ME 620: Fundamentals of Artificial Intelligence

Lecture 7: Informed Search Strategies - I

Shyamanta M Hazarika
Biomimetic Robotics and Artificial Intelligence Lab
Mechanical Engineering and M F School of Data Sc. & AI
IIT Guwahati

Informed Search Strategies

- Heuristic Search
- Types of Heuristic Search
  - Best First Search
    - Algorithm A*
  - Local search
    - Hill Climbing
    - Simulated Annealing

Informed Search

- Uninformed search methods, whether breadth-first or depth-first. are exhaustive methods for finding paths to a goal node.
  - These methods provide a solution; but often are infeasible to use because search expands too many nodes before a path is found.
- Informed search methods use task-dependent information to help reduce the search.
  - Task-dependent information is called heuristic information; and search procedures using it are called heuristic search methods.

Informed Search

- Interested in minimizing some combination of the cost of the path and the cost of search required to obtain the path.
  - Explore search methods that minimize this combination averaged over all problems likely to be encountered.
- Heuristic information is used so that search expands along those sectors of the frontier thought to be most promising.

Informed Search

- Also called heuristic search
- In a heuristic search each state is assigned a heuristic value that the search uses in selecting the best next step.
- A heuristic is an operationally-effective nugget of information on how to direct search in a problem space.

Heuristics

- Heuristics
  - (Greek *heuriskein* = find, discover): "the study of the methods and rules of discovery and invention".
- Use our knowledge of the problem to consider some (not all) successors of the current state (preferably just one, as with an oracle).
- This means pruning the state space, gaining speed, but perhaps missing the solution!

Heuristic Function

- For heuristic search to work, we must be able to rank the children of a node.
- A heuristic function takes a state and returns a numeric value -- a composite assessment of this state.
  - We then choose a child with the best score (this could be a maximum or minimum).
  - A heuristic function can help gain or lose a lot, but finding the right function is not always easy.

Heuristics and AI Search

- The principal gain - often spectacular - is the reduction of the state space.
  - For example, the full tree for Tic-Tac-Toe has 9! leaves. If we consider symmetries, the tree becomes six times smaller, but it is still quite large.
  - With a fairly simple heuristic function we can get the tree down to 40 states.
- Heuristics can also help speed up exhaustive, blind search, such as depth-first and breadth-first search.

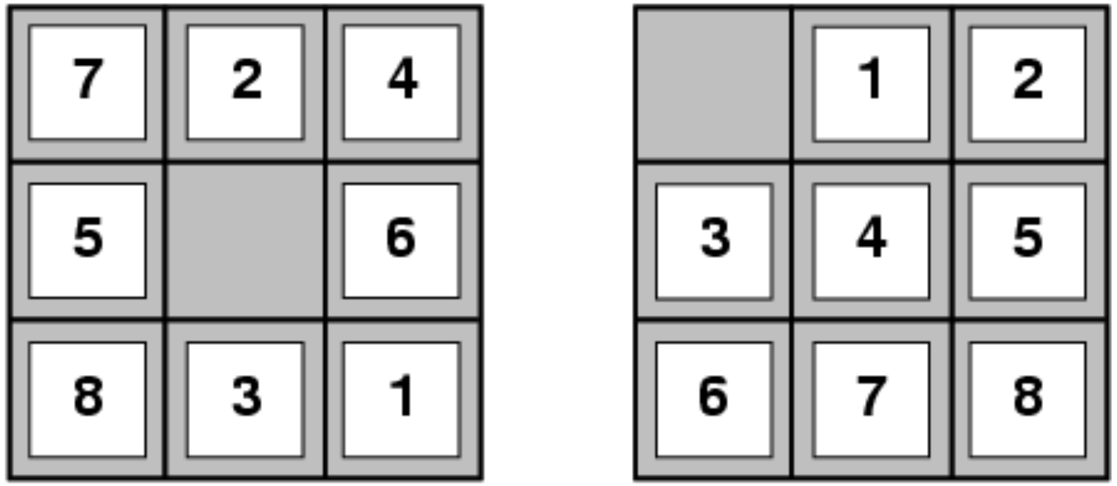
Problem Relaxation

A standard approach of creating heuristics is problem relaxation.

We relax a problem by adding some new actions such that search is not required to find the solution cost in the relaxed problem.

Heuristics created as solutions to relaxed problems are usually admissible because adding new actions should only reduce the solution cost and not increase it.

Problem Relaxation



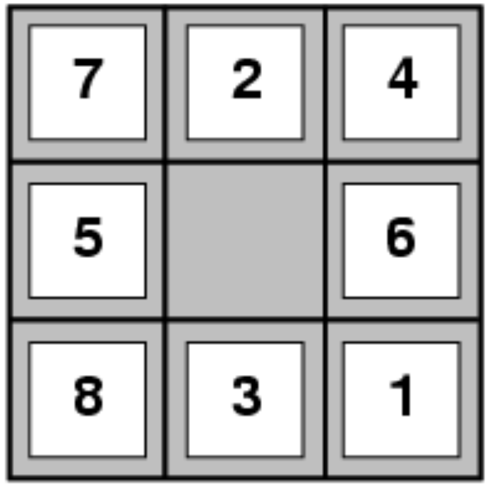
Start State Goal State

Here is an instance of the 8 puzzle problem. We are given the state on the left side and we want to transform it into the state on the right. The only action we have is sliding a tile up, down, left or right into the empty space. All actions are equivalent so let’s assign each action the same cost 1. The cost of a solution is then the number of actions taken and we want to find the cheapest solution.

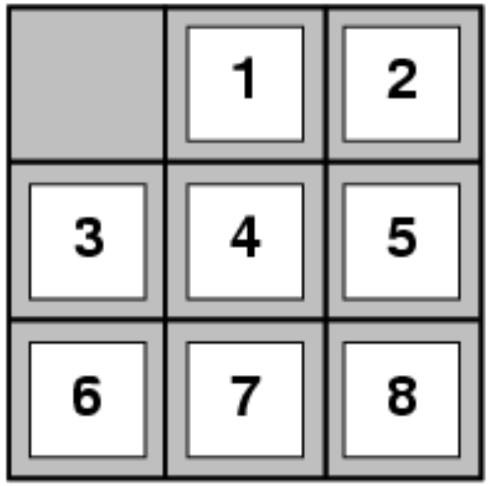
Relaxed Problem

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Number of Misplaced Tiles



Start State

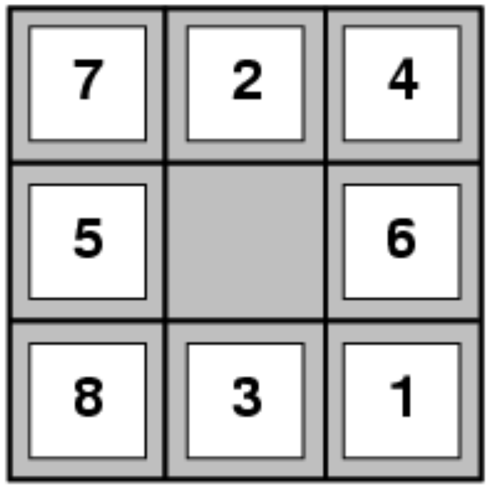


Goal State

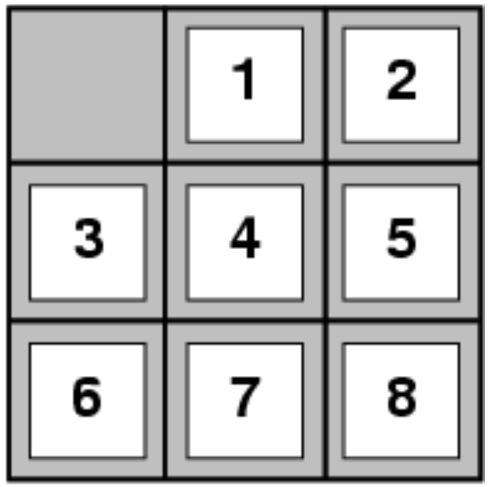
Suppose we could take out a tile and place it in its correct position in 1 move. Given any state, how many moves would it take to reach the goal state? It's precisely the number of misplaced tiles, also called the Hamming distance. In this relaxed version of 8-puzzle, we don't need to use search to find out the solution cost. We can calculate it directly so it can be used as a heuristic.

$h_1(n)$ = number of misplaced tiles

Total Manhattan Distance



Start State



Goal State

Suppose we could slide a tile into any block, as opposed to only the empty block, in 1 move (Imagine a 3D board, where each tile is in its own 2D plane). Given any state, how many moves would it take to reach the goal state? It'd be the sum of horizontal and vertical distances of each tile from its desired position, also called the Manhattan distance. We can calculate this value directly, without requiring search.

$h_2(n)$ = total Manhattan distance

Heuristic Function

- Heuristic function, h(n): estimates the cost to a goal; most promising states are selected.
- For ordering nodes for expansion, we need a method to compute the promise of a node. This is done using a real-valued evaluation function.
  - Evaluation functions have been based on a variety of ideas.

Best-first search

- Is a way of combining the advantages of both depth-first and breadth-first search into a single method.
  - Depth-first is good as it allows a solution to be found without all competing branches having to be expanded.
  - Breadth-first is good because it does not get trapped on dead-end paths.
  - Combine by following a single path at a time, but switch paths whenever some competing path looks more promising than the current one.

Best-first search

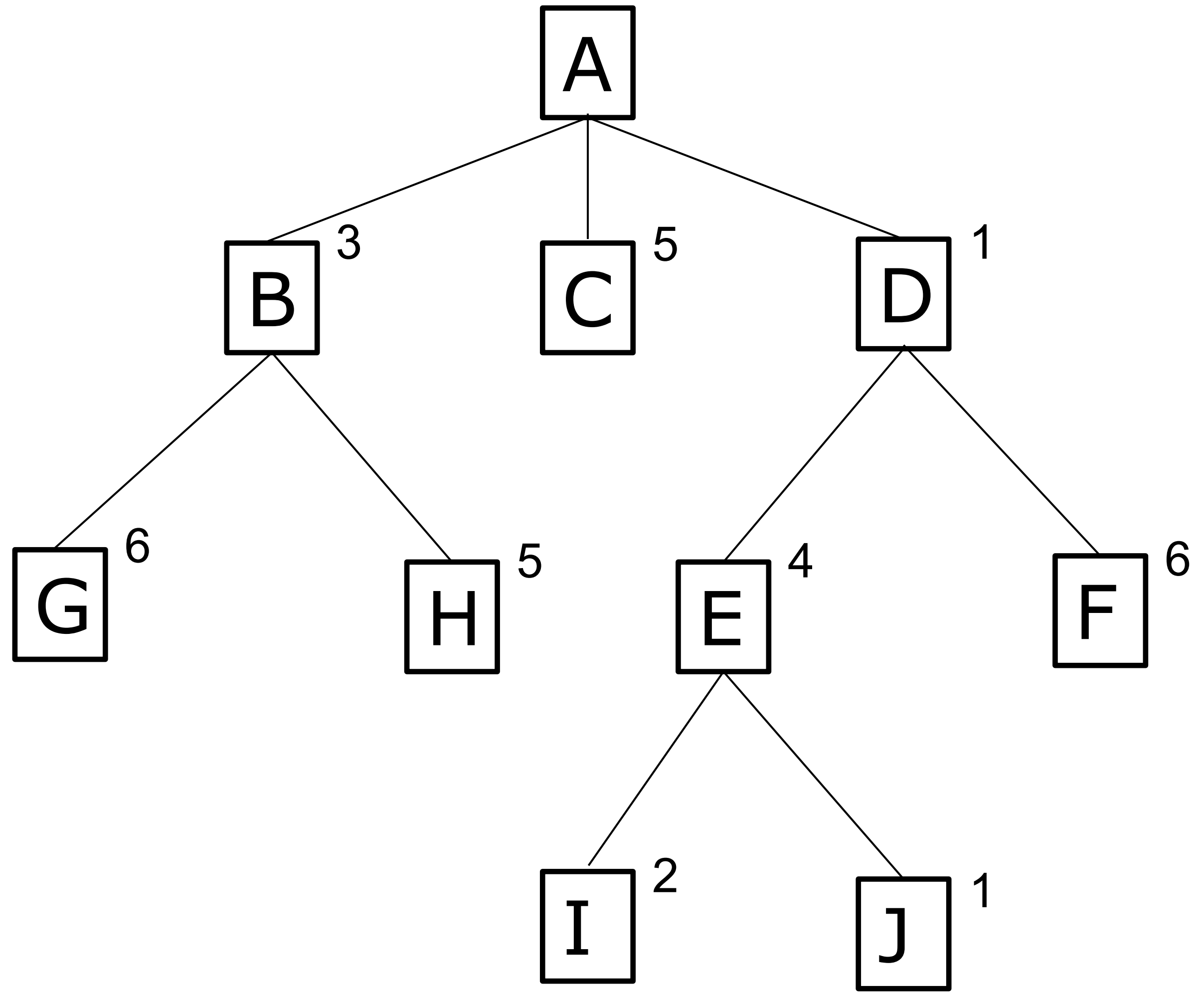
- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability”
  - Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

- Special case:
  - A^* search

Best-first search



Graph Search

Procedure **GRAPHSEARCH**

```
    Create a search graph G consisting solely of the start node s.
    Put s on a list called OPEN
    Create a list called CLOSED that is initially empty.
    Loop: If OPEN is empty, exit with failure
    Select first node on OPEN, remove it from OPEN and put it on closed. Call this node n.
    If n is goal node, exit successfully with the solution obtained by tracing a path along the
    pointers from n to s in G
    Expand node n generating the set M of its successors and install them as successors of n in G.
    Establish a pointer to n from those members of M that were not already in G. Add these members of M to OPEN.
        For each member of M already in G decide whether or not to redirect its pointer to n.
        For each member of M already on CLOSED, decide for each of its descendants in G
        whether or not to redirect its pointer.
    Reorder the list OPEN, according to heuristic merit.
    Go LOOP.
```

Algorithm A

- Idea: avoid expanding paths that are already expensive
- Evaluation Function f (n) can be defined so that its value at any node n estimates the sum of cost of minimal cost path from start node s to node n together with the cost of minimal cost path from node n to goal node.
  - f(n) is an estimate of the cost of a minimal cost path constrained to go through node n.

Algorithm A

- Let function $k(n_i, n_j)$ give the actual cost of a minimal cost path between two arbitrary nodes n_i and n_j.
  - Cost of minimal cost path from node n to some particular goal node t_i is then given by $k(n,t_i)$.
  - $h^*(n)$ be the minimum of all the $k(n,t_i)$ over the entire set of goal nodes $\{t_i\}$
- $h^*(n)$ is the cost of the minimal cost path from n to a goal node.

Algorithm A

- Cost of an optimal path from a given start node, s to some arbitrary node n is $k(s, n)$.
  - $g^*(n) = k(s, n)$; for all n accessible from s.
- Define function f^* so that its value $f^*(n)$ at any node n is the actual cost of an optimal path from node s to node n plus the cost of an optimal path from node n to a goal node.

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

Algorithm A

- Evaluation function f is an estimate of f^*. This estimate can be given by

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

  - $g(n)$ = cost so far to reach n; is an estimate of g^*
  - $h(n)$ = estimated cost from n to goal is an estimate of h^*; for which we rely on heuristic information.
  - $f(n)$ = estimated total cost of path through n to goal

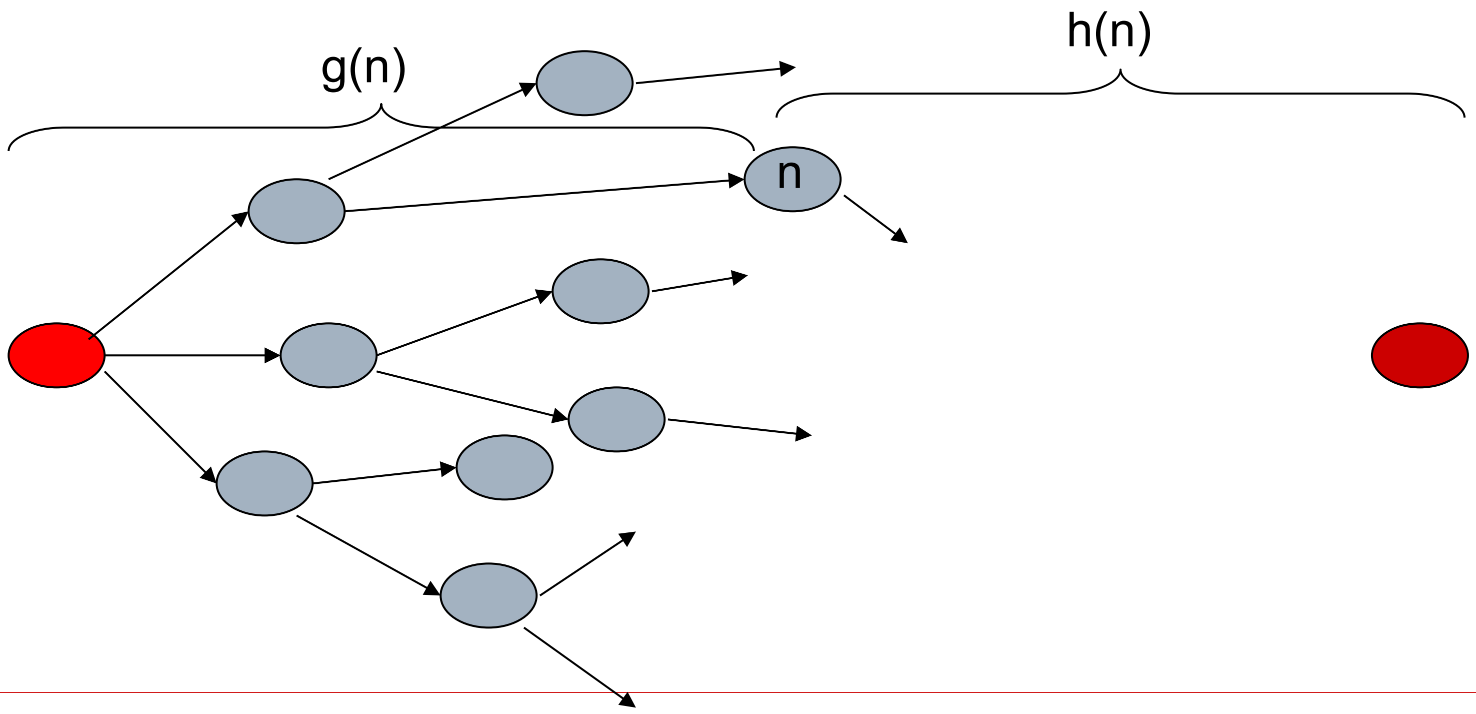
- Graph search algorithm using this evaluation function for ordering nodes in called algorithm A.

A* Algorithm

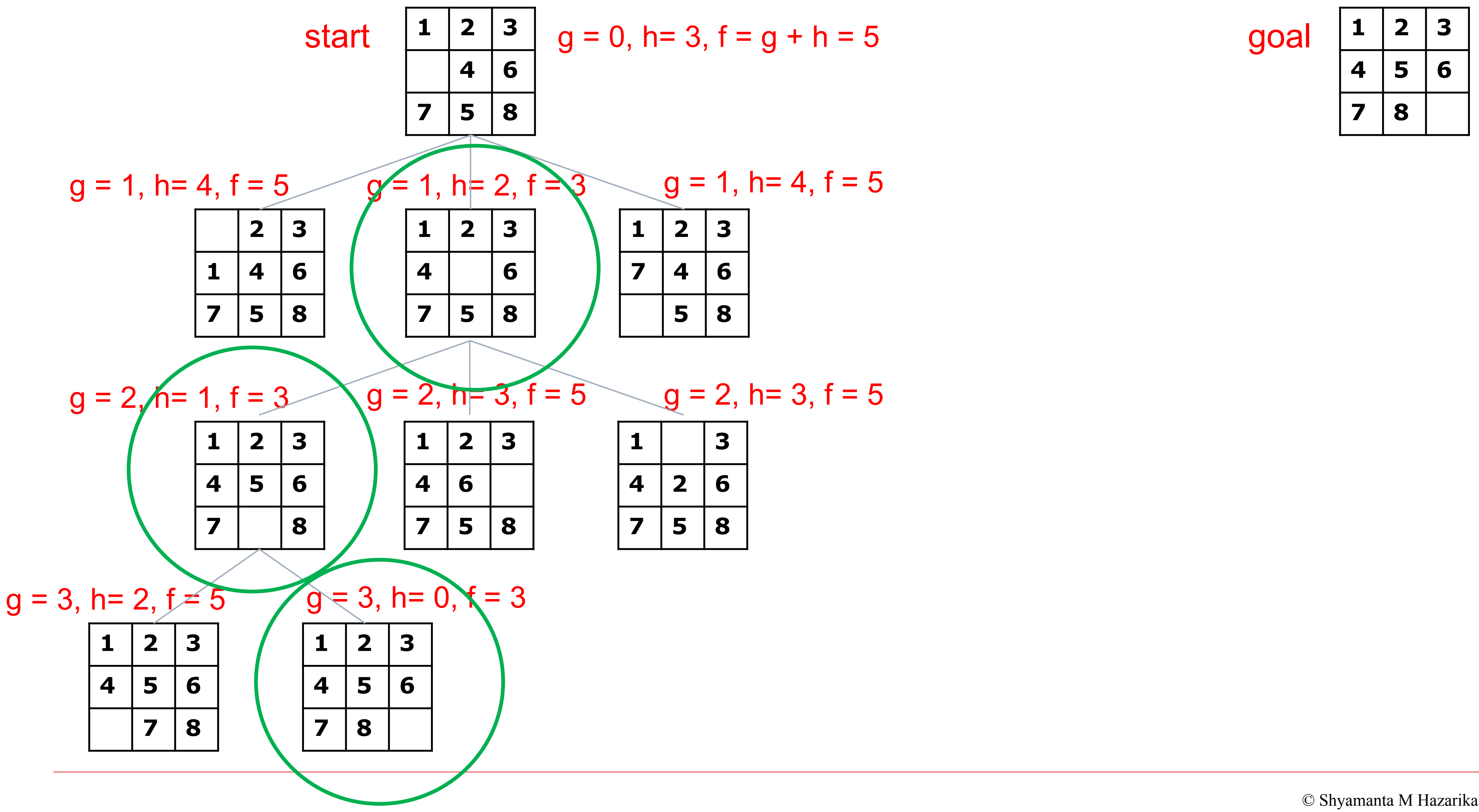
- If h is a lower bound on $h*$ i.e., if $h(n) \leq h^*(n)$ for all nodes n, then algorithm A will find an optimal path to a goal.
- When algorithm A uses a h function that is a lower bound on h^*, we call it *algorithm* A^*.
  - The best-first algorithm that was described is a simplification of algorithm A^*.
  - Since $h = 0$ is certainly a lower bound on h^*; BFS finding minimal length paths follows as a special case.

A* Algorithm

- $f(n) = g(n) + h(n)$
  - $g(n)$ = cost from the starting node to reach n
  - $h(n)$ = estimate of the cost of the cheapest path from n to the goal node



Solving 8-Puzzle using A*



Observations

- f(n) lets us choose which node to expand next on the basis not only of how good the node itself looks (as measured by h), but also on the basis of how good the path to the node was!
- The estimator of h*, distance of a node to the goal; if h is a perfect estimator of h*, A* will converge immediately to the goal with no search.
  - Better the estimate h, closer we would be to the direct approach.
  - On the other hand, if h is zero, the search would be controlled by g; g = 0: random and g = 1: BFS.