Informed Search

Skeleton of Search Algorithms

1. Start with a tree that contains only one node,
   labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n. Let s(n) be the state it is labeled with.
4. If s(n) is a goal state, stop successfully
   and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor
   states of s(n), labeled with that successor state.
6. Go to 2.

Skeleton of Search Algorithms

- The search algorithms differ only in how they select the unexpanded fringe node.
  - If no knowledge other than the current tree is available to guide the decision, then a search algorithm is called uninformed (or blind).
  - Otherwise, a search algorithm is called informed. If the knowledge consists of estimates of the goal distances of the states, the informed search algorithm is called heuristic. By goal distance, we mean the minimum cost of any path (action sequence) from the start state to any goal state.

Best-First Search (for a given priority function f)

1. Start with a tree that contains only one node,
   labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n with the smallest f(n).
   Let s(n) be the state that it is labeled with.
4. If s(n) is a goal state, stop successfully
   and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor states of s(n), labeled with that successor state.
6. Go to 2.

Heuristic Functions

- The efficiency of an uninformed (brute-force) search can be greatly improved by the use of a ***heuristic static evaluation function***, or ***heuristic function***.
- Such a function can improve the efficiency of a search algorithm in two ways:
  - leading the algorithm toward a goal state
  - pruning off branches that don't lie on any optimal solution path.

Properties of Heuristic Functions

- The two most important properties of a heuristic function are:
  - it is relatively cheap to compute
  - it is a relatively accurate estimator of the cost to reach a goal state.

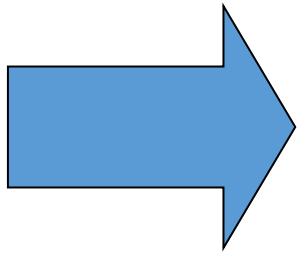
    Usually a heuristic is (roughly) *"good"* if $\frac{1}{2} opt(s) < h(s) < opt(s)$
- Another property:
  - **Admissibility** - the heuristic function is always a lower bound on actual solution cost.
  - In other words h(s) is always **under-estimating**.

Example of Heuristic Functions

- **Task**: Navigating in a network of roads from one location to another
  Heuristic function: Airline distance
- Admissible: Straight line is always a lower bound
- Cheap to calculate: Yes
- Accurate: Normally, yes

Manhattan Distance

	3	1	2
4	5	7	6
8	9	10	11
12	13	15	14



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- **Task**: Sliding-tile puzzles
- **Heuristic function**: Manhattan distance - number of horizontal and vertical grid units where each tile is displaced from its goal position
- Cheap to calculate: Yes
- Accurate: Well, somewhat
- Admissible: Yes - each tile must move at least Manhattan distance

Traveling Salesman Problem

- **Task** :Traveling Salesman Problem
- **Heuristic function**:A cost of minimum spanning tree (MST) of the cities.
- Cheap to calculate (polynomial)
- Admissible: Yes
- Accurate: MST<TSP but also TSP<2*MST
- → ½ TSP <MST < TSP

Pure Heuristic (Greedy Best-First) Search

- Given a heuristic evaluation function, the simplest algorithm that uses it is:

$$f(n) = h(s(n))$$

where f(n) - cost function
h(s(n)) - heuristic function

This algorithm is called **Pure heuristic search (PHS)**

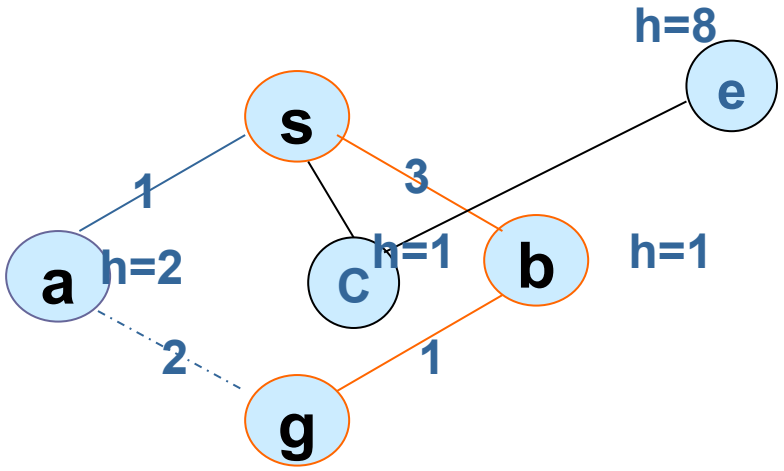
- PHS will eventually generate the entire graph finding a goal node if one exists.
- If the graph is infinite, PHS is not guaranteed to terminate, even if a goal node exists.

PHS (operator cost = positive)

1. Start with a tree that contains only one node,
   labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n with the smallest f(n) = h(s(n)),
   where s(n) is the state that node n is labeled with and h(s(n)) is an
   estimate of the goal distance gd(s(n)) of s(n).
4. If s is a goal state, stop successfully
   and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor
   states of s, labeled with that successor state.
6. Go to 2.

PHS Example

- If the PHS terminates with solution, it is not guaranteed to be an optimal one.
- **Example:**



open

s

c b a

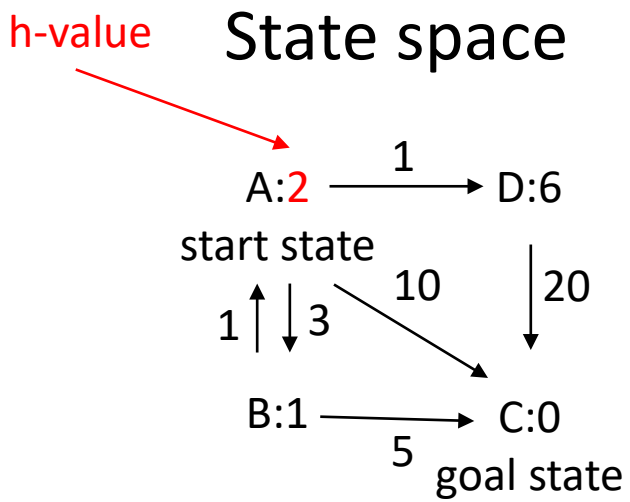
e b a

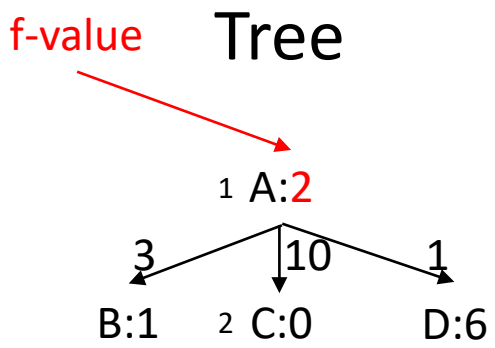
g e a

expand g

Here the algorithm will return a solution of cost 4, when one of cost 3 exists. The problem is that PHS only considers the estimated cost h(s) to a goal when choosing a node for expansion, and doesn't consider the cost g(n) from the initial state to the node.

PHS Example

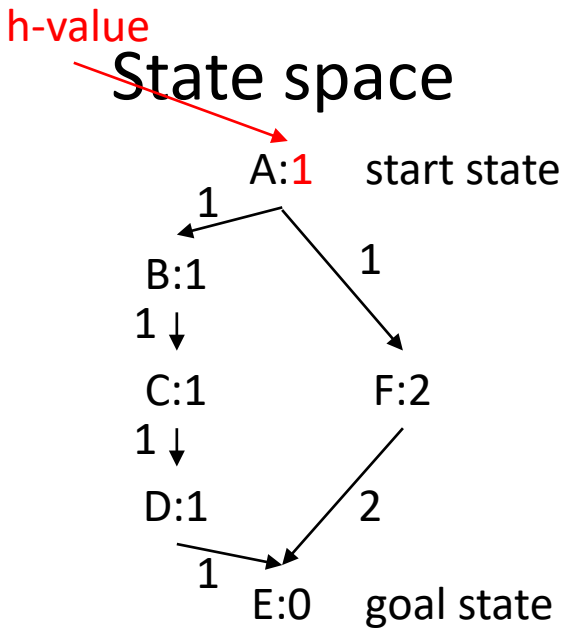


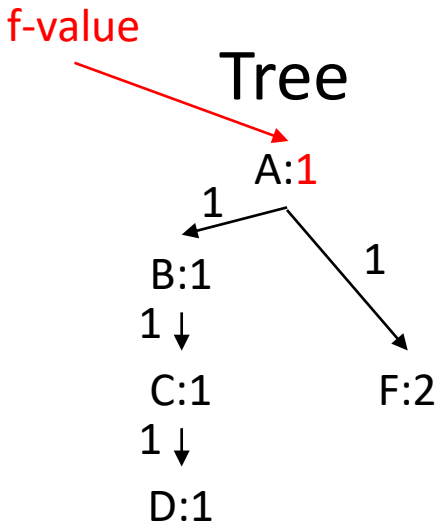


Path: A C
(non-optimal
but often finds paths with
few node expansions)

- Optional pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

PHS Example





- Optional pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- Optional termination rule: terminate once a node labeled with a goal state has been generated.

Properties of PHS

- **Complete**: Yes. It is a best-first search.
- **Optimal:** Certainly not.
- **Space complexity:** Can be exponential (up to the size of the domain)
- **Time complexity:** Hard to tell. Usually, finds the goal fast but not on an optimal path.

A*

- We take into account both the cost of reaching a node form the Initial state, g(n), as well as the heuristic estimate from that node to the goal node, h(s(n)).

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

- For given h(s(n)), f(n) is an estimate of the cost of a cost-minimal path from the root node (start state) along the tree to node n and from there to any goal state.
- The A stands for “algorithm”, and the * indicates its optimality property.

A* (operator costs = positive)

1. Start with a tree that contains only one node,
   labeled with the start state.
2. If there are no unexpanded fringe nodes, stop unsuccessfully.
3. Pick an unexpanded fringe node n with the smallest $f(n) = g(n) + h(s(n))$,
   where s(n) is the state that node n is labeled with, g(n) is the cost from the root to n and h(s(n)) is an estimate of the goal distance gd(s(n)) of s(n).
4. If s is a goal state, stop successfully
   and return the path from the root node to n in the tree.
5. Expand n, that is, create a child node of n for each of the successor states of s, labeled with that successor state.
6. Go to 2.

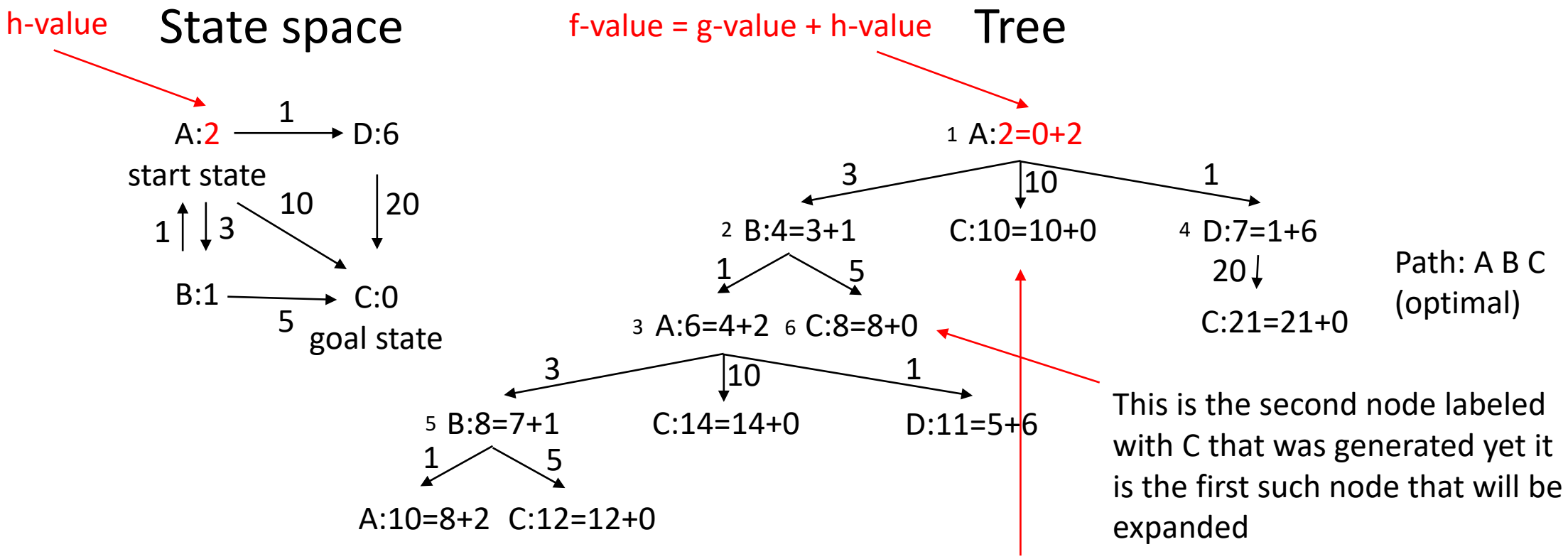
Best-First Search: Cost Functions

- **g(n):** Real distance from the initial state to n
- **h(s(n)):** The estimated remained distance from s(n) to the goal state.

Different cost combinations of g and h

- **f(n)=level(n)** Breadth-First Search.
- **f(n)=g(n)** Dijkstra's algorithms. Uniform-cost Search
- **f(n)=h(s(n))** Pure Heuristic Search (PHS).
- **f(n)=g(n)+h(s(n))** The A* algorithm (1968).

Example: A* (operator cost = positive)



- ~~Termination rule: terminate once a node labeled with a goal state has been generated.~~

Admissible Heuristics and Termination

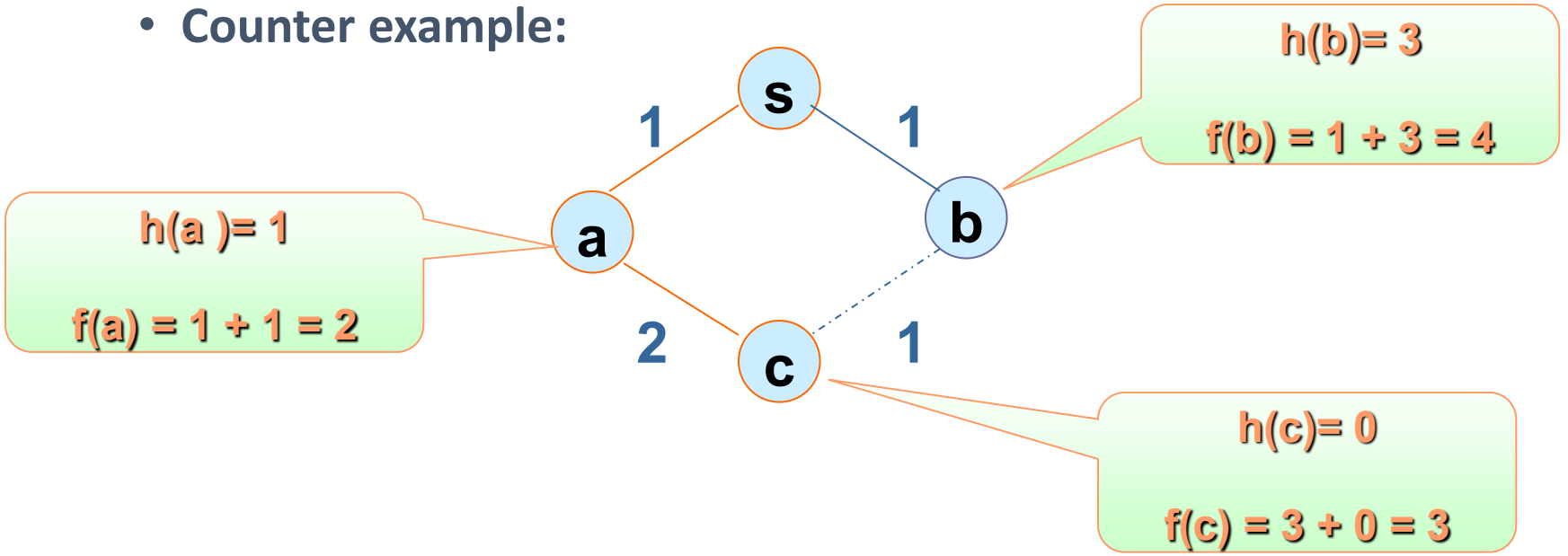
- Like all best-first searches, A* terminates when it chooses a goal node for expansion, or when there are no more Open nodes.
- Admissible condition: the estimated cost to goal always **underestimates** the real cost *h(s(n)) <= gd(s(n))*
- when *h(s(n))* is admissible, so is *f(n)*:*f(n) <= gd**

Recap: *f(n)* is the estimated cost of the cheapest solution through *n*

- We require the h-values to be admissible.
  Otherwise, A* won't be able to find minimum-cost paths.

Solution Quality

- In general, A* is guaranteed to return optimal solutions only if the heuristic is admissible.
- **Counter example:**

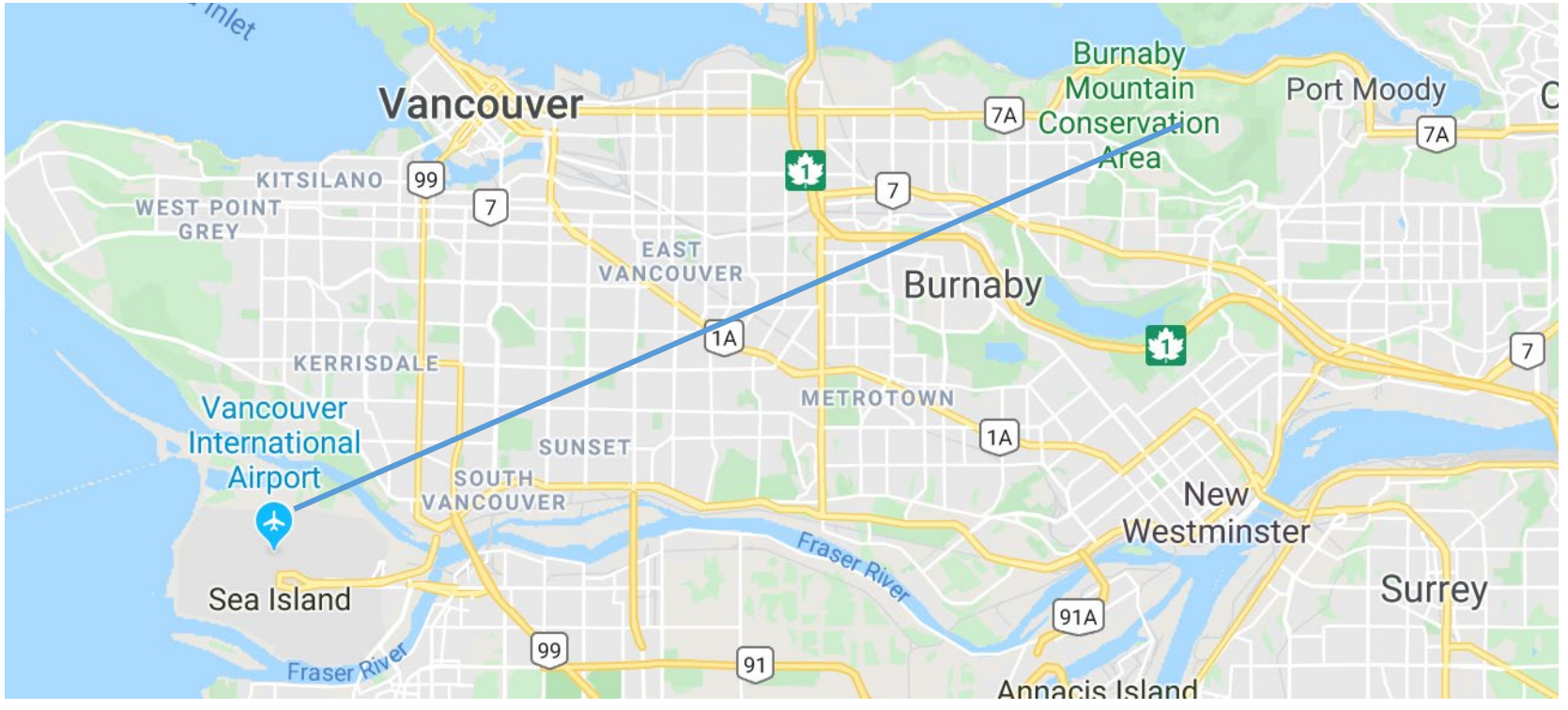


open
s
a b
c b
b

- The problem in this example is that the heuristic value at node b (3) overestimates the cost of reaching a goal from node b, which is only 1.

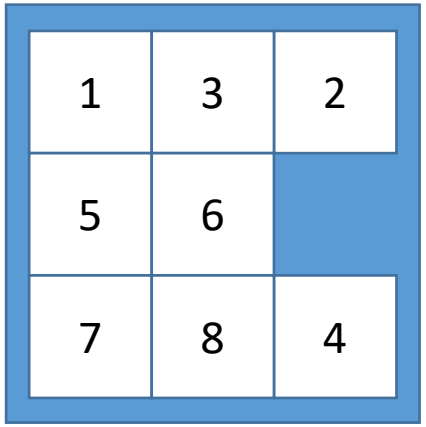
Admissible H-Values

- Find a shortest (not: fastest) path from the SFU main campus to the airport
- Straight-line-distance heuristic
  - h(location) = straight-line distance from the location to the airport

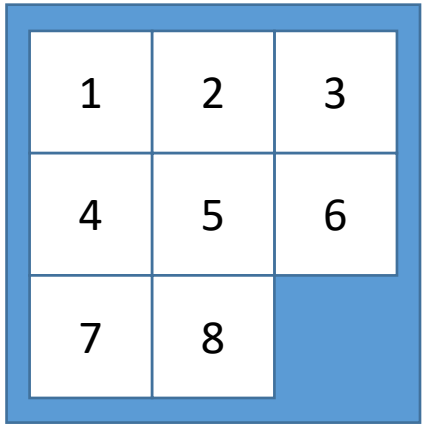


Admissible H-Values

- Find a shortest movement sequence that solves the eight-puzzle
- Tiles-out-of-order heuristic (5 for the example below)
  - h(tile configuration) = the number of tiles not at their correct place
- Manhattan-distance heuristic (0+1+1+3+1+1=7 for the example below)
  - h(tile configuration) = the sum of the x- and y-displacements of each tile from its correct place



current configuration

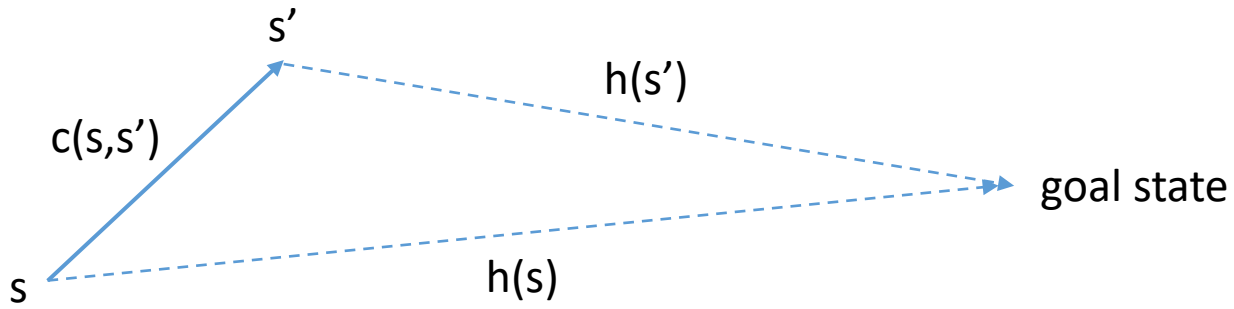


goal configuration

Consistent H-Values

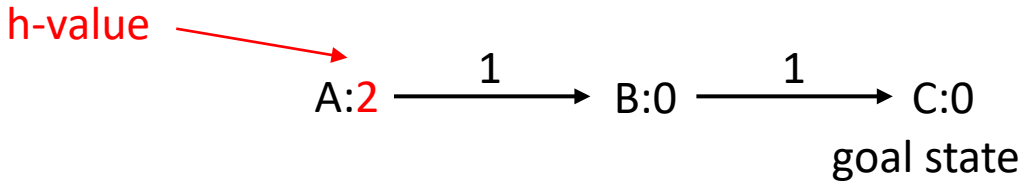
- H-values are called consistent if and only if they satisfy the triangle inequality (c(s,s') is the action cost of moving from s to s'):

  h(s) = 0 for all goal states s, and
  $0 \leq h(s) \leq c(s,s') + h(s')$ for all non-goal states s and their successor states s'.



Consistent H-Values

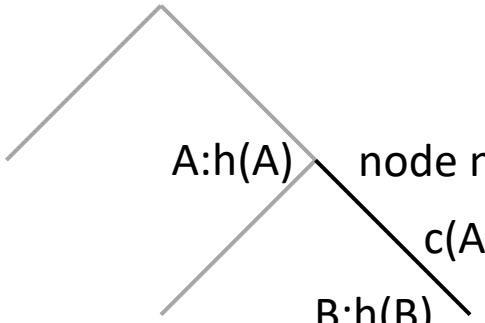
- Admissible h-values are not necessarily consistent:



- Consistent h-values are admissible:
  Proof by induction:
  The statement is true for all states s with gd(s) = 0, i.e. all goal states.
  Now pick any non-goal state s.
  Assume that the statement is true for all states s' with gd(s') < gd(s).
  Pick a cost-minimal path from s to any goal state.
  Let s'' be the successor state of s on that path.
  Then, $0 \leq h(s) \leq c(s,s'') + h(s'') \leq c(s,s'') + gd(s'') = gd(s)$. Qed.

Consistent H-Values

- Consider a search tree for consistent h-values



A:h(A) node n labeled with state A f(n) = g(n) + h(A)

c(A,B)

B:h(B) node n' labeled with state B f(n') = g(n') + h(B)
with g(n') = g(n) + c(A,B)

- Then, $f(n) \leq f(n')$. Proof:

$$h(A) \leq c(A,B)+h(B)$$
$$g(n) + h(A) \leq g(n)+c(A,B) + h(B)$$
$$g(n) + h(A) \leq g(n') + h(B)$$
$$f(n) \leq f(n')$$

- Monotonicity: The f-values of the children of any expanded node are no smaller than the f-value of the expanded node itself.

Consistent H-Values

- Assume that A* picks node n for expansion and that the set of unexpanded fringe nodes at this point in time is OPEN. Then, the f-values of all nodes in OPEN are no smaller than the f-value of node n since A* always picks an unexpanded fringe node with the smallest f-value for expansion (Property A).
- Assume that the set of children of node n after its expansion is C. The f-values of the children of node n are no smaller than the f-value of node n (Property B), see the previous slide.
- (Our argument continues on the next slide…)

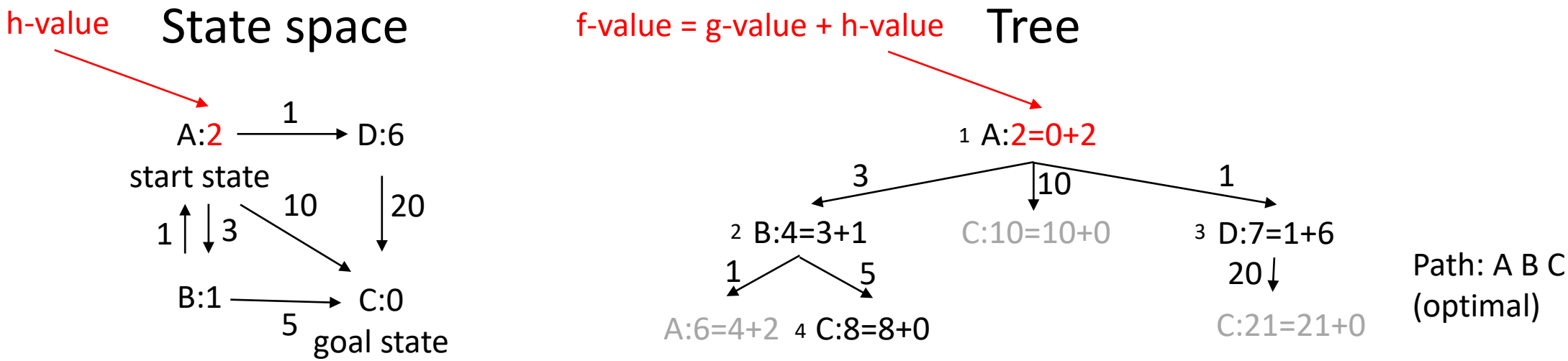
Consistent H-Values

- After the expansion of node n, the new set of unexpanded fringe nodes is OPEN’ := (OPEN\{n})∪C since node n is no longer an unexpanded fringe node but the children of node n have become new unexpanded fringe nodes.
- A* must pick one of the nodes in OPEN’ for the next expansion, and the f-values of all nodes in OPEN’ are no smaller than the f-value of node n according to (Property A) and (Property B).
- Thus, A* expands nodes in order of non-decreasing f-values. That is, a node that A* expands later than some other node has an f-value that is no smaller than the f-value of the other node.

Consistent H-Values

- Now assume that A* expands a node n labeled with state s and later another node n' labeled with the same state s. Then,
  - $f(n) \leq f(n')$
  - $g(n) + h(s) \leq g(n') + h(s)$
  - $g(n) \leq g(n')$
- Thus, the first node that A* expands has the smallest g-value among all nodes labeled with the same state that A* expands. Remember that the g-value of a node corresponds to the length of the path in the tree from the root node to the node, that is, the length of a path found in the state space from the start state to the state that labels the node. A* thus does not need to expand any nodes labeled with the same state as a node that it has already expanded!

Example: A* (operator cost = positive)

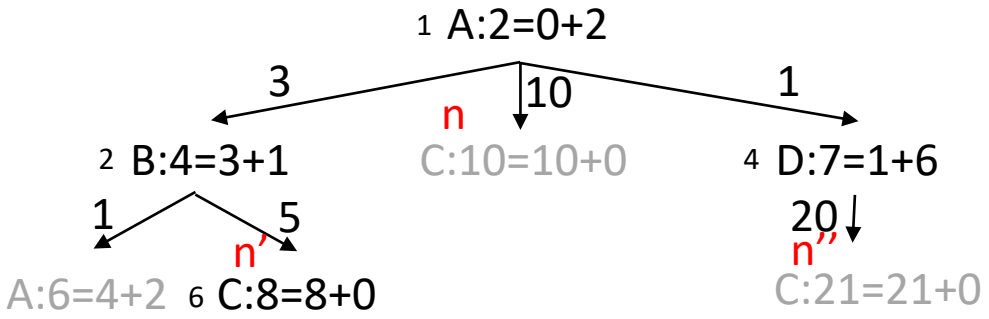


- Optional pruning rule: do not expand a node if a node labeled with the same state has already been expanded. Thus, we can say that states get expanded rather than nodes.
- ~~Termination rule: terminate once a node labeled with a goal state has been generated.~~

Implementation of A*

In this case (if the optional pruning rule is used), every node in the search tree is labeled with a different state – and one can now talk about "states" in the search tree instead of "nodes."

- For each state, maintain at most one node labeled with it, namely the one with the smallest f-value so far (the node might or might not have been expanded already).



- Maintain the unexpanded fringe nodes in a heap (often called OPEN list) with their f-values as keys. Always choose the "top" of the heap (a node in the heap with a smallest f-value) for expansion. Break ties among nodes with the smallest f-value in favor of nodes with larger g-values.

Problem Relaxation

- Obtain a new planning problem by relaxing constraints of the actions (e.g. by deleting preconditions of operator schemata), which can add states and actions to the state space.
- Typically, this is done in a way so that the goal distances for the new planning problem can be computed without search.
- Use the goal distance of a state for the new planning problem as the h-value of the state for the original planning problem.
- The resulting h-values are consistent and thus also admissible.
- Many human-created admissible h-values can be explained as resulting from this process. Thus, in practice, many human-created admissible h-values are consistent!

Problem Relaxation

- Find a shortest (not: fastest) path from the SFU main campus to the airport
- Straight-line-distance heuristic
  - h(location) = straight-line distance from the location to the airport
  - Relaxation: one can drive on- and off-roads

Problem Relaxation

- Find a shortest movement sequence that solves the eight-puzzle
- Tiles-out-of-order heuristic (5 for the example below)
  - h(tile configuration) = the number of tiles not at their correct place
  - Relaxation: one can move any tile to any place in one move, even if that place is already occupied by another tile
- Manhattan-distance heuristic (0+1+1+3+1+1=7 for the example below)
  - h(tile configuration) = the sum of the x- and y-displacements of each tile from its correct place
  - Relaxation: one can move any tile from its current place to any adjacent place, even if that place is already occupied by another tile

Consistent H-Values

- To verify that h-values are consistent,
  - either prove that the triangle inequality holds or
  - show that they can result from a problem relaxation.
- To create consistent h-values,
  - create admissible h-values and verify that they are consistent.

Dominating H-Values

- A* expands nodes in order of non-decreasing f-values. Let gd* be the goal distance of the start state or, equivalently, the g-value and f-value of the node labeled with a goal state that A* is about to expand when it terminates. Then, A* expands
  - all nodes n with $f(n) < gd^*$, and
  - no nodes n with $f(n) > gd^*$.

Dominating H-Values

- H-values h(s) dominate h-values h'(s) if and only if, for all states s, $h(s) \geq h'(s)$.
- Consider consistent h-values h(s) and h'(s) where the h-values h(s) dominate the h-values h'(s). Then, A* with h'(s) expands at least all nodes n that A* with h(s) expands, except perhaps for some states n whose f-values under both searches equal their goal distances.
  Proof: Consider any state n expanded by A* with h(s). Then, $g(n) + h(s(n)) = f(n) \leq gd^*$, which implies that $h'(s(n)) \leq h(s(n)) \leq gd^* - g(n)$. Thus, either $h'(s(n)) = h(s(n)) = gd^* - g(n)$, i.e. $f'(n) = f(n) = gd^*$, or $h'(s(n)) < gd^* - g(n)$, i.e. $f'(n) = g(n) + h'(s(n)) < gd^*$ and A* with h'(s) expands n as well. Qed.

Dominating H-Values

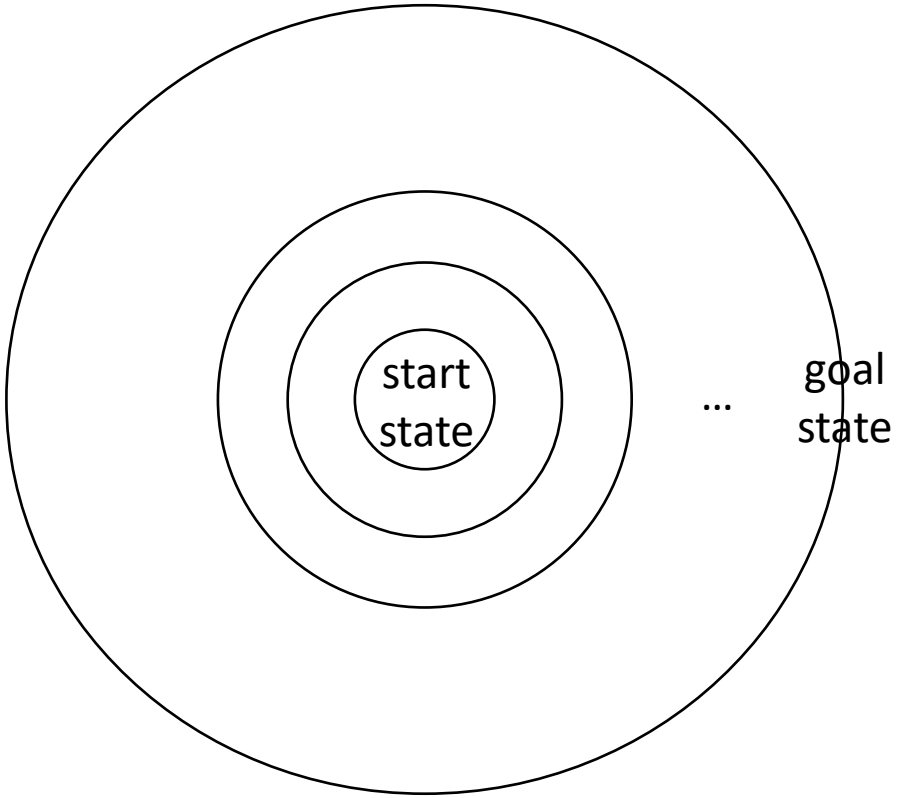
- Given consistent h-values h(s) and h’(s) where the h-values h(s) dominate the h-values h’(s). Then, A* with h’(s) and A* with h(s) both find cost-minimal paths but A* with h(s) runs at least as fast (in terms of node expansions) as A* with h’(s), perhaps up to tie-breaking among nodes whose f-values equal their goal distances.
- Note: This does not take into account that calculating the h-values h(s) and h’(s) can take different amounts of time!

Examples: Dominating H-Values

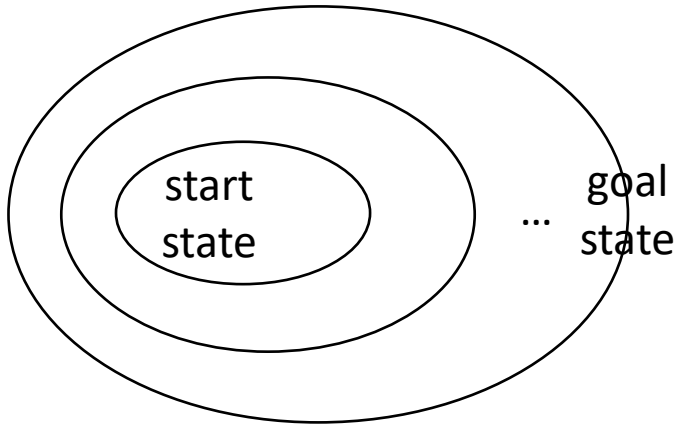
- The tiles-out-of-order h-values and the Manhattan-distance h-values are both consistent (since they result from problem relaxations), and the Manhattan-distance h-values dominate the tiles-out-of-order h-values. Thus, you want to use A* with the Manhattan-distance h-values rather than A* with the tiles-out-of-order h-values.
- Given two consistent h-values h(s) and h’(s), the h-values max(h(s),h’(s)) are consistent and dominate both h(s) and h’(s) (prove it yourself). Thus, you want to use A* with max(h(s),h’(s)) rather than A* with h(s) or A* with h’(s).

Uninformed Search vs. Informed Search

Uniform cost search (A* with h(s) = 0)



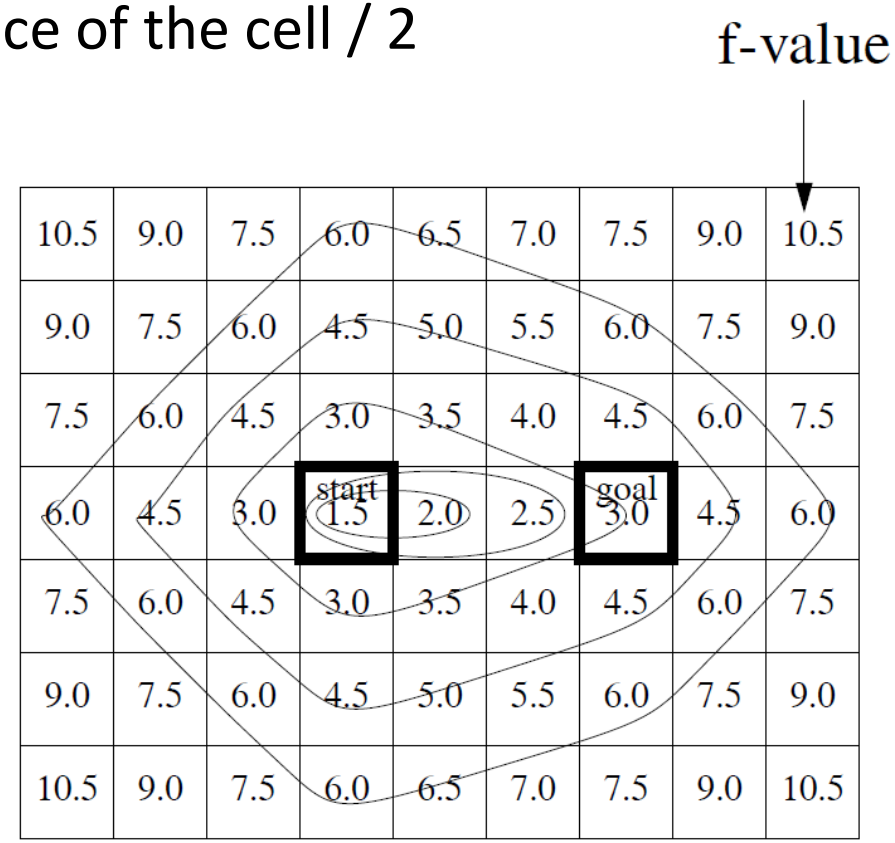
A*



iso f-value contours

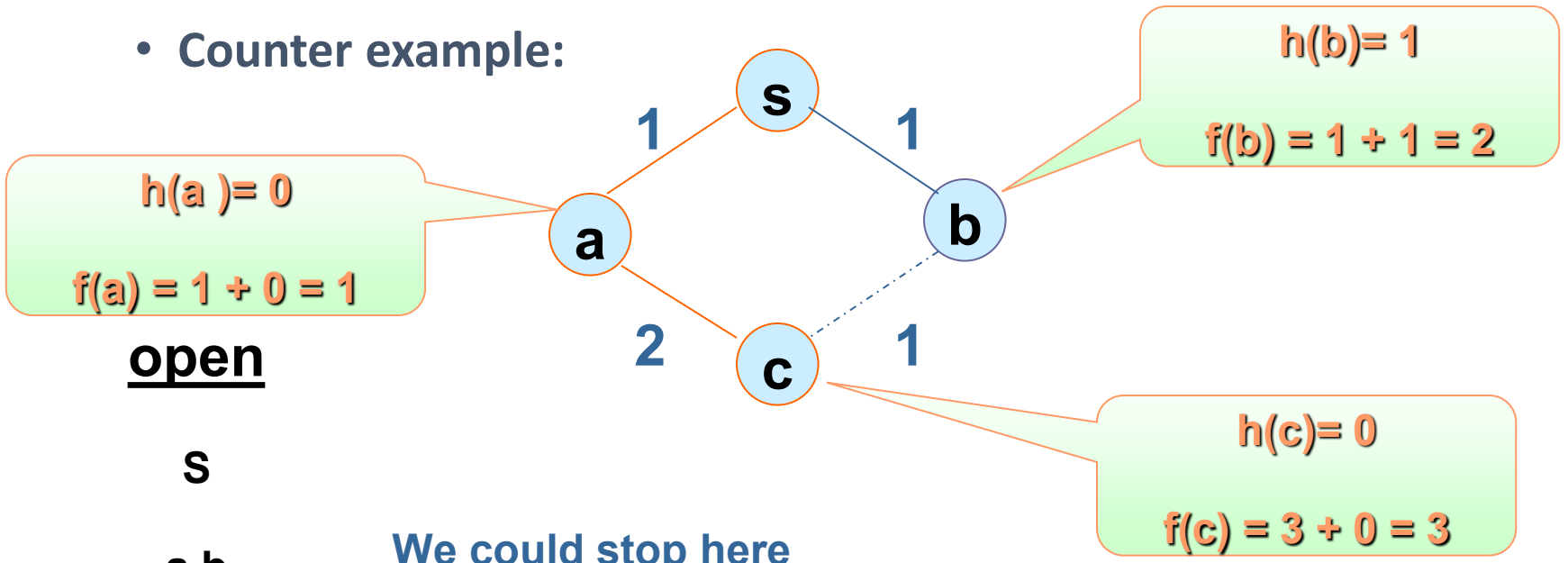
Uninformed Search vs. Informed Search

- Example
  - Grid world in which one can move N, E, S and W with cost 1
  - h(cell) = goal distance of the cell / 2



Termination Condition

- Stop running the algorithm only after the goal node was chosen for expansion.
- **Counter example:**



open

s

a b

b c(3)

---

c(2) c(3)

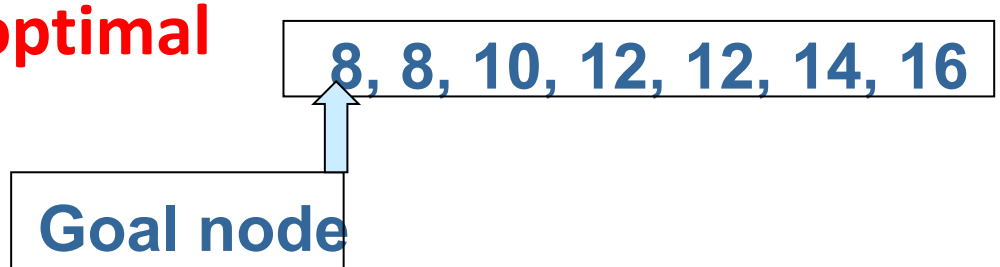
c(3)

We could stop here after generating the goal node

An optimal solution is attained only If we stop here after the goal node is chosen for expansion.

Optimality

- **Theorem:** when A* uses an admissible heuristic h(n), it returns an optimal solution when the goal node is chosen for expansion.
- **Proof:** The open list is a perimeter around the start node.
- Each node n in the open list has a lower bound on the path to the goal via n.
- The node chosen for expansion has the smallest lower bound.
- Descendants of lower bounds cannot have a better lower bound then their parents.
- When the goal node is chosen for expansion it has a real path of cost c, while all other nodes in the open list have lower bounds >=c.
- Thus c is **optimal**



Time Complexity

- The running time of A* is proportional to the number of nodes generated or expanded.Therefore the branching factor is at most a **constant**, and heuristic evaluation of a node can be done in constant time.
- The Open and Closed lists can be maintained in **constant** time per node expansion.
  - **Closed list** - can be organized as hash table, since we only need to check for the occurrence of a node.
  - **Open list** - we have to be able to insert a node and retrieve a lowest-cost node in constant time. This would take time that is logarithmic in the size of the Open list.
  - In many cases, the heuristic functions and edge costs are integer valued or have a small number of distinct values. In such case, the Open can be maintained as an array of lists, separate list for each different cost. This allows **constant**-time insertion and retrieval from the Open list.

Thus, the question is how many nodes A* generates in the process of finding a solution. The answer depends on the quality of the heuristic function.

Special Cases

- **Worst case: Cost function f(n) = g(n)**
  - the heuristic function returns zero for every node and provides no information to the algorithm, but is still a lower-bound on actual cost. This is identical to the UCS or Dijkstra's algorithm, which has a worst-case time complexity of

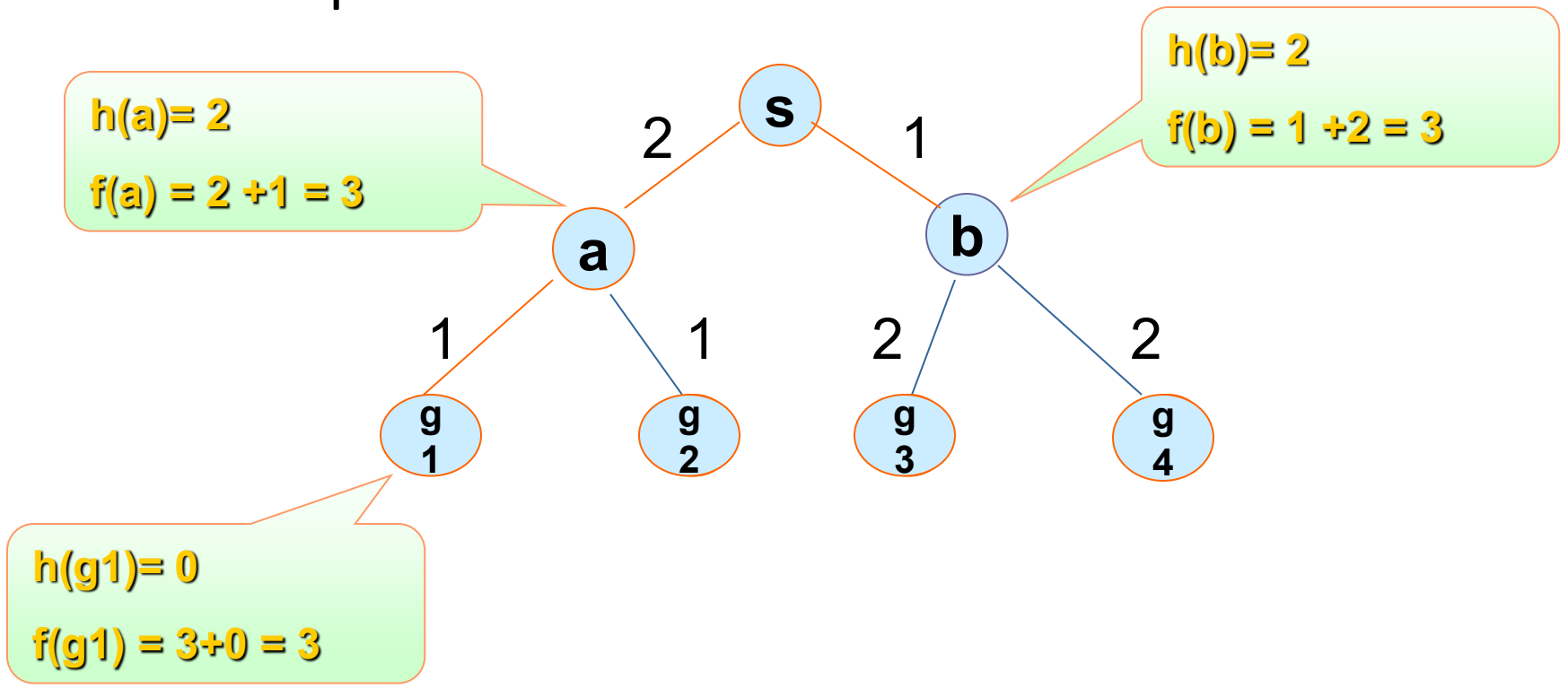
$$O(b^{c/e})$$

- **Best case: Cost function f(n) = g(n) + h*(s(n))**
  - the heuristic function is perfect and always returns the exact optimal cost to a goal state from any given state,The optimal path will be chosen, and the number of node-expansion cycles will be the depth of the optimal path *d*. Thus, the asymptotic time complexity is

$$O(bd) = O(d)$$

Tie Breaking

- Example



Tie Breaking

- Consider a problem where every node at depth *d* is a goal node, and every path to depth *d* is an optimal solution path. As such a tree is explored, every node will have the same cost. If ties are broken in favor of nodes with lower *g* costs **the entire tree may be generated before a goal node.** Thus, the asymptotic time complexity would be **$O(b^d)$,** in spite of the fact that we have a perfect heuristic function.

Tie Breaking

- A better tie breaking rule is to always break ties among nodes with the same f(n) value in favor of nodes with the smallest h(s(n)) value (or the largest g(n) value). This ensures that:
  - any tie will always be broken in favor of a goal node, which has h(Goal) = 0 by definition.
  - the time complexity of A* with a perfect heuristic will be **O(d).**

Conditions for Node Expansion

- If the heuristic function is consistent, then the cost function f(n) is nondecreasing along any path away from the root node. The sequence of nodes expanded by A* starts at the h(Start) and stays the same or increases until it reaches the cost of an optimal solution. Some nodes with the optimal solution cost might be expanded, until a goal node is chosen.

Conditions for Node Expansion

- This means that all nodes n whose cost f(n) < c will certainly be expanded, (where c is optimal solution cost), and no nodes n whose cost f(n) > c will be expanded. Some nodes n whose cost f(n) = c will be expanded.

Thus, $f(n) < c$ is a sufficient condition for A* to expand node n, and $f(n) \leq c$ is a necessary condition.

Time optimality of A*

- **Theorem 3.3** : ***For a given consistent (admissible) heuristic function, every admissible algorithm must expand all nodes surely expanded by A*.***
- **Intuition:** If a node n with f(n)<C is not expanded, maybe there is a solution path going through that node!!
- **Proof:** Suppose that there exists an admissible algorithm *B*, a problem *P*, a consistent heuristic function *h* and a node n such that node n is not expanded by algorithm B on problem P with heuristic function h, but node n is surely expanded by A*, meaning that f(n) = g(n) + h(n) < c.

  Let's construct a new problem P' that is identical to problem P, except for the addition of a single new edge leaving node n, which leads to a new goal node z. Let the cost of the edge from node n to node z be h(n), the heuristic value of node n in problem P', or c'(n,z) = h(n). We use c'(n,z) here to denote the actual cost from n to z in problem P'.

  In problem P', the cost of an optional path from the initial state to the new goal state is g(n) + c'(n,z) = g(n) + h(n) = f(n), since every path to z must go through node n. Since f(n) < c, the cost of an optimal solution to problem P, the optimal solution to problem P' is the path from the start through node n to goal z.

  When we apply algorithm B to problem P'.By assumption, B never expands node n on problem P, thus it must not expand node n on problem P' either. Thus B must fail to find the optimal solution to problem P'. **Contradiction !**

Space Complexity

- **The main drawback of A* is its space complexity.**
- **Like all best-first search algorithms' it stores all the nodes it generates in either the Open list or the Closed list. Thus, its space complexity is the same as its time complexity, assuming that a node can be stored in a constant amount of space.**
- **On current computers, it will typically exhaust the available memory in a number of minutes .**
- **This algorithm is memory- limited.**

Informed Search

- Want to play around with informed search algorithms?
- Go here: http://aispace.org/search/