Introduction to AI

Lecture 5
Informed Search

Dr. Tamal Ghosh
Department of CSE
Adamas University

Informed Search: Add Domain-Specific Information

- Add domain-specific information to select what is the best path to continue searching along
- Define a heuristic function, h(n), that estimates the "goodness" of a node n with respect to reaching a goal.
- Specifically, h(n) = estimated cost (or distance) of minimal cost path from n to a goal state.
- **h(n) is about cost of the future search, g(n) past search (path cost from start to node n)**
- h(n) is an estimate (rule of thumb), based on domain-specific information that is **computable** from the current state description. Heuristics do not guarantee feasible solutions and are often without theoretical basis.

Heuristics

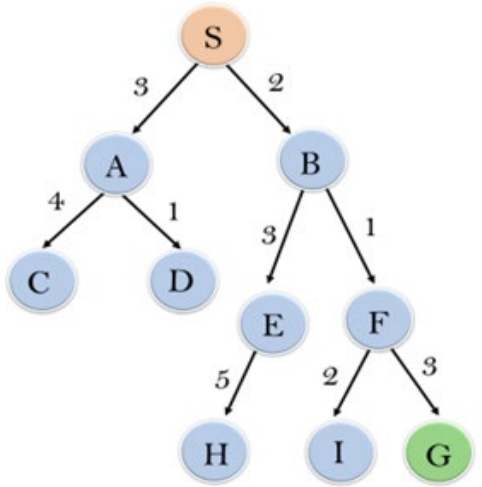
- In general:
  - h(n) >= 0 for all nodes n
  - h(n) = 0 implies that n is a goal node
  - h(n) = infinity implies that n is a dead-end from which a goal cannot be reached

Best First Search

- Order nodes on the OPEN list by increasing value of an evaluation function, f(n) , that incorporates domain-specific information in some way.
- Example of f(n):
  - f(n) = g(n) (uniform-cost)
  - f(n) = h(n) (greedy algorithm)
  - f(n) = g(n) + h(n) (**algorithm A**)
- This is a generic way of referring to the class of informed methods.

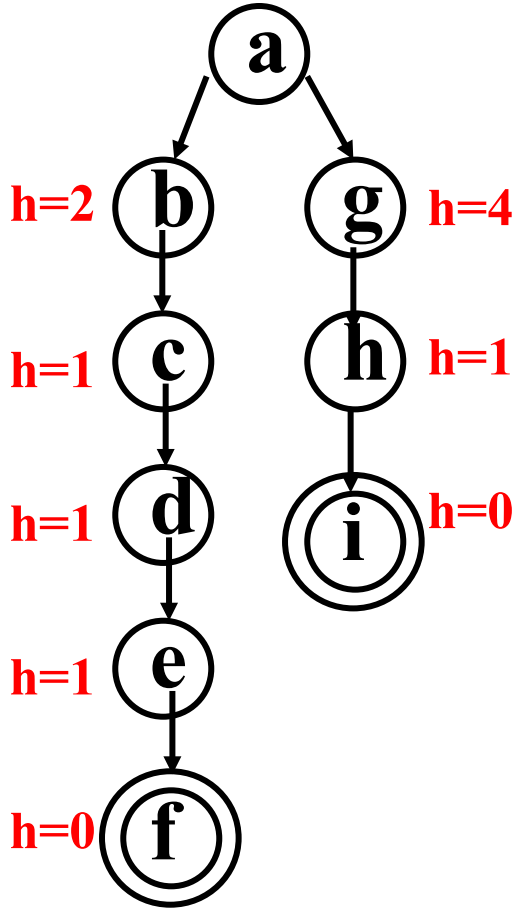
Best First Search

- Here we have a graph where our aim is to traverse from the node S to node G
- Create open and close , initally. Open : [S], Closed: []
- First we pop node S and move it to the closed list and the children nodes are added to open. Open: [B, A], Closed: [S]
- 2nd, the heuristic value of nodes A and B are compared , B is popped and moved to the closed list. Neighboring nodes of B are pushed to the open list. Open: [F, E, A], Closed: [S,B]
- 3rd the heuristic values of E, F and A are compared and since has lowest heuristic it is added to the closed list.Neighbors of are added to the open list. Open: [I,G,E,A], Closed: [S,B,F]
- For the fourth iteration we have our goal node in the open list hence we select that and move it to the closed list.
- Open: [I,E,A] Closed: [S,B,F,G]
- The path taken is S->B->F->G



Greedy Search

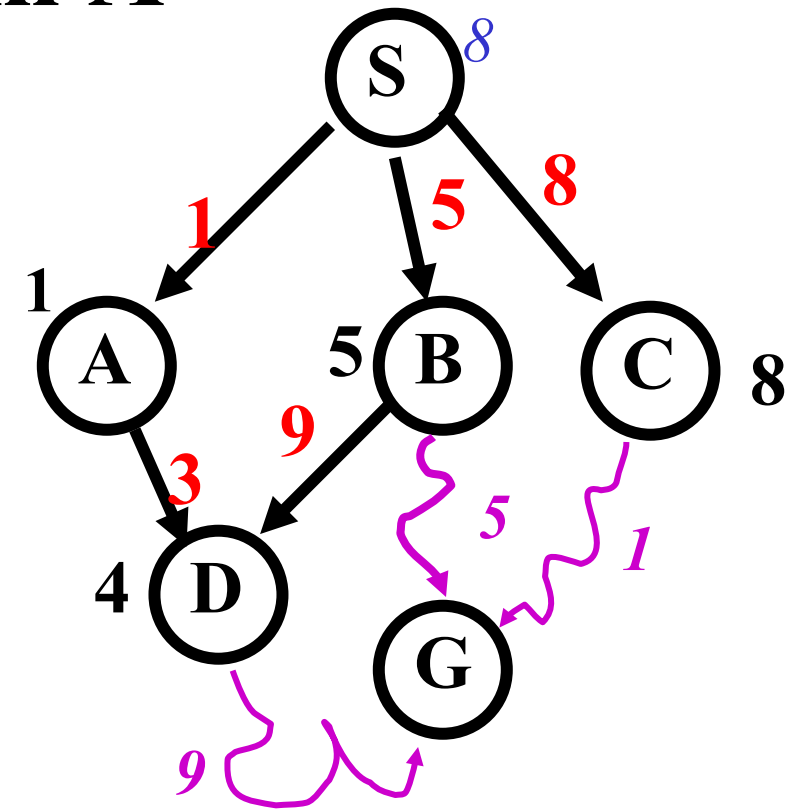
- Evaluation function f(n) = h(n), sorting open nodes by increasing values of f.
- Selects node to expand believed to be closest (hence it's "greedy") to a goal node (i.e., smallest f = h value)
- Not admissible, as in the example. Assuming all arc costs are 1, then Greedy search will find goal f, which has a solution cost of 5, while the optimal solution is the path to goal i with cost 3.
- Not complete (if no duplicate check)



Algorithm A

- Use as an evaluation function

  f(n) = g(n) + h(n)
- The h(n) term represents a “depth-first“ factor in f(n)
- g(n) = minimal cost path from the start state to state n **generated so far**
- The g(n) term adds a "breadth-first" component to f(n).
- Rank nodes on OPEN list by estimated cost of solution from start node through the given node to goal.
- Not complete if h(n) can equal infinity.
- Not admissible



(h*(D)=2, h*(C)=2)

f(D)=4+9=13
f(B)=5+5=10
f(C)=8+1=9

C is chosen next to expand

Algorithm A

OPEN := {S}; CLOSED := {};
repeat
    Select node n from OPEN with minimal f(n) and place n on CLOSED;
    if n is a goal node **exit** with success;
    Expand(n);
    For each child n' of n do
        if n' is not already on OPEN or CLOSED **then**
            put n′ on OPEN; set backpointer from n' to n
            compute h(n'), g(n')=g(n)+ c(n,n'), f(n')=g(n')+h(n');
        else if n' is already on OPEN or CLOSED and **if** g(n') is lower for the new version of n' **then**
            discard the old version of n';
            Put n' on OPEN; set backpointer from n' to n
until OPEN = {};
exit with failure

Algorithm A*

- **h*(n)** = true cost of the minimal cost path from n to any goal.
- **g*(n)** = true cost of the minimal cost path from S to n.
- **f*(n) =** h*(n)+g*(n) = true cost of the minimal cost solution path from S to to any goal going through n.
- A* is algorithm A with constraint that **h(n) <= h*(n)**
- h is **admissible** when h(n) <= h*(n) holds for every n.
- Using an admissible heuristic guarantees that the first solution found will be an optimal one.
- A* is **complete** whenever the branching factor is finite, and every operator has a fixed positive cost (total # of nodes with f(.) <= f*(goal) is finite)
- A* is **admissible**

Some Observations on A*

- **Null heuristic:** If h(n) = 0 for all n, then this is an admissible heuristic and A* acts like Uniform-Cost Search.
- **Better heuristic:** If h1(n) <= h2(n) <= h*(n) for all non-goal nodes, then h2 is a better heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*.
  - In other words, A1 expands at least as many nodes as A2*.
  - We say that A2* is better informed than A1*.
- The closer h is to h*, the fewer extra nodes will be expanded
- **Perfect heuristic:** If h(n) = h*(n) for all n, then only the nodes on the optimal solution path will be expanded. So, no extra work will be performed.

Example 1

- Given an initial state of a 8-puzzle problem and final state to be reached-

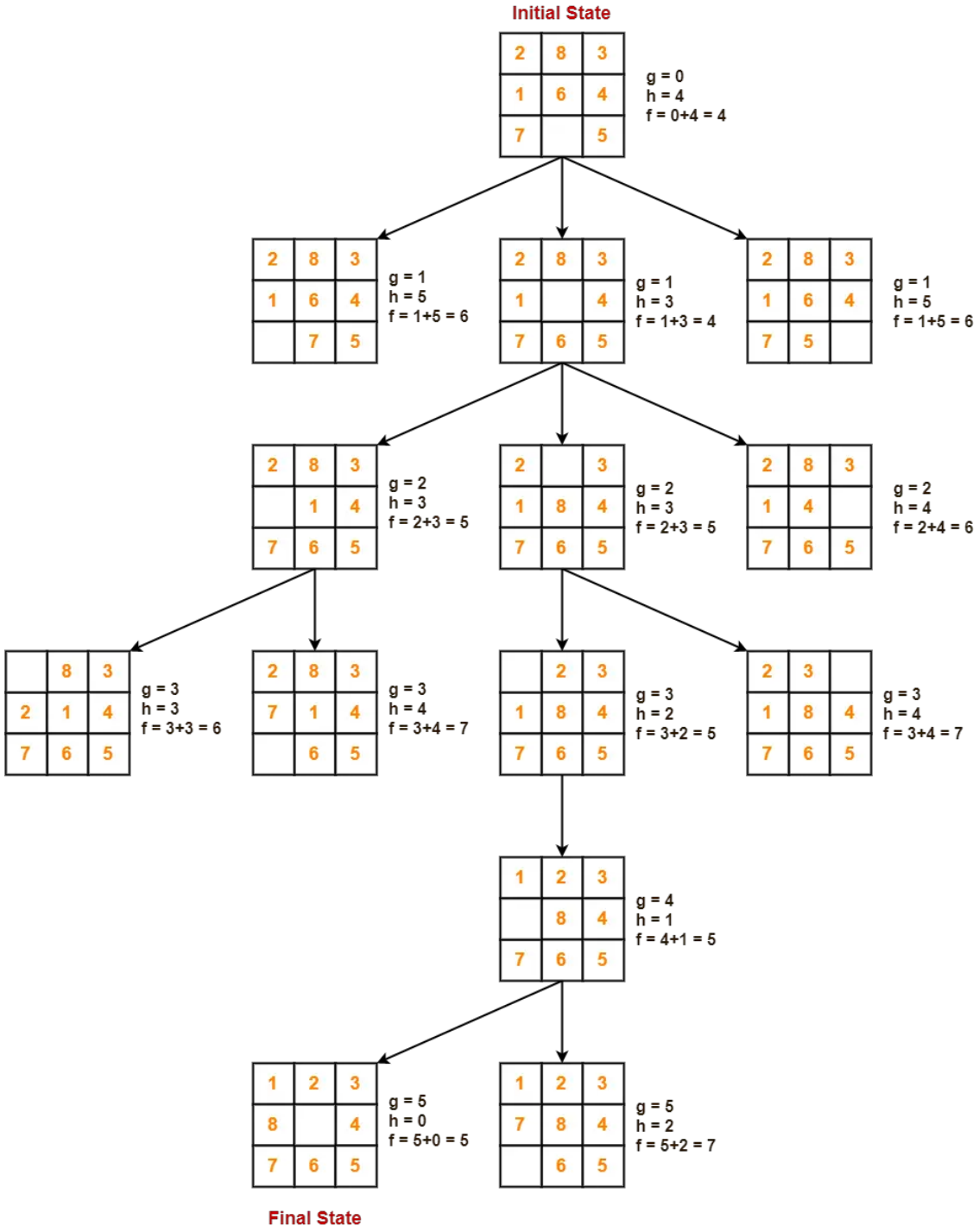
Initial State

Final State

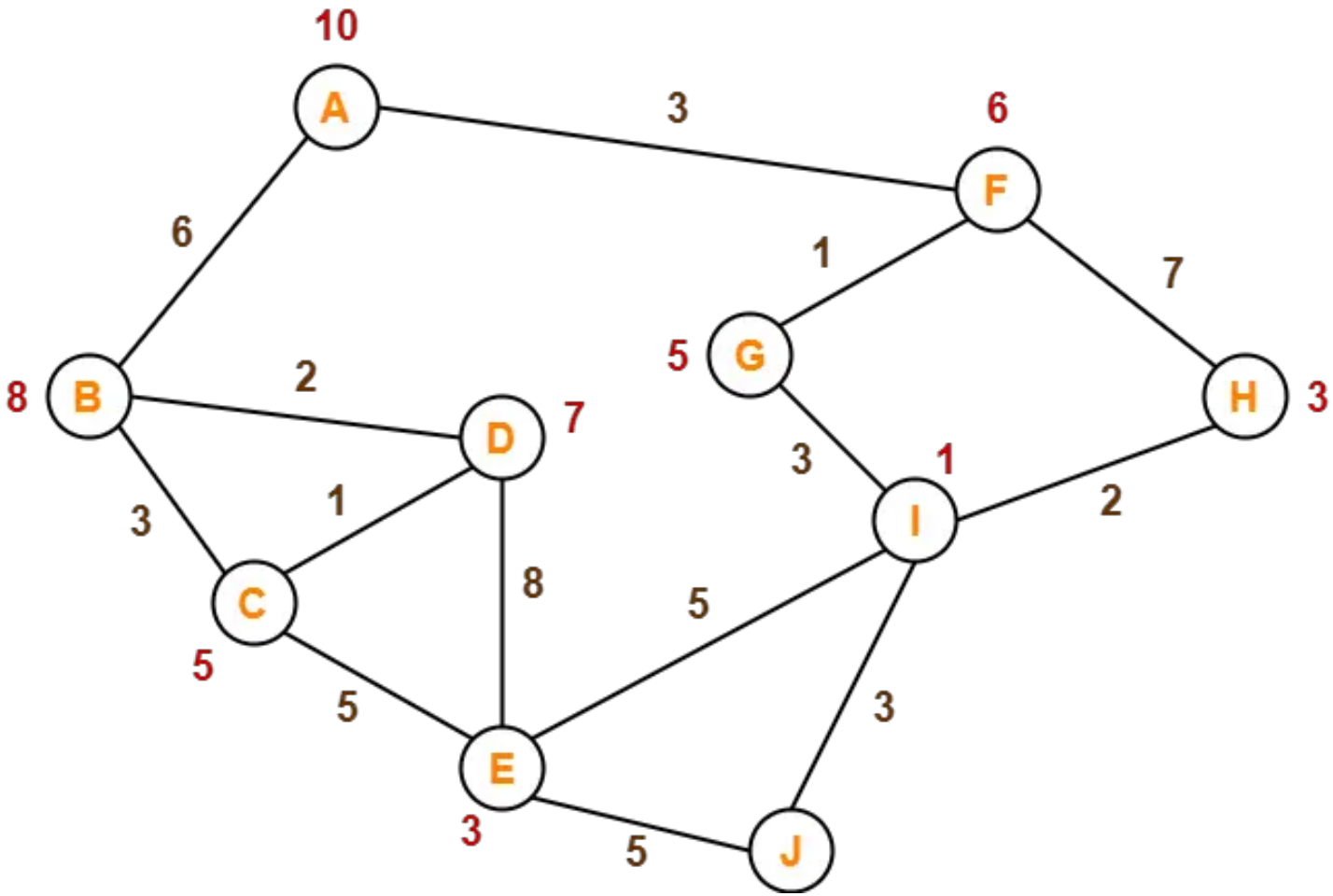
- Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
- Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

Solution

- A* Algorithm maintains a tree of paths originating at the initial state.
- It extends those paths one edge at a time.
- It continues until final state is reached.



Example 2

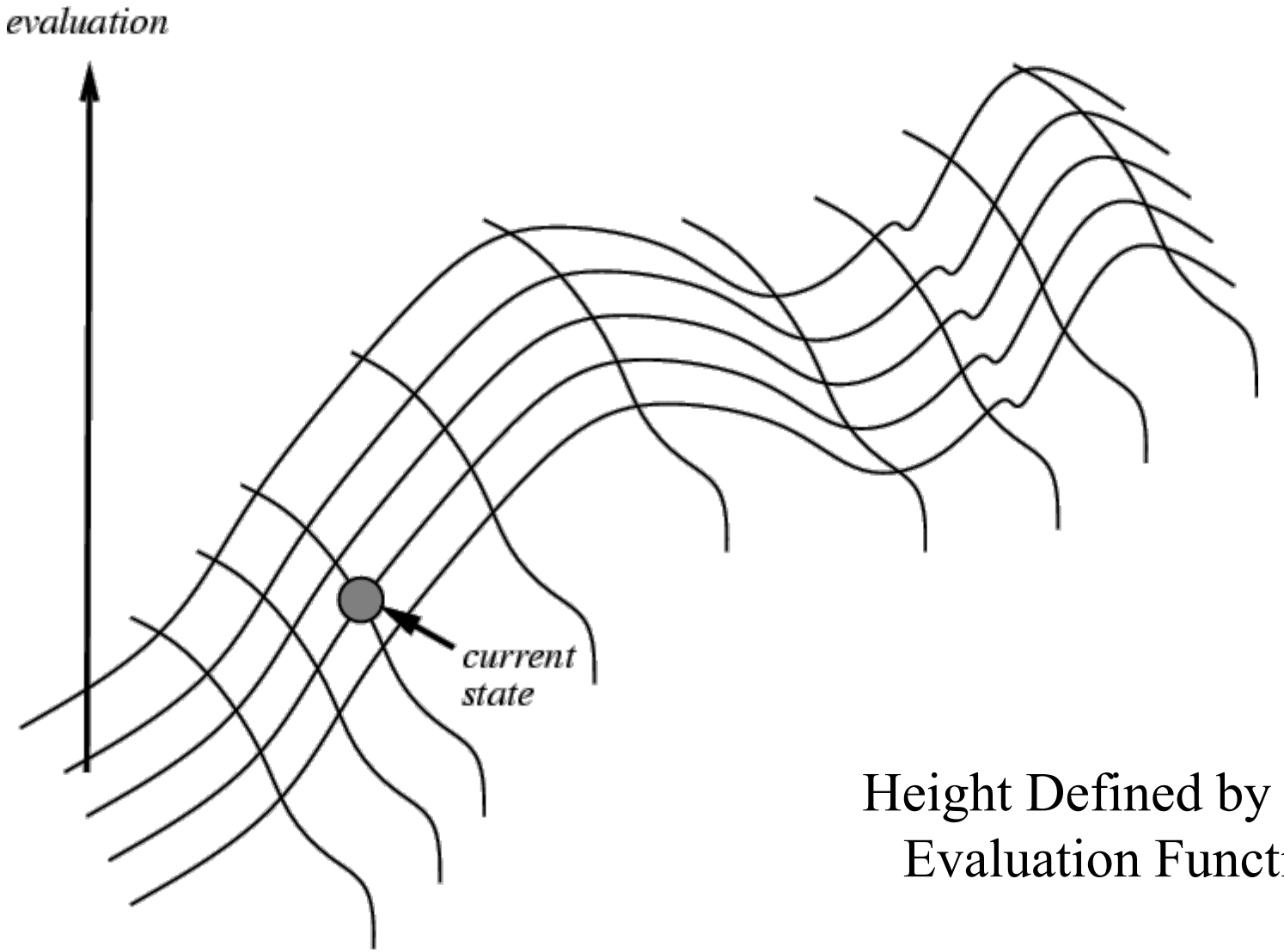


- It is important to note that-
- A* Algorithm is one of the best path-finding algorithms.
- But it does not produce the shortest path always.
- This is because it heavily depends on heuristics.

Iterative Improvement Search

- Another approach to search involves starting with an initial guess at a solution and gradually improving it until it is one.
- Some examples:
  - Hill Climbing
  - Simulated Annealing
  - Genetic algorithm

Hill Climbing on a Surface of States

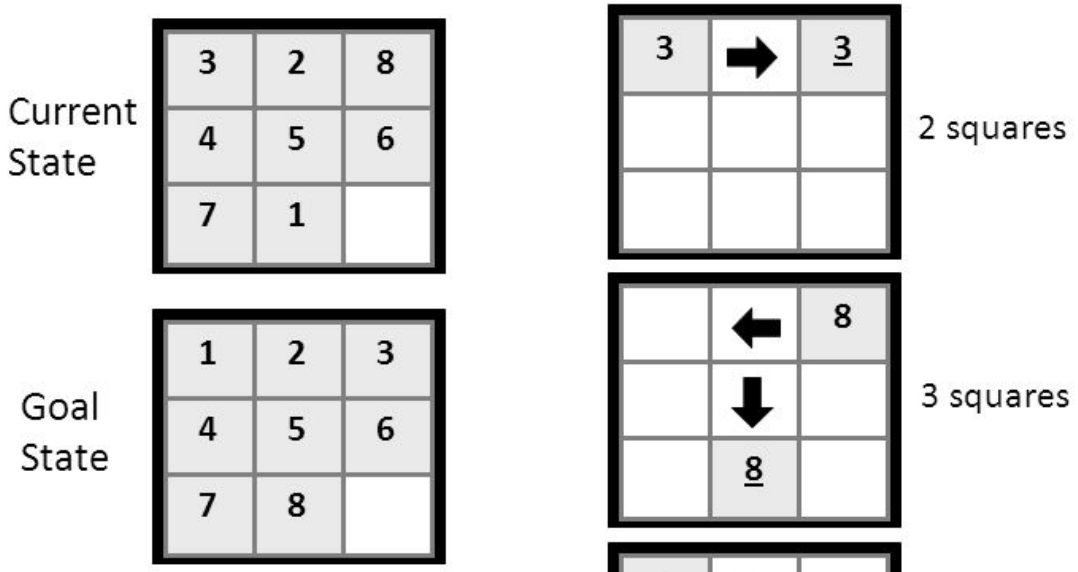


Height Defined by
Evaluation Function

Hill Climbing Search

- If there exists a successor n’ for the current state n such that
  - h(n’) < h(n)
  - h(n’) <= h(t) for all the successors t of n,
- then move from n to n’. Otherwise, halt at n (local optima).
- A random restart is required to avoid local maxima/minima.
- Looks one step ahead to determine if any successor is better than the current state; if there is, move to the best successor.
- Similar to Greedy search in that it uses h, but does not allow backtracking or jumping to an alternative path since it doesn’t “remember” where it has been.
- OPEN = {current-node}
- Not complete since the search will terminate at "local minima," "plateaus," and "ridges."

Calculate Manhattan Heuristic for 8 puzzle



•The **Manhattan Distance** (not including the blank)

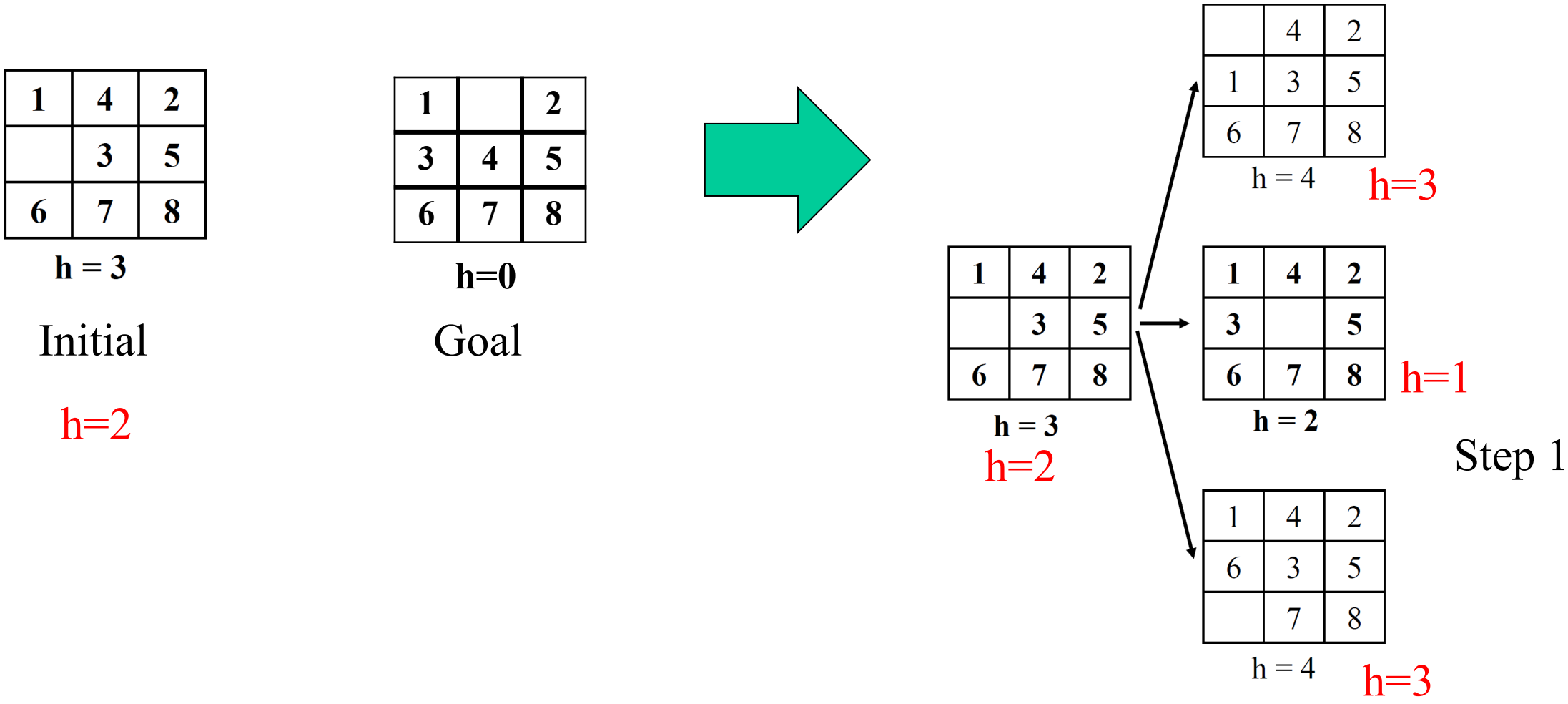
In this case, only the "**3**", "**8**" and "**1**" tiles are misplaced, by 2, 3, and 3 squares respectively, so the heuristic function evaluates to 8.

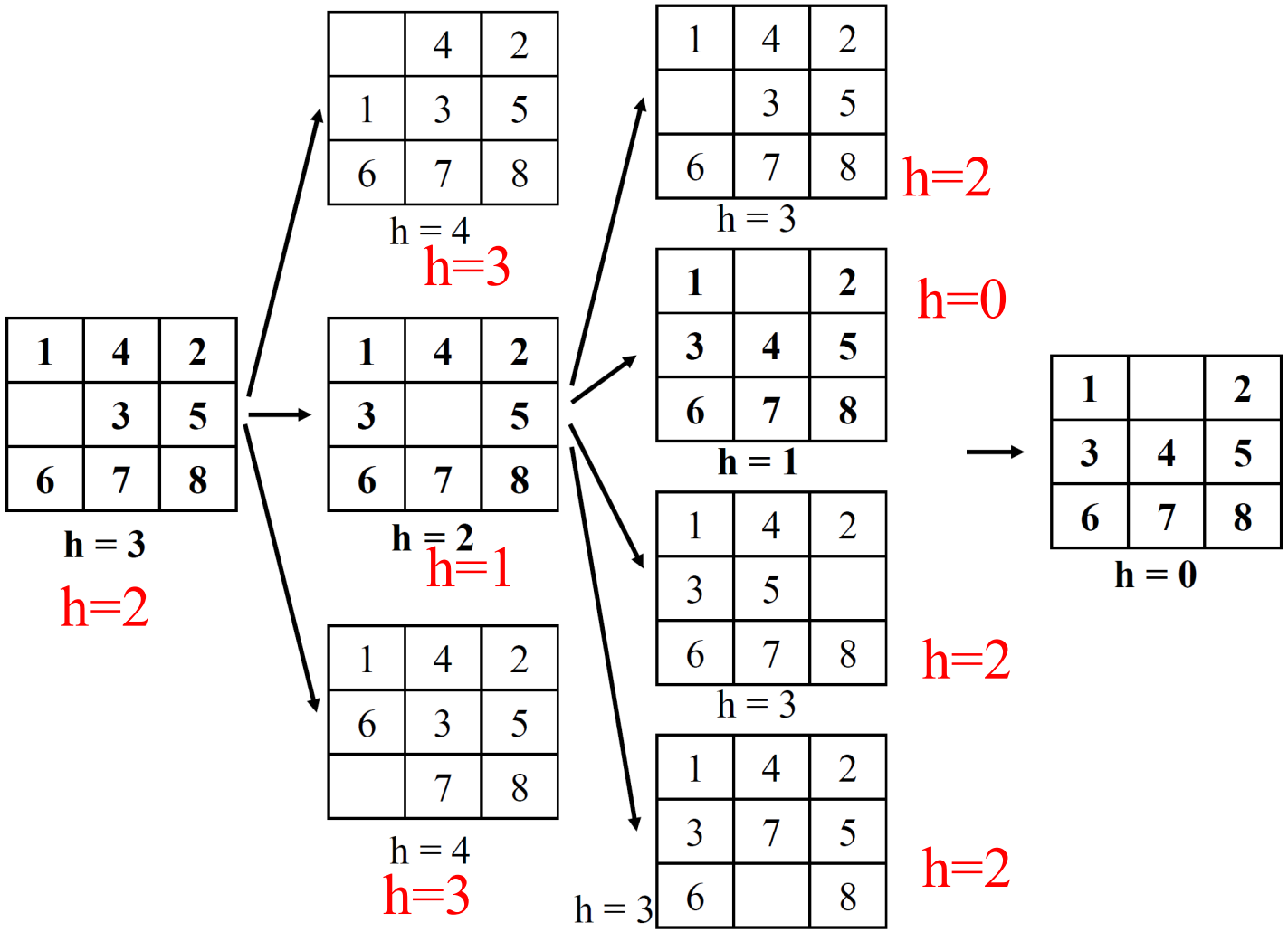
In other words, the heuristic is *telling* us, that it *thinks* a solution is available in just 8 more moves.

Notation: $h(n)$ h(current state) = 8

- Heuristic function is Manhattan Distance

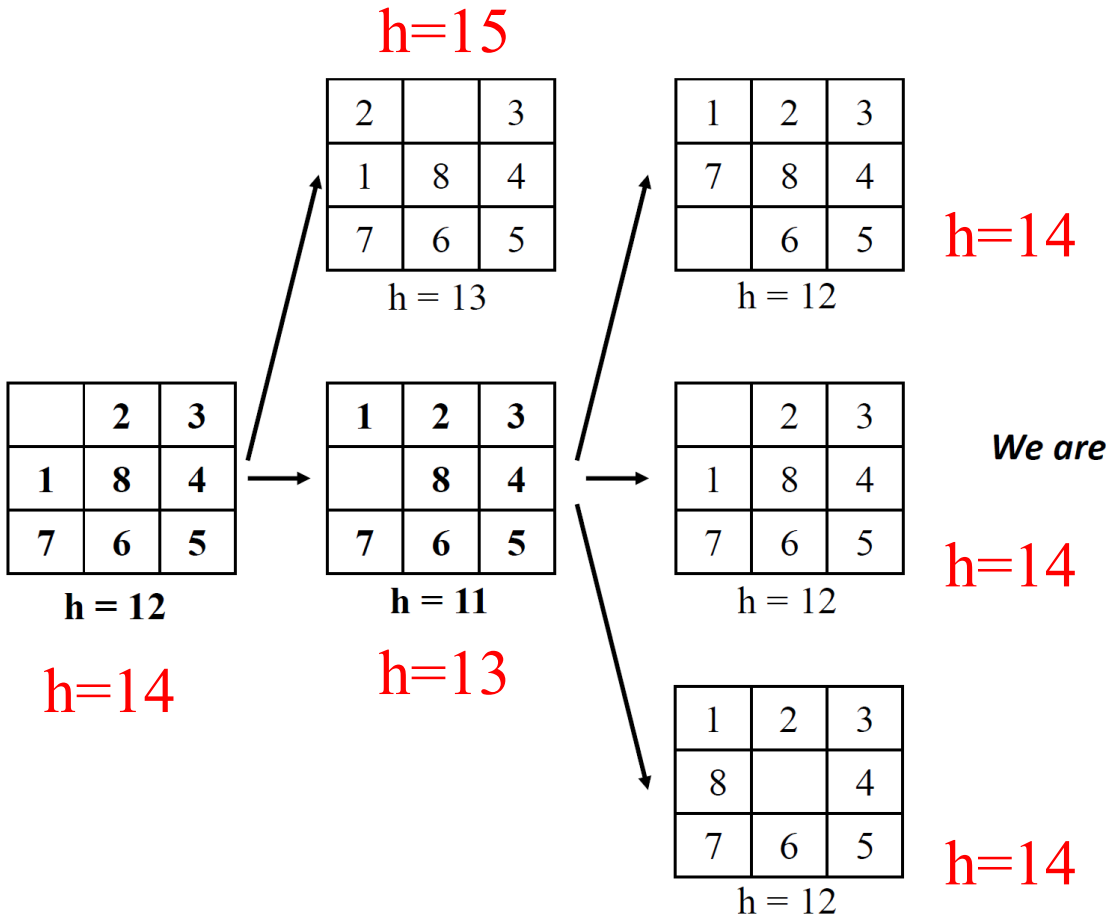
Solution to 8 puzzle problem





- A solution case.

No Admissibility example



Heuristic function is Manhattan Distance

We are stuck with a local maximum.

	8	7
6	5	4
1	2	3

- Reference goal state, h=0

Drawbacks of Hill-Climbing

- Problems:
  - **Local Maxima**:
  - **Plateaus:** the space has a broad flat plateau with a singularity as it's maximum
  - **Ridges:** steps to the North, East, South and West may go down, but a step to the NW may go up.
- Remedy:
  - Random Restart.
  - Multiple HC searches from different start states
- Some problems spaces are great for Hill Climbing and others horrible.

Simulated Annealing

- A **hill-climbing algorithm** that never makes "downhill" moves toward states with lower value (or higher cost) is guaranteed to be **incomplete**, because ***it can get stuck on a local maximum***.
  - In contrast, a **purely random walk**—that is, moving to a successor chosen uniformly at random from the set of successors—is **complete but extremely inefficient**.
  - Therefore, it seems reasonable to ***combine hill climbing with a random walk in some way that yields both efficiency and completeness***.
- **Idea:** ***escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency.***
- The **simulated annealing algorithm**, a version of ***stochastic hill climbing*** where some downhill moves are allowed.
- **Annealing:** the process of gradually cooling metal to allow it to form stronger crystalline structures
- **Simulated annealing algorithm:** gradually "cool" search algorithm from Random Walk to First-Choice Hill Climbing

Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"

   current ← MAKE-NODE(problem.INITIAL-STATE)
   for t = 1 to ∞ do
       T ← schedule(t)
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← next.VALUE – current.VALUE
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```

- **Downhill moves** are accepted readily early in the annealing schedule and then less often as time goes on.
- The **schedule** input determines the value of the temperature T as a function of time.

Simulated Annealing (Time Gradiant aware)

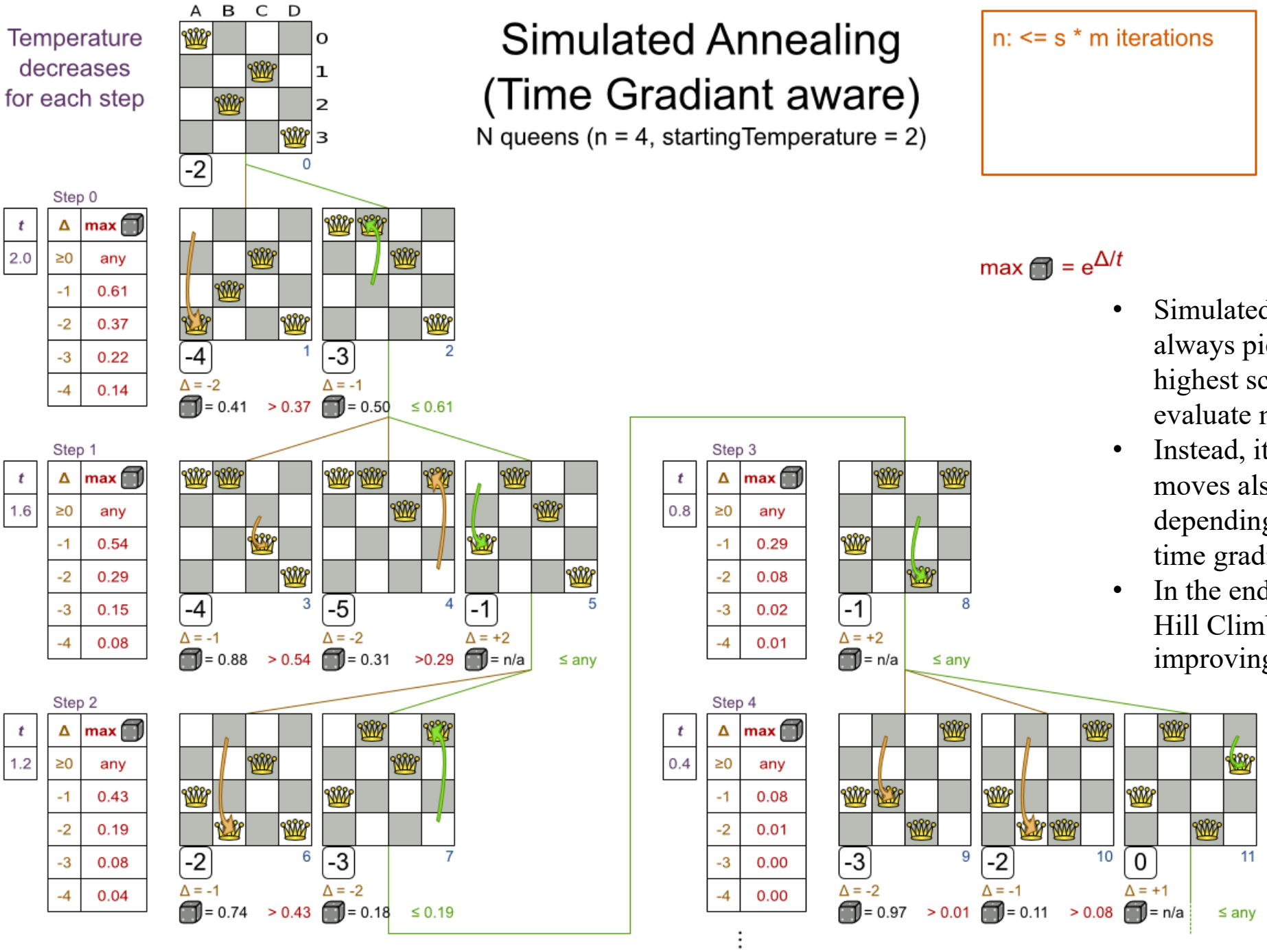
N queens (n = 4, startingTemperature = 2)

Temperature decreases for each step

n: <= s * m iterations

$\max \square = e^{\Delta/t}$

- Simulated Annealing does not always pick the move with the highest score, neither does it evaluate many moves per step.
- Instead, it gives non-improving moves also a chance to be picked, depending on its score and the time gradient of the Termination.
- In the end, it gradually turns into Hill Climbing, only accepting improving moves.



Informed Search Summary

- **Best-first search** is a general search strategy where the minimum cost open node (according to some measure) is selected for expansion at each step.
- **Greedy search** uses minimal estimated cost to the goal state, h(n), as measure. This reduces the search time, but the algorithm is neither complete nor optimal.
- **A* search** combines uniform-cost search and greedy search: f(n) = g(n) + h(n) and handles state repetitions and h(n) never overestimates.
  - A* is complete, optimal and optimally efficient (i.e., no other optimal algorithm expands fewer nodes), but its space complexity is still bad.
  - The time complexity depends on the quality of the heuristic function.
  - IDA* reduces the memory requirements of A*.
- **Hill-climbing algorithms** keep only a single state in memory, but can get stuck on local optima.
- **Simulated annealing** escapes local optima, and is complete and optimal given a slow enough cooling schedule (in probability).

Exercise 1

- SOET, Adamas University is preparing a trip for 400 students. The school has 10 buses of 50 seats each and 8 buses of 40 seats but only has 9 drivers available. The operating cost for the bigger bus is 2000 INR and for the smaller bus is 1500 INR. Calculate how many buses of each type should be used for the trip for the cheapest possible cost.

Problem Formulation

- Assume we need x larger buses and y smaller buses.
- Our heuristic function is

$$Min\ f(x, y) = 2000x + 1500y$$

Constraints

$$50x + 40y >= 400$$

$$x + y <= 9$$

$$1 <= x <= 10$$

$$1 <= y <= 8$$

Exercise 2

- A transport company has two types of trucks, Type A and Type B. Type A has a refrigerated capacity of 20m3 and a non-refrigerated capacity of 40m3. In contrast, Type B has the same overall volume with equal refrigerated and non-refrigerated stock sections. A grocer must hire trucks to transport 3000m3 of refrigerated stock and 4000m3 of non-refrigerated stock. The cost per kilometer of Type A is 30, and 40 for Type B. How many trucks of each type should the grocer rent to achieve the minimum total cost?