AI 2002 Artificial Intelligence

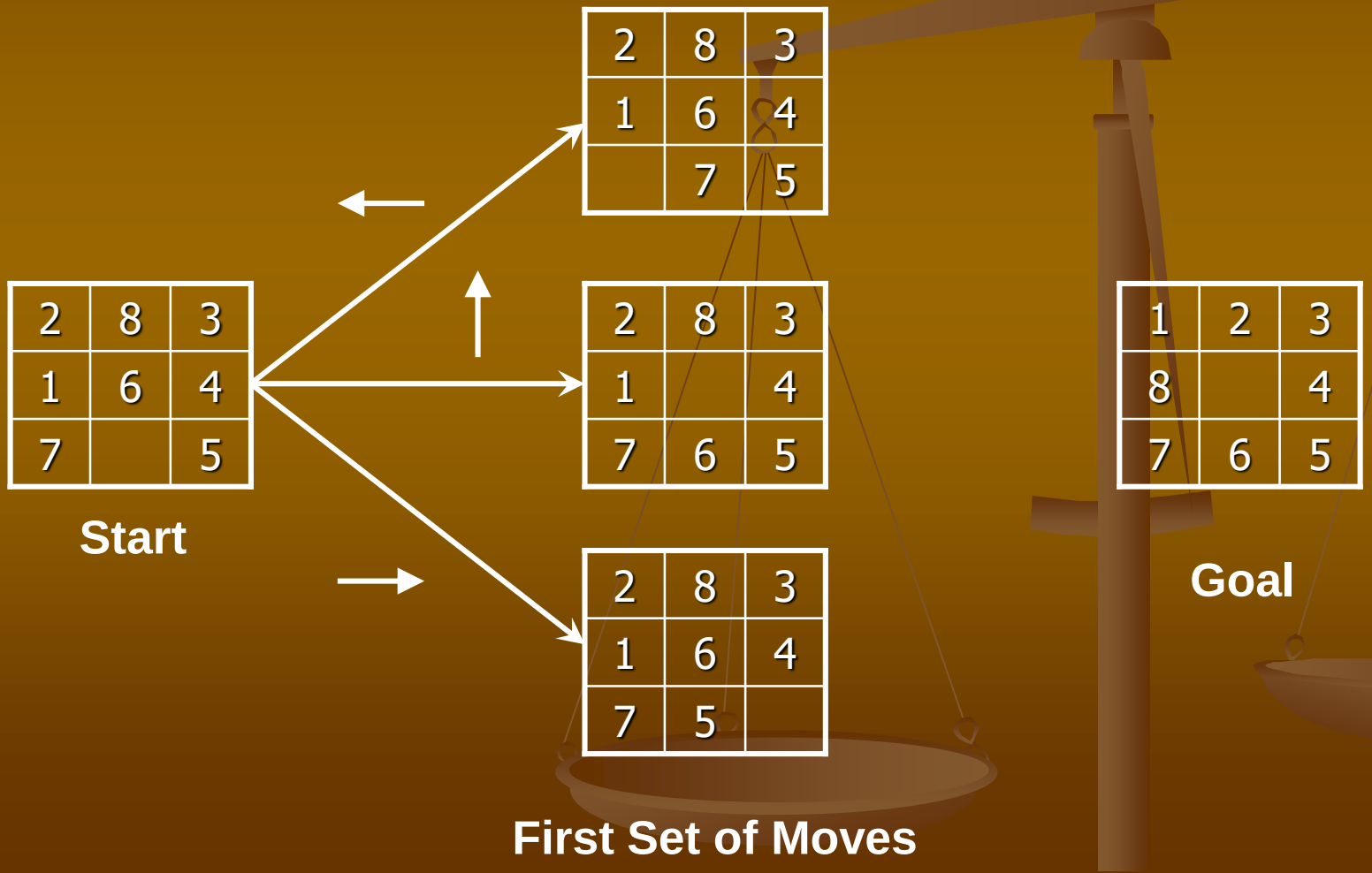
FAST-NU
Heuristic Search Algorithms II

Lecture 5

Department of Computer Science
National University of Computers & Emerging Sciences
Lahore.

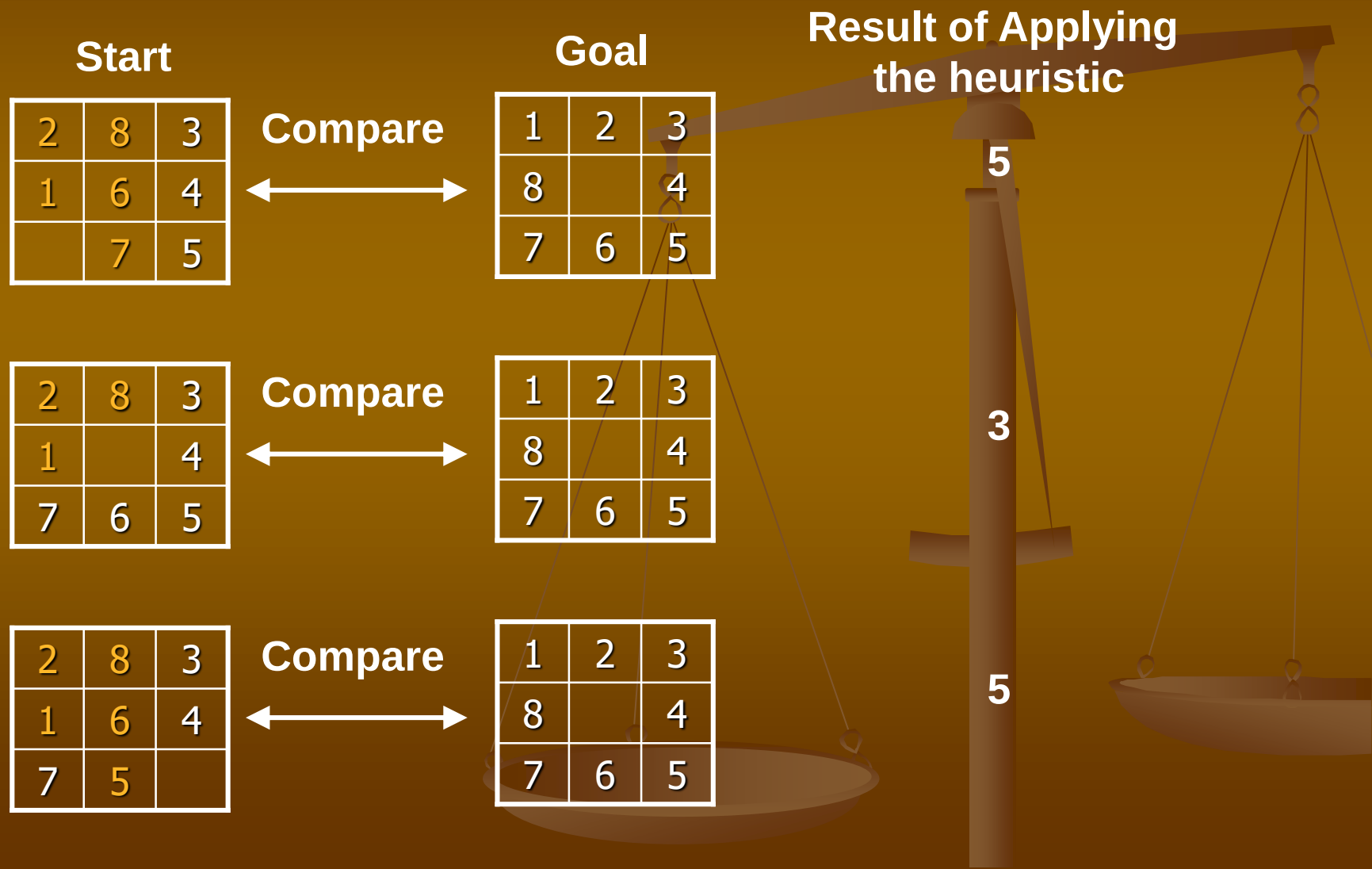
8-Puzzle with Heuristic Search

- Lets look at the performance of several different heuristics for solving the 8-puzzle.
- The figure 4.6 shows a start and goal state for the 8-puzzle, along with the first three states generated in the search.



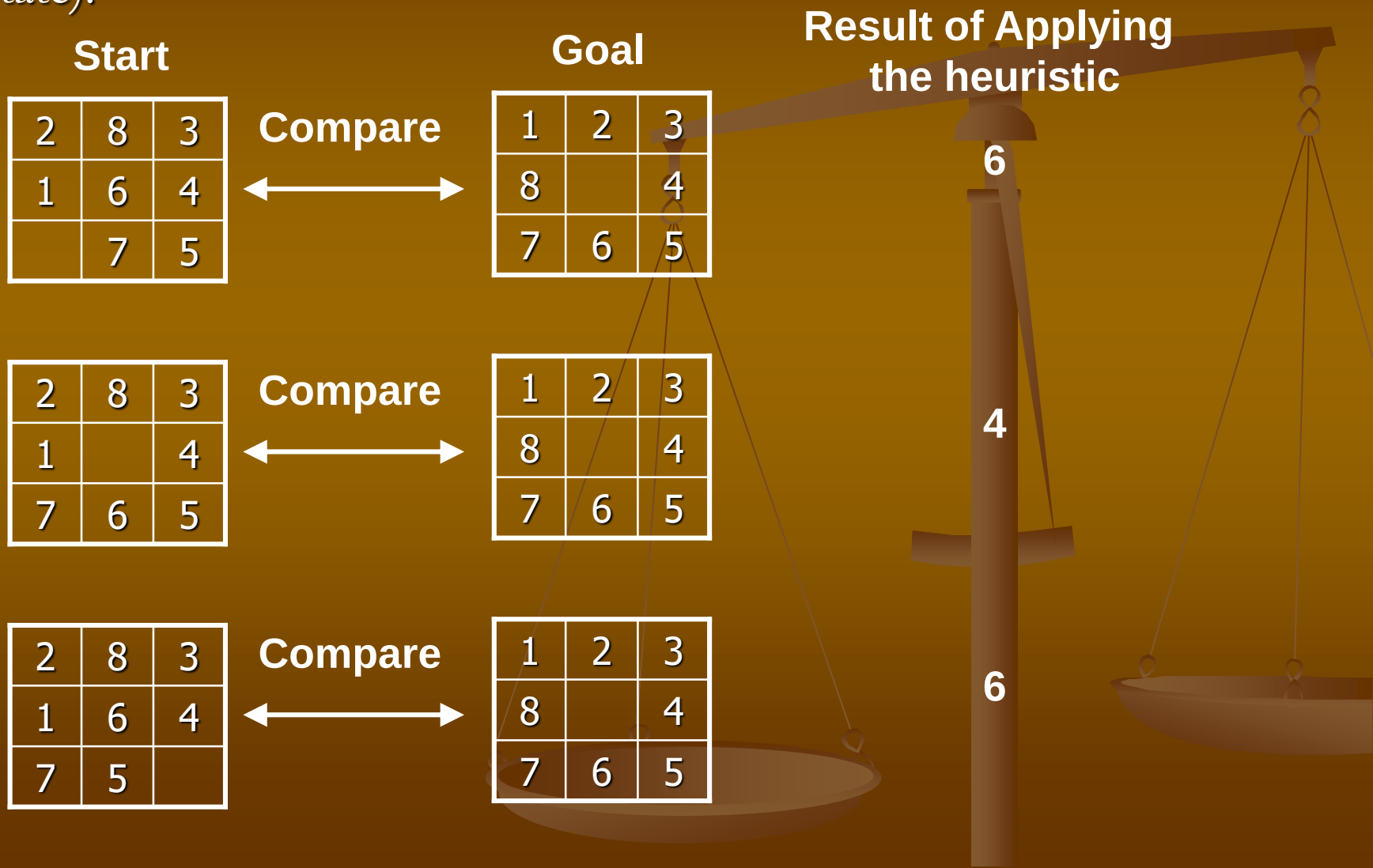
8-Puzzle with Heuristic Search

- Heuristic #1: Count the tiles out of place in each state when it is compared with the goal.



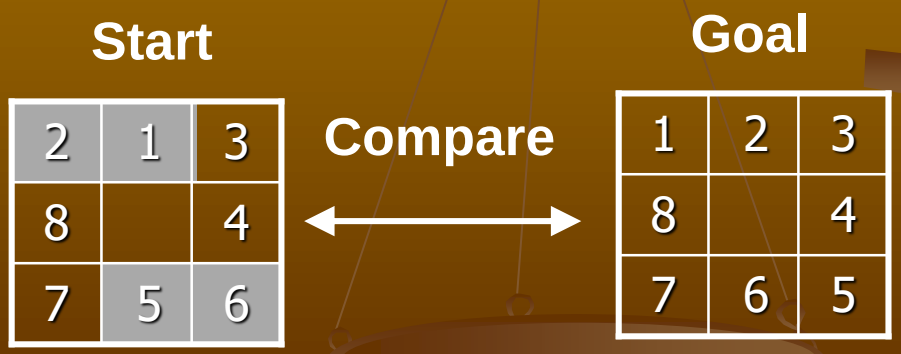
8-Puzzle with Heuristic Search

- Heuristic #2: Sum all the distances by which the tiles are out of place (one for each square, a tile be moved to reach its position in the goal state).



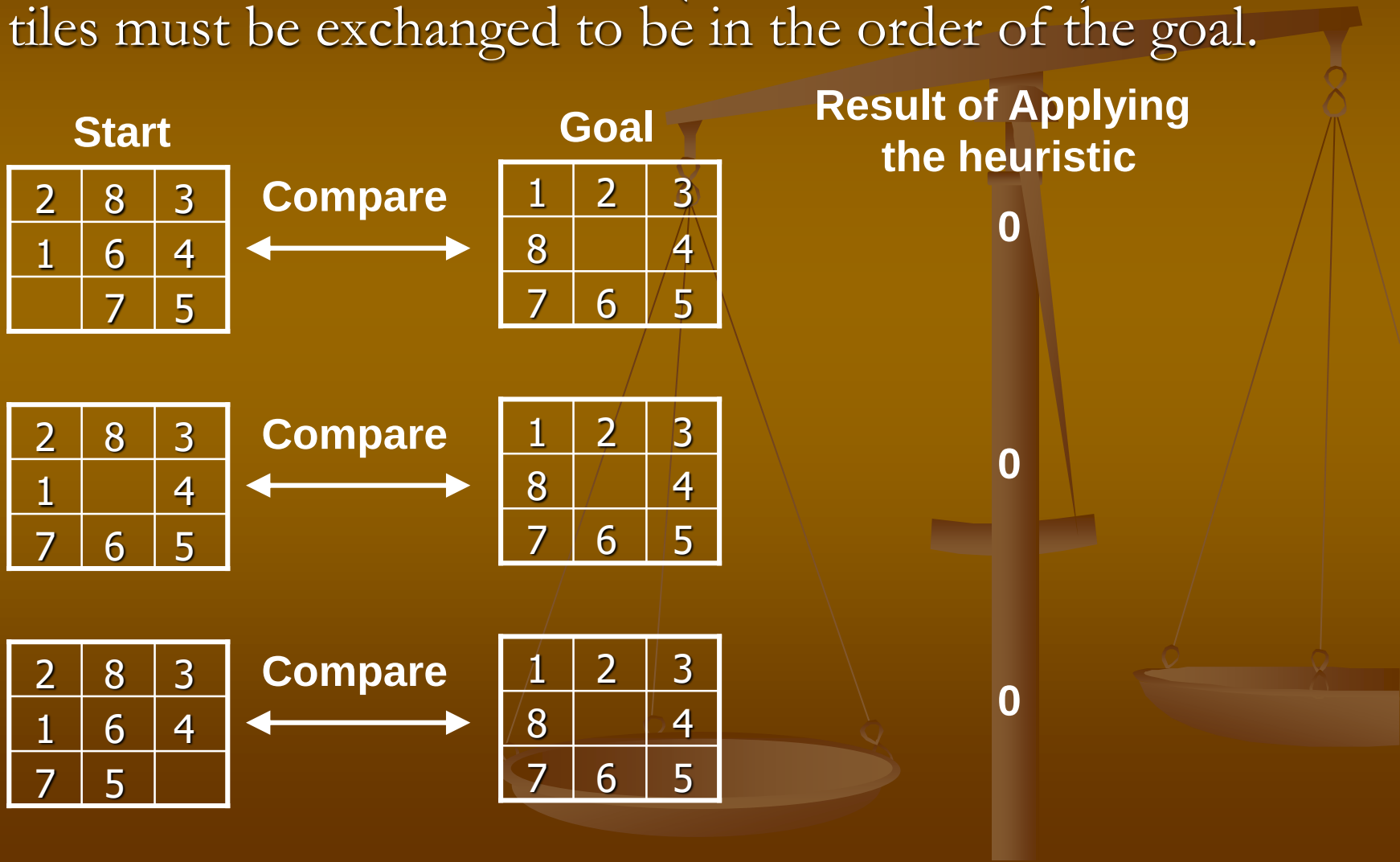
8-Puzzle with Heuristic Search

- Heuristic #3: Both above heuristics can be criticized for failing to acknowledge the difficulty of tile reversals.
- Tile reversals means if two tiles are next to each other and the goal requires their being in opposite locations.
- It takes (many) more than two moves to put them back in place, as the tile must "go around" each other
- The following figure shows an 8-puzzle with a goal and two tile reversals, i.e. 1 and 2; 5 and 6.



8-Puzzle with Heuristic Search

- Heuristic #3 (continued): A heuristic that takes into account could be to multiply a small number (say 2) with each direct tile reversal (i.e. where two adjacent tiles must be exchanged to be in the order of the goal.



8-Puzzle with Heuristic Search

- In the example being considered, the "sum of distances" heuristic does indeed seem to provide a more accurate estimate of the work to be done than the simple "count of the number of tiles out of place".
- Also note that the tile reversal heuristic fails to distinguish between these states, giving each an evaluation of 0. Although it is an *intuitively appealing* heuristic, it breaks down since none of these states have any direct reversals.
- A fourth heuristic, which may overcome the limitation of the tile reversal heuristic could be: add the sum of distances out of place and 2 times the number of direct tile reversals.
- It must be noted that each of the above heuristic ignores some critical bit of information and is subject to improvements.

Algorithm A - Admissibility

- Consider the evaluation function f(n) = g(n) + h(n), where
  - n is any state encountered in the search.
  - g(n) is the cost of n from the start state.
  - h(n) is the heuristic estimate of the cost of going from n to goal.

  If this evaluation function is used with the best-first search algorithm, the result is called *algorithm A*.

- A search algorithm is *admissible* if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists.

Algorithm A for 8-Puzzle

- Lets solve 8-puzzle now with h(n) being the number of tiles out of place. The evaluation function f(n) = g(n) + h(n) thus in this case would be composed of

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

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

- The full best-first search of the 8-puzzle graph, using f as defined above, appears in Figure 4.10 (next slide).

Algorithm A for 8-Puzzle

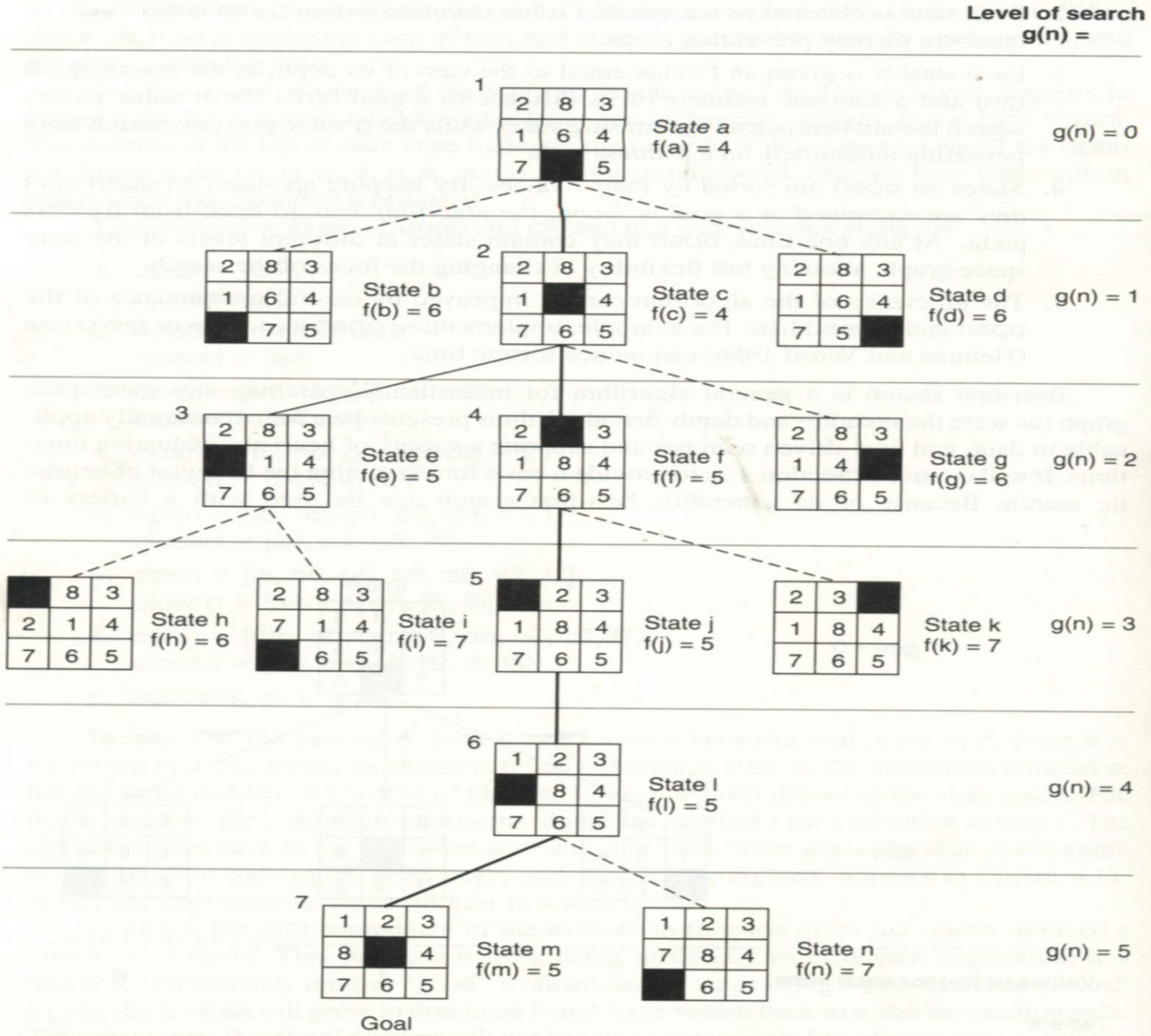


Figure 4.10 State space generated in heuristic search of the 8-puzzle graph.

Algorithm A for 8-Puzzle

- In step 3 of the execution, both e and f have a heuristic evaluation of 5. State e is examined first, producing its children, h and i.
- Although state h, the immediate child of e, has the same number of tiles out of place as f, it is one level deeper in the state space.
- The depth measure, g(n), therefore causes the algorithm to select f for evaluation in step 4.
- The algorithm goes back to the shallower state and continues to the goal.
- The g(n) component of the evaluation function gives the search more of a breadth-first flavor. This prevents it from being misled by an erroneous evaluation: if a heuristic continuously returns "good" evaluations for states along a path that fails to reach a goal, the g(n) value will grow to dominate h(n) and force search back to a shorter solution path.

Algorithm A*

- Consider an evaluation function:

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

- Now consider another evaluation function:

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

  where g*(n) is the cost of the *shortest path* from the start node n and h* returns the *actual cost* of the shortest path from n to the goal. Thus f*(n) is the actual cost of the optimal path from a start node to a goal node that passes through node n.

- If we apply best-first search with the evaluation function f*, the resulting search is admissible.
  - *If algorithm A is used with an evaluation function in which h(n) is less than or equal to the cost of the minimal path from n to the goal, the resulting search algorithm is called algorithm A*.*
  - *Which means that all A* algorithms are admissible.*
- This leads us to the theorem that any A* algorithm (i.e. one that uses a heuristic h(n) such that h(n) is less or equal to h*(n) for all n) is guaranteed to find the minimal path from n to the goal, if such a path exists.

Algorithm A* and 8-Puzzle

- Several heuristics from the 8-puzzle provide examples of A* algorithms.
- For instance, the heuristic of counting the number of tiles not in the goal position, is certainly less than or equal to the number of moves required to move them in their goal position. Thus this heuristic is admissible and guarantees to fall into the optimal (shortest) solution path.
- The sum of the direct distances of titles out of place is also less than or equal to the minimal actual path.
- Even using small multipliers for direct tile reversals gives an admissible heuristic.

- Even though the actual cost of the shortest path to a goal may not always be computed, we can often prove that a heuristic is bounded from above by this value. When this can be done, the resulting search will terminate in the discovery of the shortest path when such a path exists.

A* Search Algorithm

- A* is a special case of the best-first search algorithm that guarantees to find the shortest route, or least cost route from start to goal.

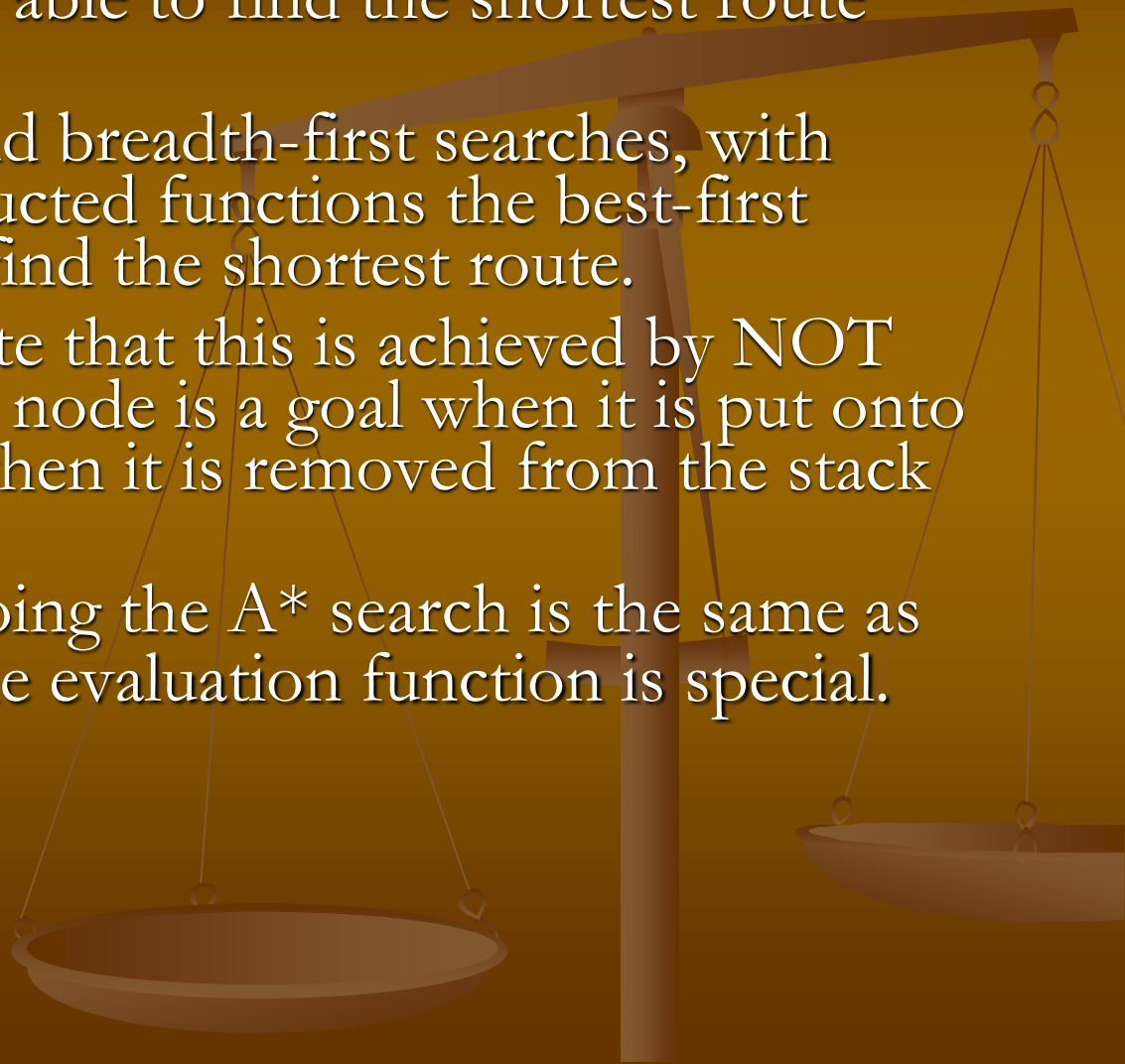
  Let us consider the following evaluation function:

  $$e(n) = c(n) + d'(n)$$

- The function e(n) is the evaluation value at node n and is composed of two parts
  - c(n) is the cost function that evaluates the cost of getting to node n which may include the depth down the tree and accumulated extra costs from each node visited. If we are looking for the short route from city A to city G then the cost will include distance traveled between the intermediate cities.
  - The function d'(n) is the estimated minimum distance of the node from a goal.
  - A distance estimation function d'(n) is said to be admissible if d'(n) <= d(n) , d'(n) >= 0 and d(n) >= 0 for all n, where d(n) is the actual shortest distance of node n from a goal.
  - In other words, the distance estimation must always be either exactly correct or an under-estimate.

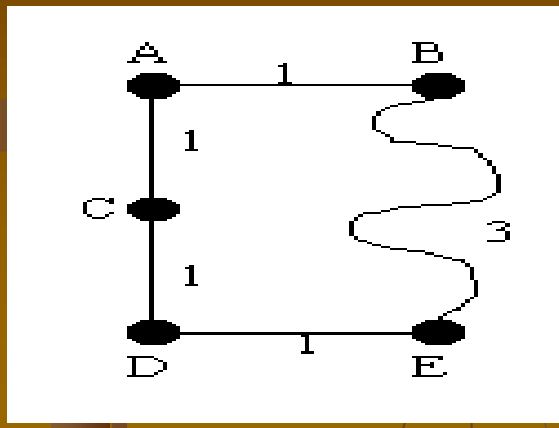
A* Search Algorithm (continued ….)

- With an appropriate costing function and estimation function we are now able to find the shortest route between nodes.
- Unlike depth-first and breadth-first searches, with appropriately constructed functions the best-first solution will always find the shortest route.
- It is important to note that this is achieved by NOT evaluating whether a node is a goal when it is put onto the stack, but only when it is removed from the stack and expanded.
- The algorithm for doing the A* search is the same as for best-first, only the evaluation function is special.

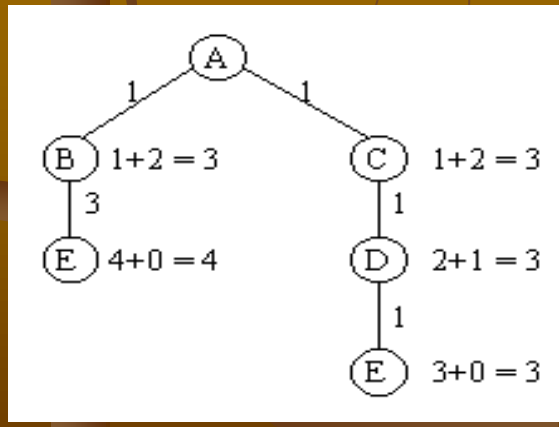


A* Search – An Example

- This example demonstrates A* and the principles of best-first as a whole.
- The first diagram shows a simple map with distances between cities, the search needs to find the shortest distance between A and E. The second diagram shows the search tree used by the algorithm. This is how it runs the search:



1. For both B and C the cost of going from A is evaluated and added to the estimated distance of going from here to E. The estimate is the simple horizontal, vertical step measurement. The actual cost and estimate of future cost are added together and this provides the overall evaluation value. Both B and C produce value 3.
2. Let us assume that for equal values, the search always takes the leftmost value, so we expand node B. We eliminate backtracking so we are left with node E, we put this onto the list with value 4 - we have traveled four units in total and the estimate says there are no units to travel from E to E.

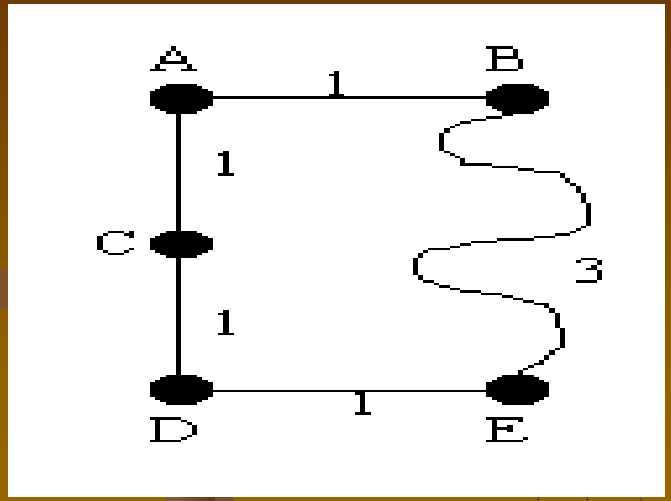


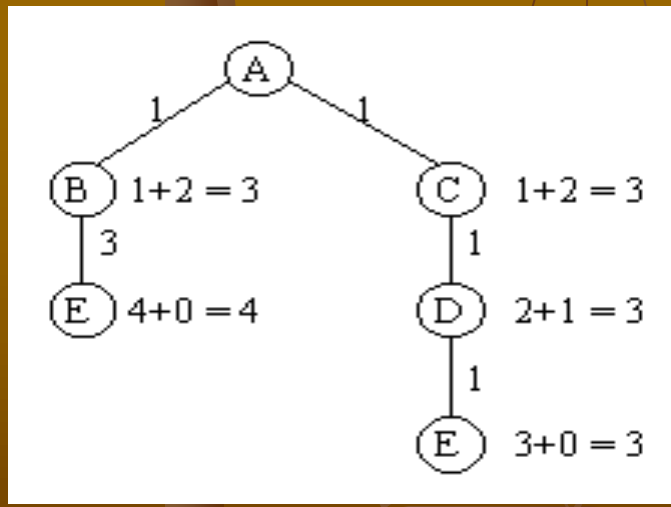
A* Search – An Example (contineud …)

3. C is now the lowest value on the stack with value 3 against the value 4 from E, we expand this node to produce an evaluation of 3 for D.

4. Again, 3 is less than 4, so node D is expanded giving a value of 3 for node E.

5. The node E that came from D has a smaller value than the one from B so is chosen. This node is found to solve the problem so no further expansion is necessary and the shortest distance has been found automatically.





A* Search – An Example (contineud …)

- Note that using the horizontal and vertical distances added together may not always be suitable in which case the Euclidean distance measure would be better, i.e. finding the shortest distance 'as the crow flies'.
- This demonstrates the important measure of informedness. If we have two admissible cost estimates, d1'(n) and d2'(n) then the search using d2'(n) is said to be more informed if d1'(n) < d2'(n) for the majority of n.
- What this is saying is that d2'(n) is the better approximation to d(n) - the actual cost value.
- The more we know beforehand about a state space and can put into the search, the more informed it should be, if we use the knowledge wisely.
- A final word on this technique; although the A* and best-first searches can be made to find the optimal solutions, they are often slow because of much branching.

- When a state is discovered by using heuristic search, is there any guarantee that the same state won't be found later in the search at a cheaper cost (with a shorter path from the start state) This is the property of ***monotonicity***
- In what sense is one heuristic "better" than another? This is the ***informedness*** of a heuristic.
- Heuristics that find the shortest path to a goal whenever it exists are said to be ***admissible***.

Monotonicity

- A heuristic function h is monotone if
  1. h(ni) – h(nj) <= cost(ni, nj)
     where cost(ni, nj) is the actual cost (in number of moves) of going from state ni to nj.
  2. The heuristic evaluation of the goal state is zero i.e. h(goal) = 0.

- A simple argument can show us that any monotonic heuristic is admissible.
- This argument considers any path in the space as a sequence if states s1, s2, … sg, where s1 is the start state and sg is the goal state. Now consider:

s1 to s2	h(s1)-h(s2) <= cost(s1,s2) by monotonic property
s2 to s3	h(s2)-h(s3) <= cost(s2,s3) by monotonic property
…….	
s(g-1) to sg	h(s(g-1))-h(sg) <= cost(s(g-1),sg) by monotonic property

Adding up each column (knowing that h(sg) = 0), we get

s(1) to sgh(s1) <= cost(s1,sg)

- This means that monotonic heuristic h is A* and admissible.

Informedness

- For two heuristics h1 and h2, if h1(n) <= h2(n), for all states n in the search space, heuristic h2 is said to be *more informed* than h1.
- Breadth-first search is equivalent to the A* algorithm with heuristic h1, such that h1(x) = 0 for all states x. This is trivially less than h* (the optimal path).
- The heuristic (we will call it h2): the number of tiles out of place with respect to the goal state, is a lower than h*.
- Thus h1 <= h2 <= h*.
- It thus follows that the "number of tiles out of place" heuristic is more informed than breadth-first search.
- The figure 4.12 (next slide) compare the spaces searched by these two heuristics.
- Both h1 and h2 find the optimal path, but h2 evaluates many fewer states in the process.

Informedness

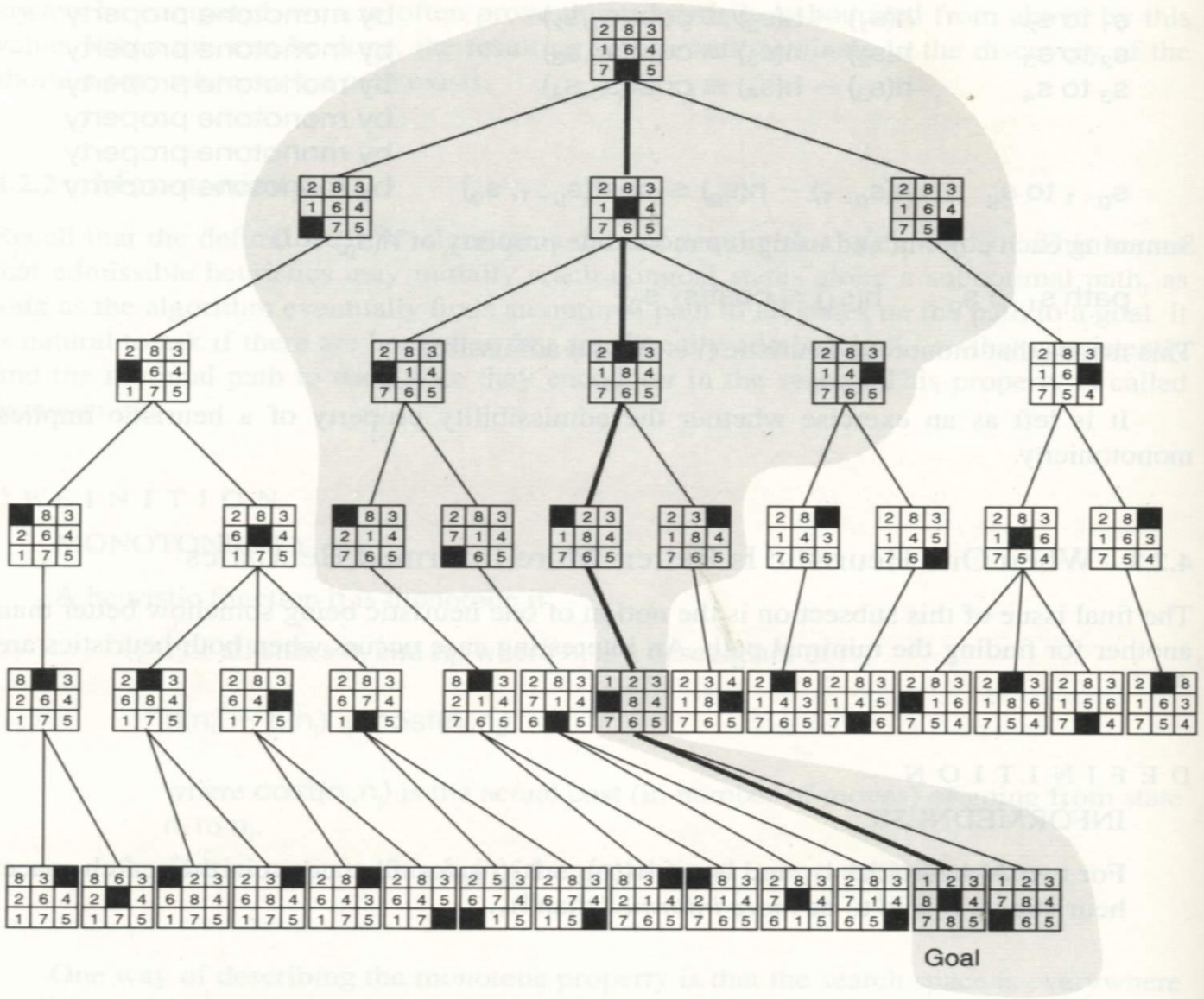


Figure 4.12 Comparison of state space searched using heuristic search with space searched by breadth-first search. The portion of the graph searched heuristically is shaded. The optimal solution path is in bold. Heuristic used is **f(n) = g(n) + h(n)** where **h(n)** is tiles out of place.

Measuring problem-solving performance

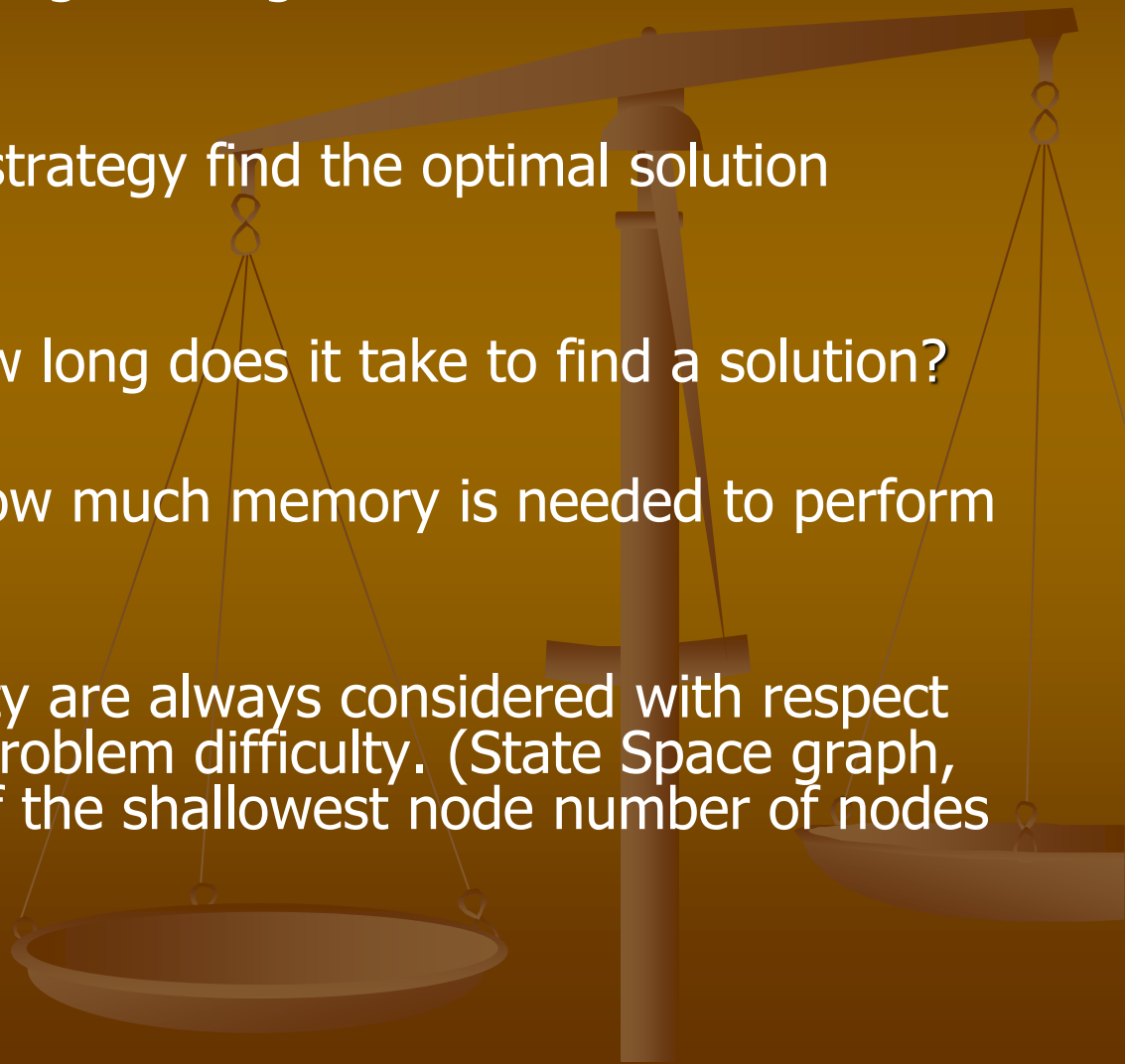
- **Completeness:** Is the algorithm guaranteed to find a solution when there is one?

- **Optimality:** Does the strategy find the optimal solution (lowest cost)?

- **Time complexity:** How long does it take to find a solution?

- **Space complexity:** How much memory is needed to perform the search?

Time and space complexity are always considered with respect to some measure of the problem difficulty. (State Space graph, branching factor, depth of the shallowest node number of nodes generated)



Comparison of Uniformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

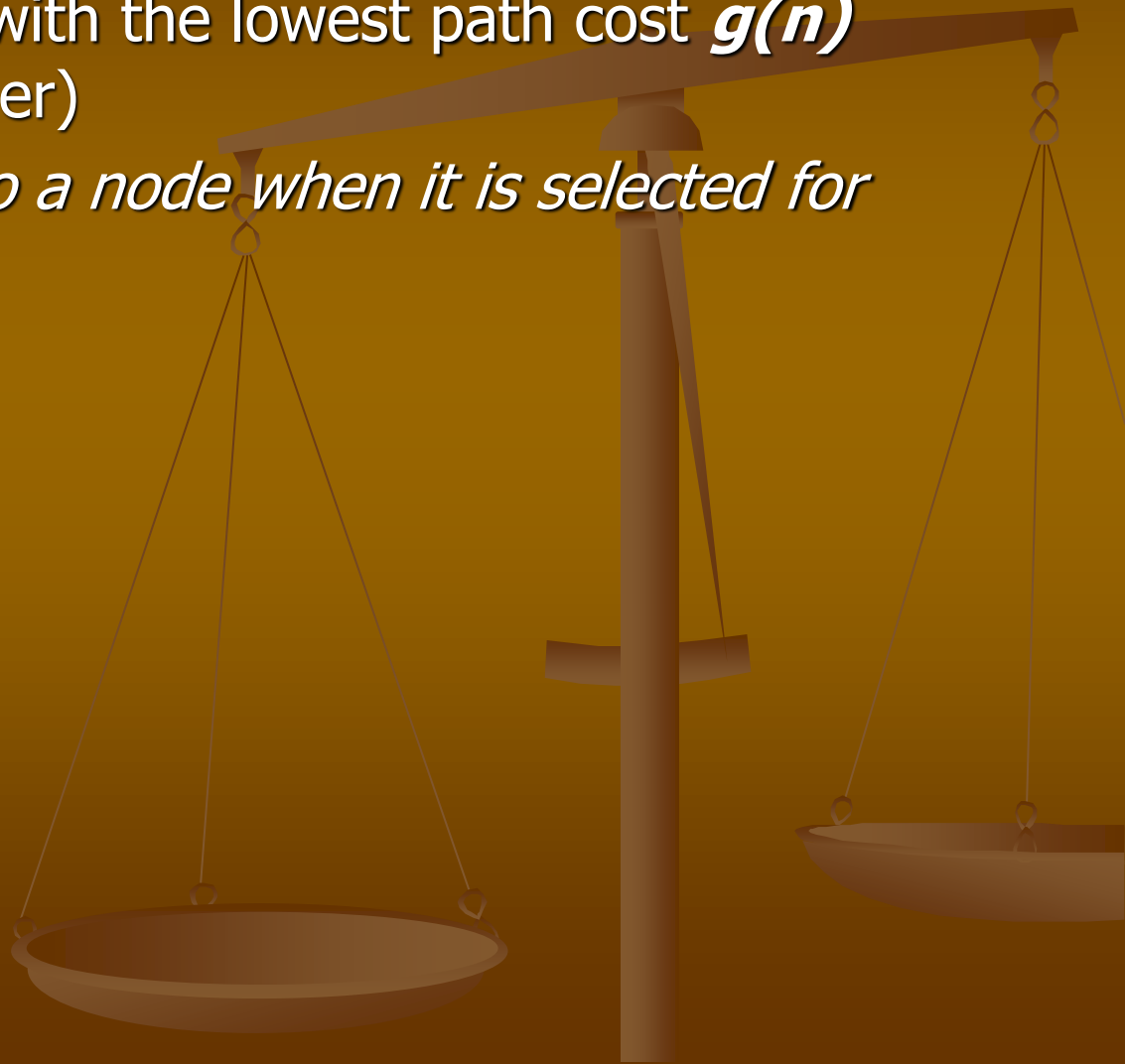
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

- Exponential-complexity search problems cannot be solved by uninformed methods for any but the smallest instances.

Uniform Cost Search

- Expands the node n with the lowest path cost ***g(n)*** (Priority Queue frontier)
- *Goal test is applied to a node when it is selected for expansion.*



Home Work

- Read chapter 3 and 4
  from George F. Luger
- Read chapter 3 and 4
  from Russell & Norvig