.

h2() Sum all the distances (Manhattan distance) by which the tiles are out of place.

h3() Multiplies a small number, e.g., 2 times each direct tile reversal.

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

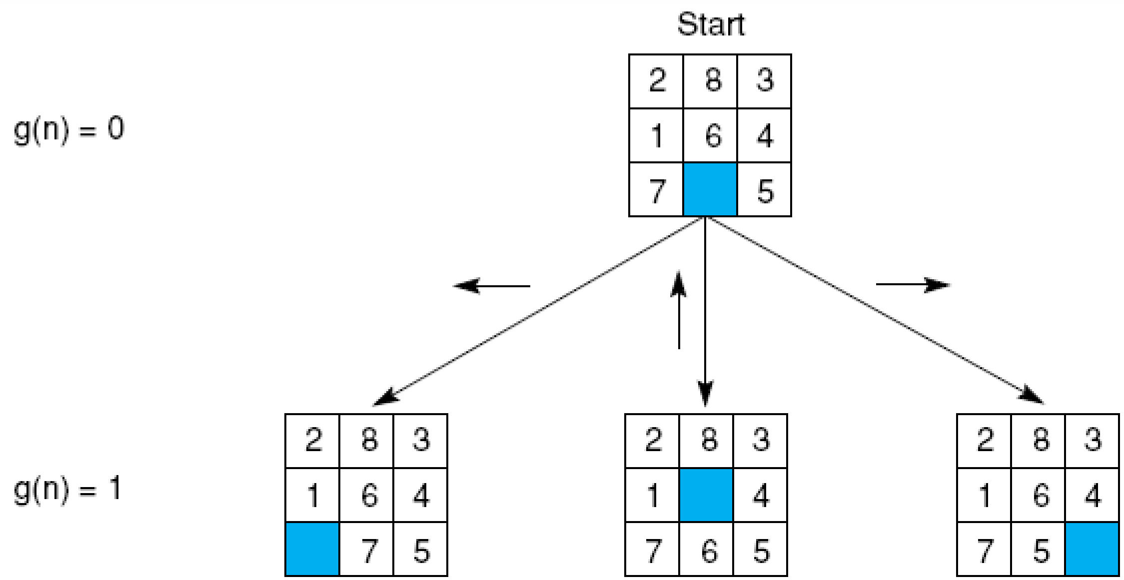
h1(n) **h2(n)** **h3(n)** n represents a state, e.g., s1, s2, s3

*What could be the problem of each heuristic?

*What happens if a heuristic function returns non-unique and conflicting scores?

* Devising good heuristics based on the limited information is not easy but critical in solving a complex problem!

The Heuristic *f()* Applied to States in the 8-puzzle



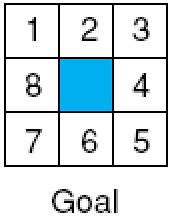
Values of f(n) for each state, 6 4 6

where:

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

g(n) = actual distance from n to the start state, and

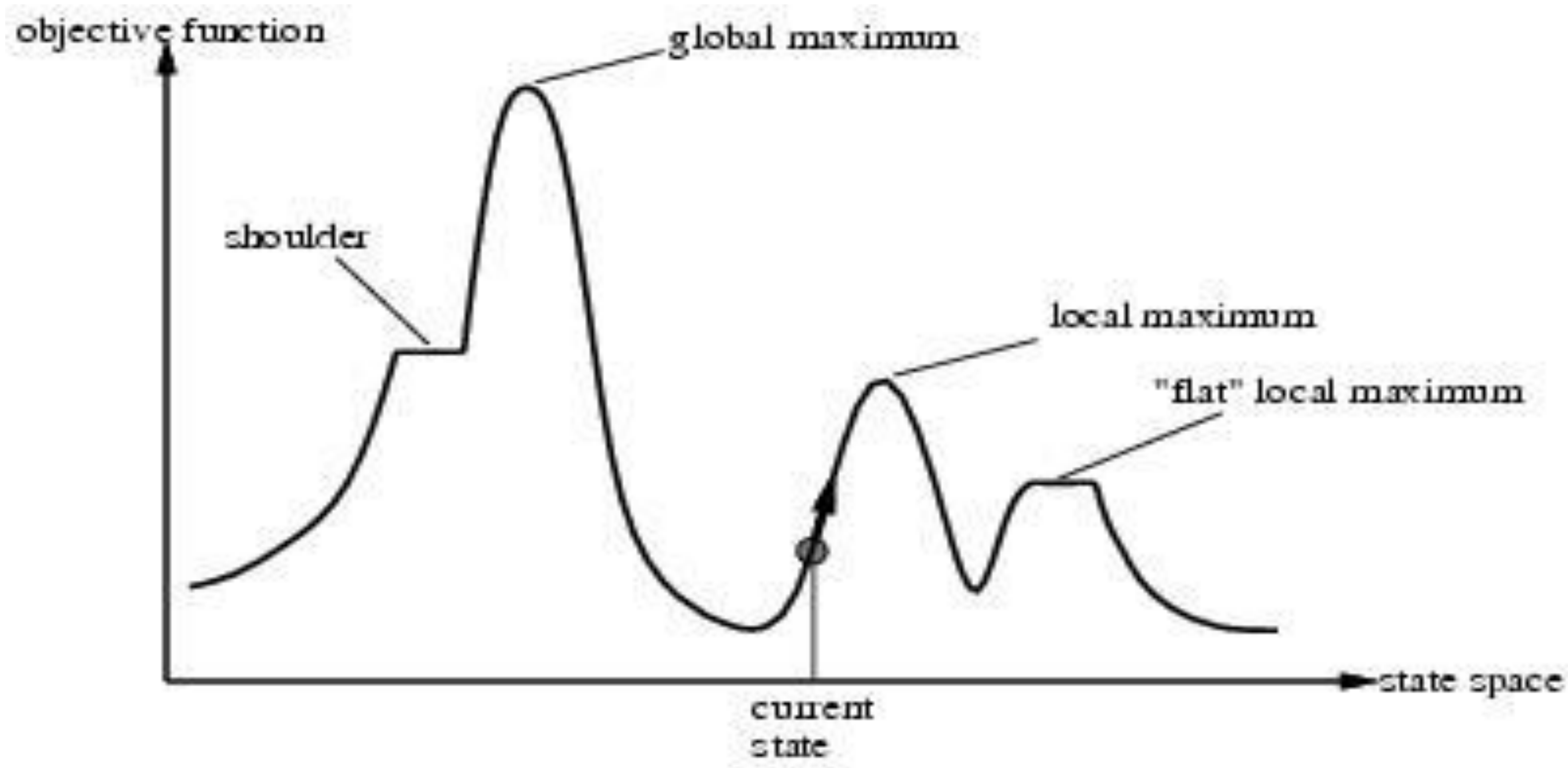
h(n) = number of tiles out of place.



Try to code the heuristic function, f().

How can we utilize f(n) in DFS?

Hill-climbing



“Landscape” of search for max value

Simple Hill-climbing search

Algorithm sketch

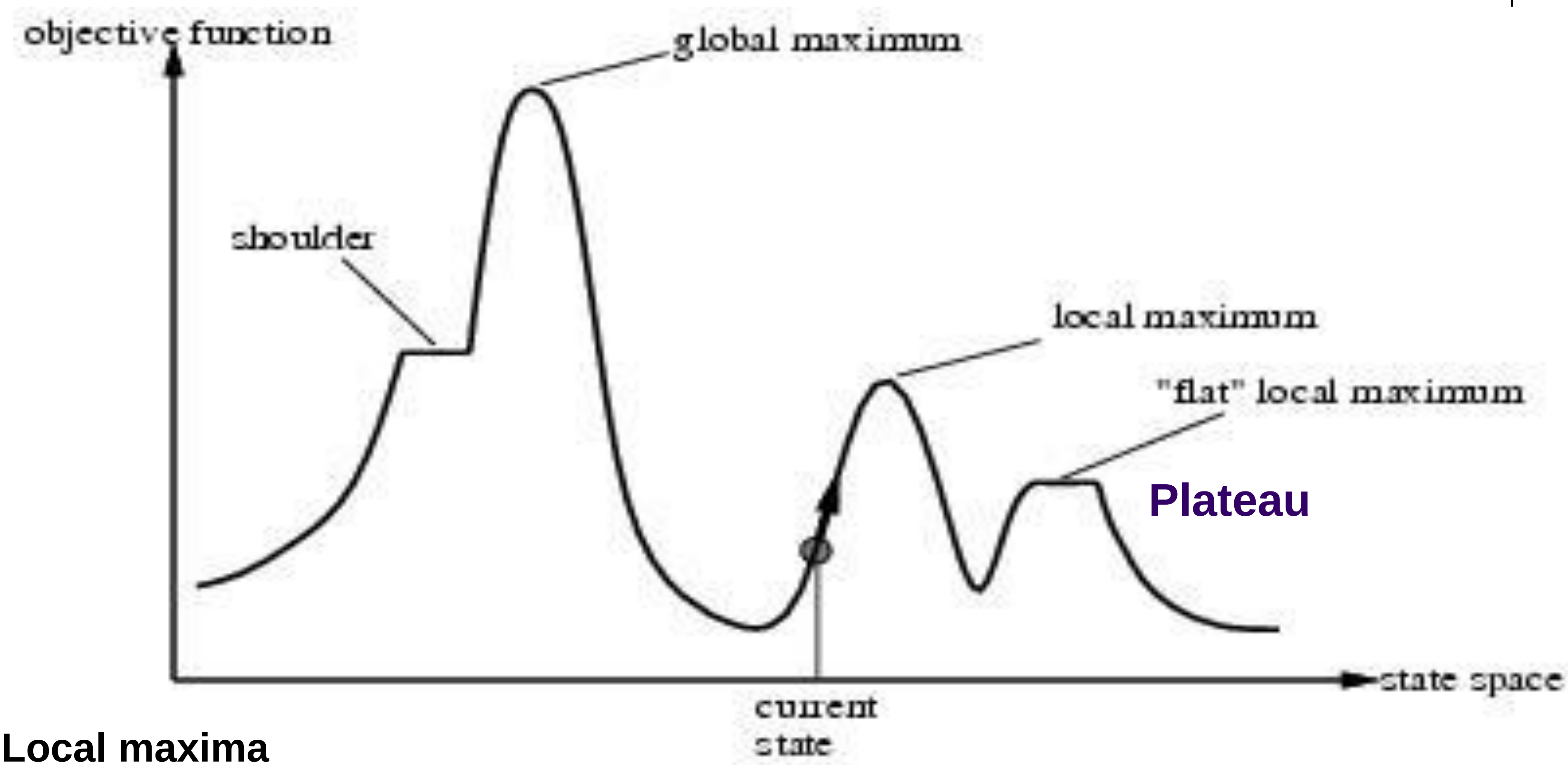
1. Check if the current state is a solution, if so return it else go to next step.
2. Expand the current state of the search.
3. Evaluate its children states using a heuristic function.
4. Select the FIRST **better state** for further expansion.

Continue steps 1 - 4 until it finds a solution or reach no better state.

<u>Simple</u> Hill-climbing as a Heuristic Search

- **Local search**
  - Keep track of single current state
  - Move only to neighboring states
  - Ignore paths
- **Advantages:**
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.
  - One of the simplest search methods based on heuristics. It works like DFS that utilizes a heuristic.

Problems of Simple Hill-climbing



Local maxima

may get stuck and may fail to find the best solution when it reaches a state that has no other better states.

Plateau

may get confused by the result of evaluation when the best is not clear

Shoulder

may get confused by the result of evaluation when the best is not the direct successor

Variations of Hill-climbing

- **Simple Hill-climbing** focusing on **step 3**
  1. Check if the current state is a solution, if so return it else go to next step.
  2. Expand the current state of the search.
  3. Evaluate its children states.
  4. Select the FIRST better state for further expansion.

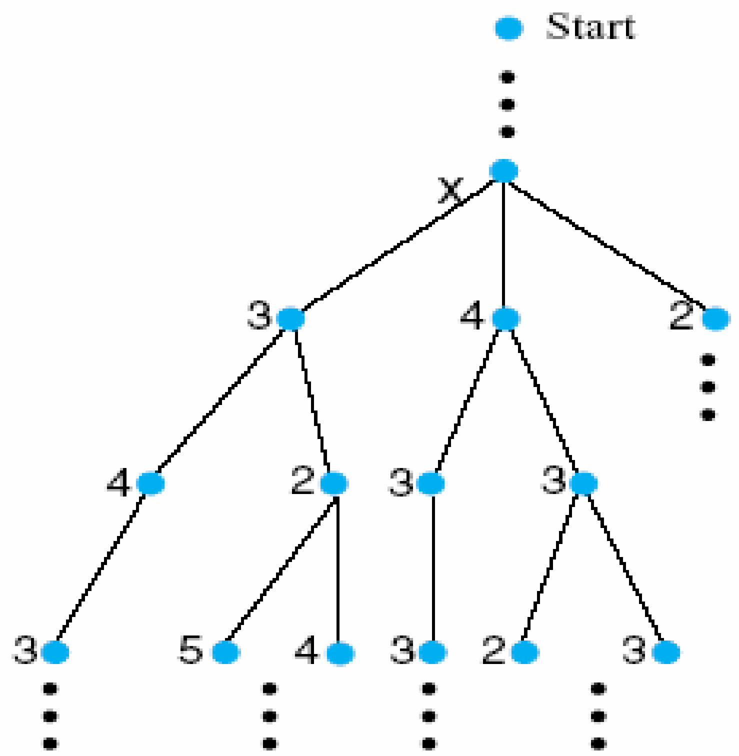
  Continue steps 1 – 4 until it finds a solution or reach a no better state.
- **Improvement** at step 3
  - **All** successors are compared and select the best state.
    - Called Steepest-ascent/descent Hill-climbing or Gradient Search
      - The decision on descent or ascent **depends on the heuristics**.

Gradient ascent/descent search

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum
input: *problem*, a problem
local variables: *current*, **a node.**
neighbor, **a node.**

current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
loop do
neighbor $\leftarrow$ a highest valued successor of *current*
if VALUE [*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]
current $\leftarrow$ *neighbor*

Plateau Problem in Hill-Climbing with 3-Level Look Ahead



*Hill-climbing can get confused in this case (**plateau**) as the cost of all paths are the same or similar.

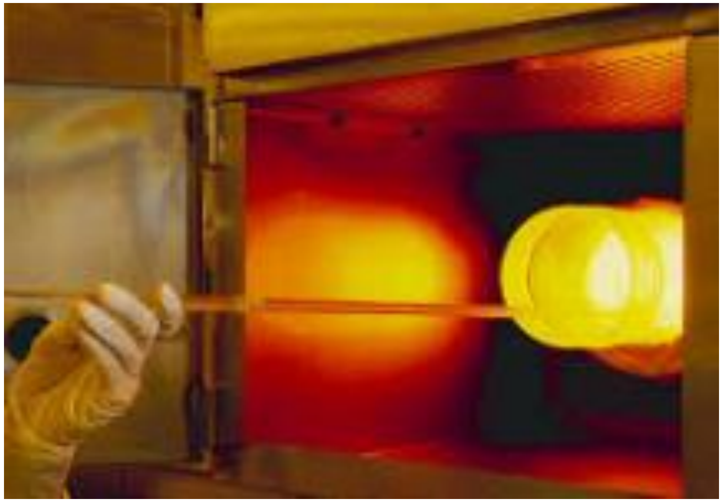
How can we handle the local maxima problem?

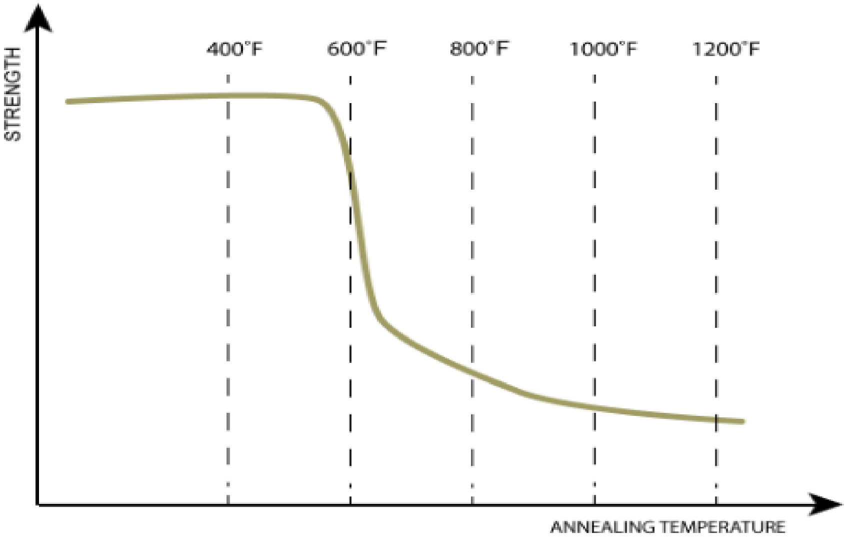
Student Participation: Hill-climbing

Can these variations of Hill-climbing solve local maximum, shoulder, plateau problems?

Simulated Annealing (SA)

- The name and inspiration of SA come from **annealing in metallurgy**, a technique involving heating and controlled cooling of a material to reduce their defects.
- A generic probabilistic meta-heuristic for the **global optimization problem**.

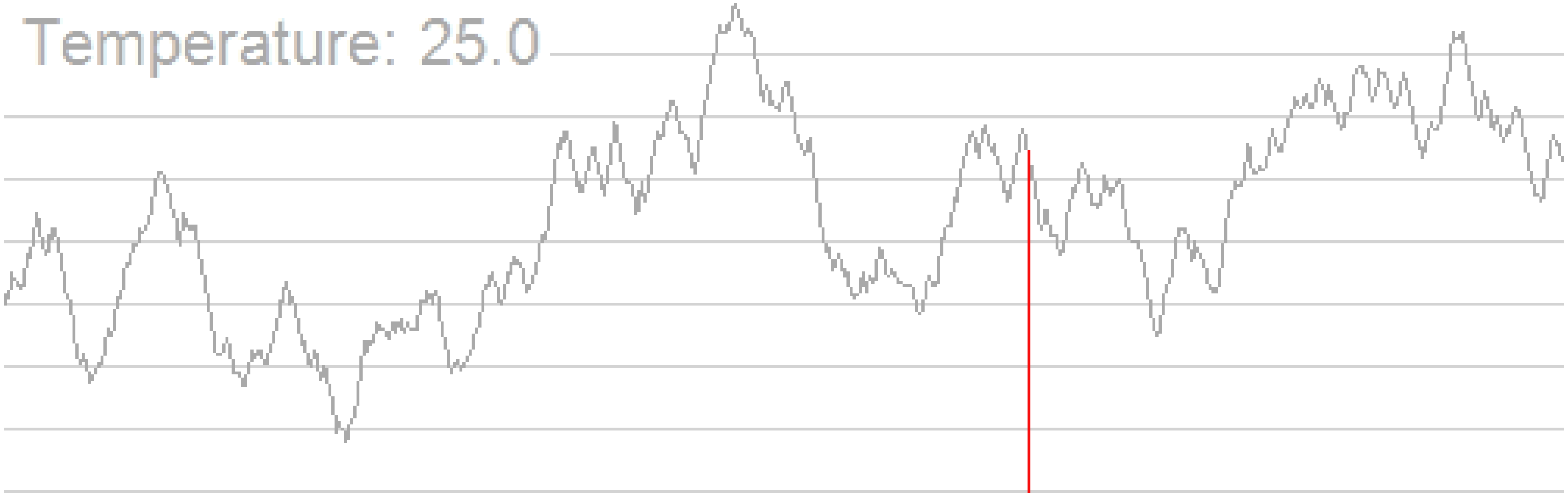




Physical Interpretation of Simulated Annealing

- Annealing = physical process of cooling a liquid or metal until particles achieve a certain frozen crystal state
  - simulated annealing:
    - free variables are like particles
    - seek “low energy” (high quality) configuration
    - get this by slowly reducing temperature T, which particles move around randomly

Is SA search able to handle the local maxima?



Despite the many local maxima in this graph, the global maximum can still be found using **simulated annealing**.

Search using Simulated Annealing

- Simulated Annealing = hill-climbing with **non-deterministic** search
- Basic ideas:
  - like hill-climbing identify the quality of the local improvements
  - instead of picking the best move, pick one **randomly**
  - say the change in objective function is Δ (big delta)
  - if Δ is positive, then move to that state
  - otherwise:
    - move to this state with **probability proportional to Δ**
    - thus: worse moves (very large negative Δ) are executed less often
  - however, there is always **a chance of escaping** from local maxima over time, make it less likely to accept locally bad moves
  - Can also make the size of the move random as well, i.e., allow "large" steps in state space

Simulated Annealing

function SIMULATED-ANNEALING(*problem, schedule*) **return** a solution state
input: *problem*, a problem
schedule, a mapping from time to temperature
local variables: *current***,** a node.
*next***,** a node.
*T***,** a "temperature" controlling the probability of downward steps

current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
for t $\leftarrow$ 1 to ∞ do
T $\leftarrow$ *schedule*[*t*]
if *T = 0* **then return** *current*
next $\leftarrow$ a randomly selected successor of *current*
$\Delta E \leftarrow$ VALUE[*next*] - VALUE[*current*]
if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
else *current* $\leftarrow$ *next* only with probability $e^{\Delta E/T}$

Algorithm sketch

At each step, the SA heuristic considers some neighbor, s' of the current state s, and *probabilistically* decides moving from s to s'.

The probabilities are chosen so that the system ultimately tends to move to states of **lower energy** (annealing schedule, p = $e^{\Delta E/T}$).

Defining an annealing schedule, p is critical.

This step is repeated until the system reaches a state that is solution, good enough for the application, or until a given computation budget has been exhausted.

More Details on Simulated Annealing

- Lets say there are 3 moves available, with changes in the objective function of $\Delta E_1 = -0.1$, $\Delta E_2 = 0.5$, $\Delta E_3 = -5$. (Let T = 1).
- pick a move randomly:
  - if ΔE_2 is picked, move there.
  - if ΔE_1 or ΔE_3 are picked, probability of move = $\exp(\Delta E/T)$
  - move 1: prob1 = exp(-0.1) = 0.9,
    - i.e., 90% of the time we will accept this move
  - move 3: prob3 = exp(-5) = 0.05
    - i.e., 5% of the time we will accept this move

- T = "temperature" parameter
  - high T => probability of "locally bad" move is higher
  - low T => probability of "locally bad" move is lower
  - typically, T is decreased as the algorithm runs longer
    - i.e., there is a "temperature schedule"

Simulated Annealing in Practice

- The method was proposed in 1983 by IBM researchers for solving VLSI layout problems (Kirkpatrick et al, *Science*, 220:671-680, 1983).
  - theoretically will always find the global optimum (the best solution)

Problems of SA

- Useful for some problems, but can be very **slow**
  - slowness comes about because T must be decreased very gradually to retain optimality
  - In practice **how do we decide the rate at which to decrease T**? (this is a practical problem with this method)
- Unfortunately, the applicability of simulated annealing is **problem-specific** because it relies on finding **lucky jumps** that improve the position. In such extreme examples, hill climbing will most probably produce a local maxima.

References

- George Fluger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition, **Chapter 4**, Addison Wesley, 2009.
- Russel and Norvig, Artificial Intelligence: A Modern Approach, 3rd edition, Prentice Hall, 2010.