

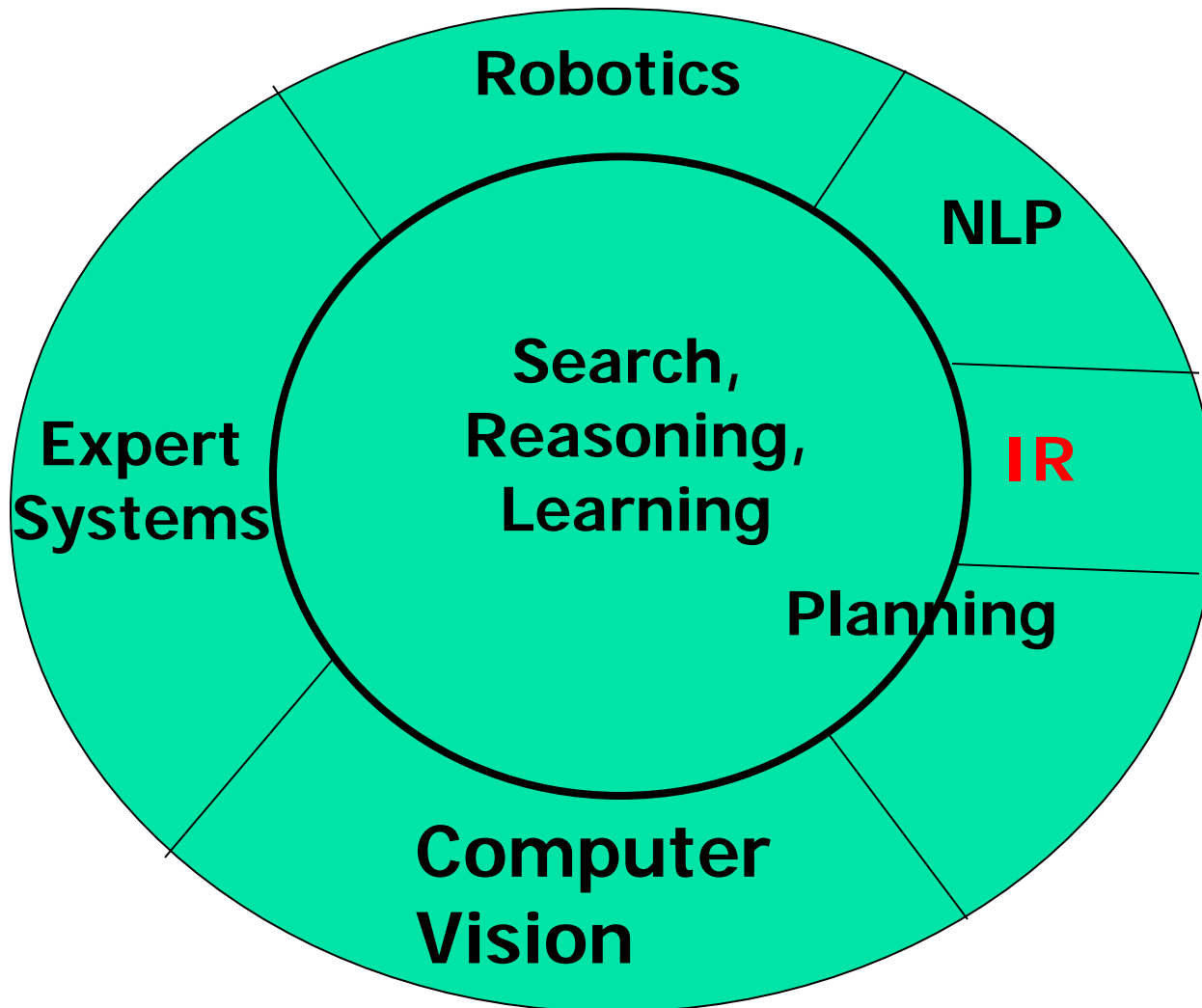
CS561: Artificial Intelligence

A* Search

Pushpak Bhattacharyya
CSE Dept.,
IIT Patna

From 6th Nov, 2017

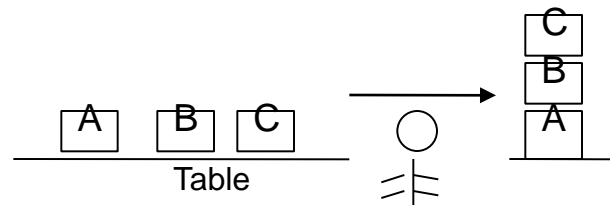
AI Perspective (post-web)



Search: Everywhere

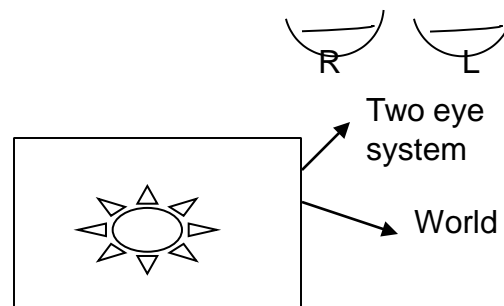
Planning

- (a) which block to *pick*, (b) which to *stack*, (c) which to *unstack*, (d) whether to *stack* a block or (e) whether to *unstack* an already stacked block. These options have to be searched in order to arrive at the right sequence of actions.



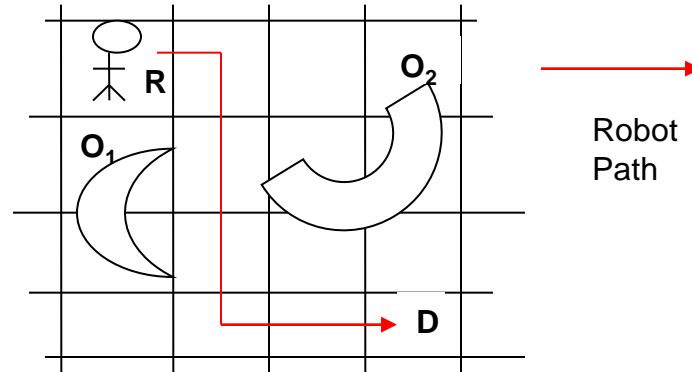
Vision

- A search needs to be carried out to find which point in the image of L corresponds to which point in R . Naively carried out, this can become an $O(n^2)$ process where n is the number of points in the retinal images.



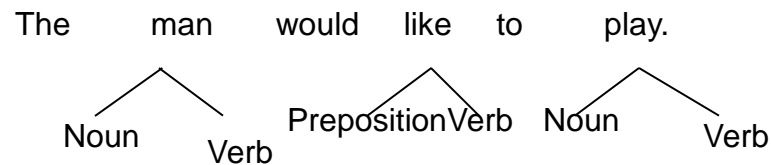
Robot Path Planning

- searching amongst the options of moving *Left*, *Right*, *Up* or *Down*. Additionally, each movement has an associated cost representing the relative difficulty of each movement. The search then will have to find the *optimal*, *i.e.*, the *least cost* path.



Natural Language Processing

- search among many combinations of parts of speech on the way to deciphering the meaning. This applies to every level of processing- *syntax, semantics, pragmatics* and *discourse*.



Expert Systems

Search among rules, many of which can apply to a situation:

If-conditions

the infection is primary-bacteremia

AND the site of the culture is one of the sterile sites

AND the suspected portal of entry is the gastrointestinal tract

THEN

there is suggestive evidence (0.7) that infection is bacteroid

(from MYCIN)

Search building blocks

- State Space : Graph of states (Express constraints and parameters of the problem)
- Operators : Transformations applied to the states.
- Start state : S_0 (Search starts from here)
- Goal state : $\{G\}$ - Search terminates here.
- Cost : Effort involved in using an operator.
- Optimal path : Least cost path

Examples

Problem 1 : 8 – puzzle

4	3	6
2	1	8
7		5

S

1	2	3
4	5	6
7	8	

G

Tile movement represented as the movement of the blank space.

Operators:

L : Blank moves left

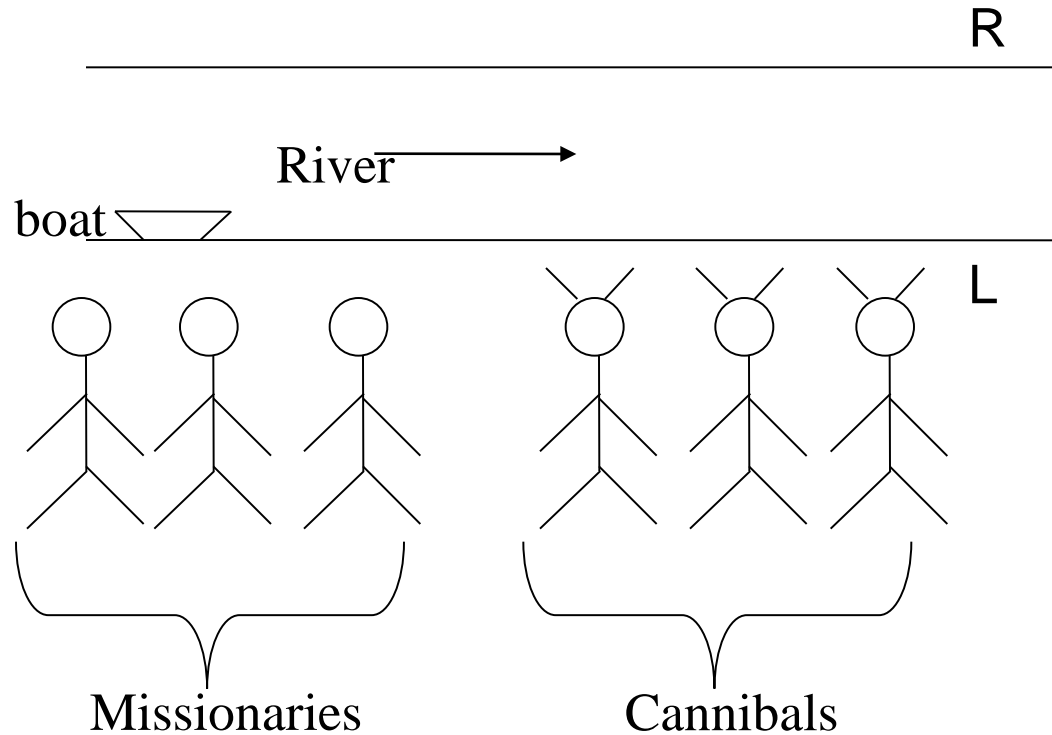
R : Blank moves right

U : Blank moves up

D : Blank moves down

$$C(L) = C(R) = C(U) = C(D) = 1$$

Problem 2: Missionaries and Cannibals



Constraints

- The boat can carry at most 2 people
- On no bank should the cannibals outnumber the missionaries

State : $\langle \#M, \#C, P \rangle$

$\#M$ = Number of missionaries on bank L

$\#C$ = Number of cannibals on bank L

P = Position of the boat

$S0 = \langle 3, 3, L \rangle$

$G = \langle 0, 0, R \rangle$

Operations

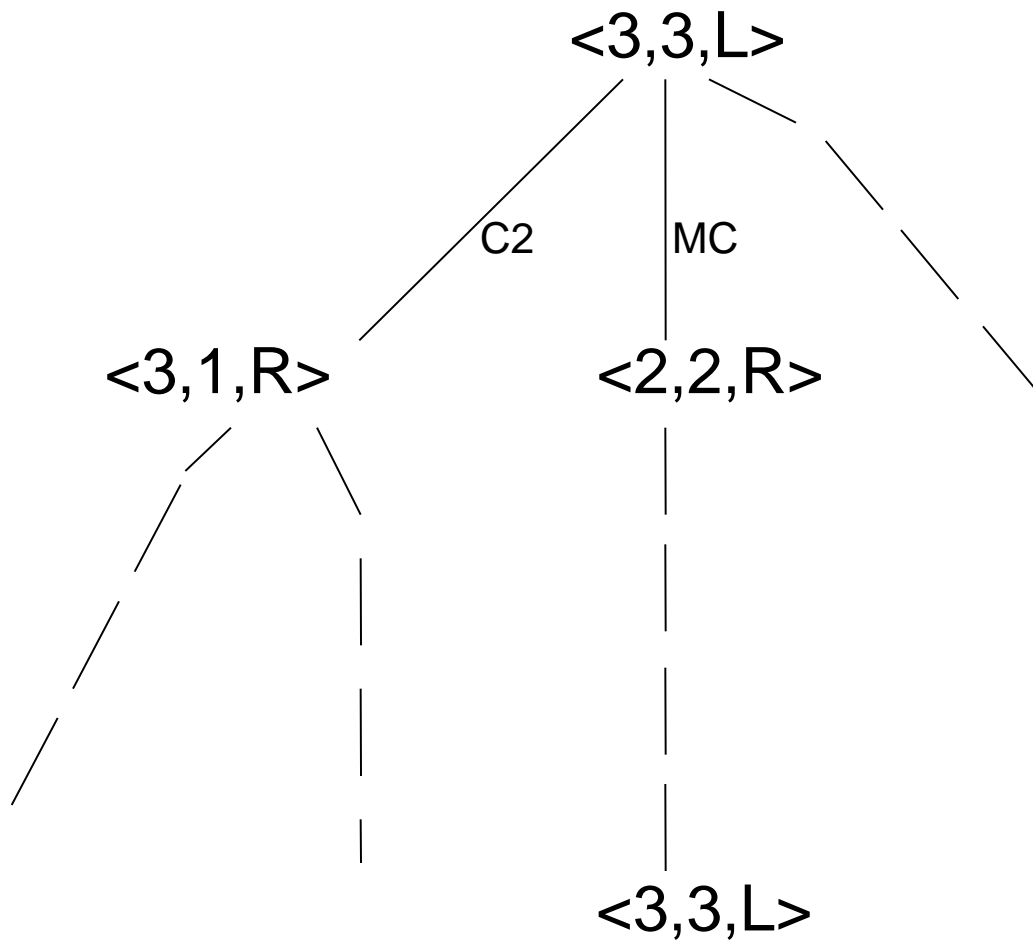
$M2$ = Two missionaries take boat

$M1$ = One missionary takes boat

$C2$ = Two cannibals take boat

$C1$ = One cannibal takes boat

MC = One missionary and one cannibal takes boat



Partial search
tree

Problem 3

B	B	B	W	W	W	
---	---	---	---	---	---	--

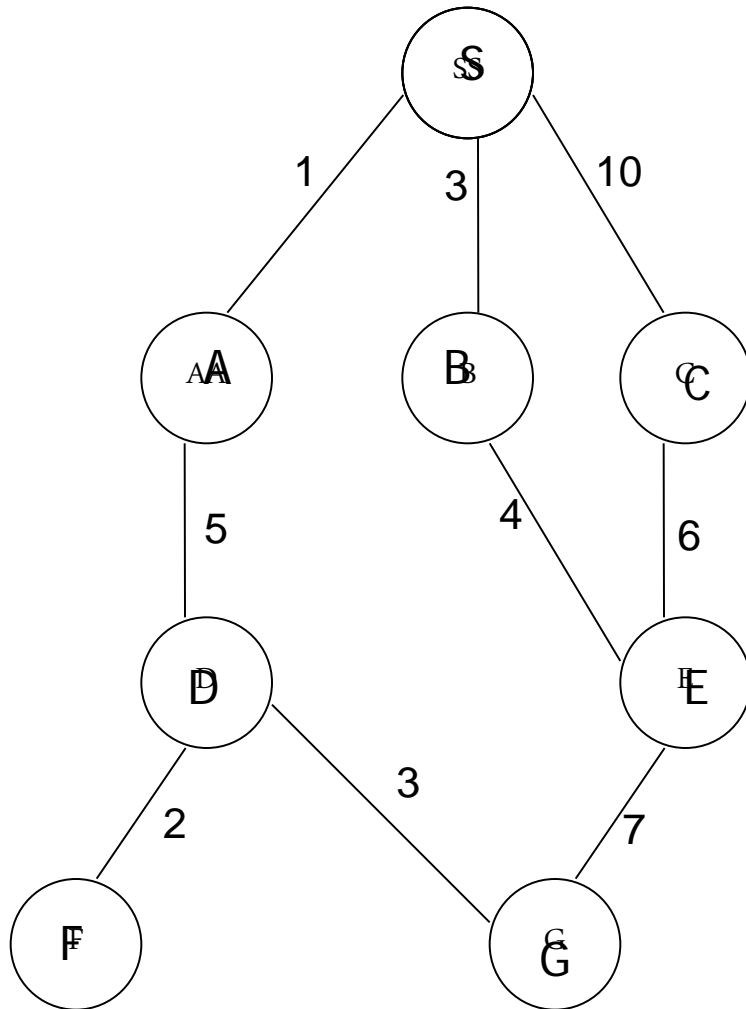
G: States where no **B** is to the left of any **W**

Operators:

- 1) A tile jumps over another tile into a blank tile with cost 2
- 2) A tile translates into a blank space with cost 1

Algorithmics of Search

General Graph search Algorithm



Graph $G = (V, E)$

1) Open List : S $(\emptyset, 0)$

Closed list : \emptyset

2) OL : A $^{(S,1)}$, B $^{(S,3)}$, C $^{(S,10)}$

CL : S

3) OL : B $^{(S,3)}$, C $^{(S,10)}$, D $^{(A,6)}$

CL : S, A

4) OL : C $^{(S,10)}$, D $^{(A,6)}$, E $^{(B,7)}$

CL: S, A, B

5) OL : D $^{(A,6)}$, E $^{(B,7)}$

CL : S, A, B , C

6) OL : E $^{(B,7)}$, F $^{(D,8)}$, G $^{(D, 9)}$

CL : S, A, B, C, D

7) OL : F $^{(D,8)}$, G $^{(D,9)}$

CL : S, A, B, C, D, E

8) OL : G $^{(D,9)}$

CL : S, A, B, C, D, E, F

9) OL : \emptyset

CL : S, A, B, C, D, E,
F, G

Steps of GGS

GENERAL GGraph search

(principles of AI, Nilsson,)

- 1. Create a search graph G , consisting solely of the start node S ; put S on a list called $OPEN$.
- 2. Create a list called $CLOSED$ that is initially empty.
- 3. Loop: if $OPEN$ is empty, exit with failure.
- 4. Select the first node on $OPEN$, remove from $OPEN$ and put on $CLOSED$, call this node n .
- 5. if n is the goal node, exit with the solution obtained by tracing a path along the pointers from n to s in G . (ointers are established in step 7).
- 6. Expand node n , generating the set M of its successors that are not ancestors of n . Install these memes of M as successors of n in G .

Isn't searching the ancestors going to be too expensive we need to search all the ancestor or only the immediate ?

There are two entities in this one is G the graph and the other is T the search Tree. All the parents pointers are part of the tree and tree gives a single path to each node in the graph, whereas the successor of a node are part of the graph i.e. a node can be successor of multiple nodes(G), but it can have only one parent(T)

GGs steps (contd.)

- 7. Establish a pointer to n from those members of M that were not already in G (i.e., not already on either *OPEN* or *CLOSED*). Add these members of M to *OPEN*. For each member of M that was already on *OPEN* or *CLOSED*, 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.
- 8. Reorder the list *OPEN* using some strategy.
- 9. Go *LOOP*.

Why will we redirect the pointer of the children node, instead we should only change the cost as the descendant will automatically be taken care if we insert the node in the open list

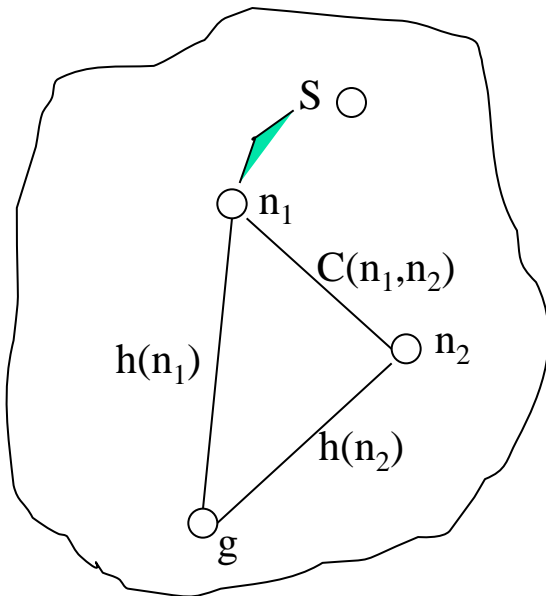
Answer : This step is correct it may be that the successor of the node in the closed list might have a different parent pointer and that parent pointer stands to be modified

GGs is a general umbrella

OL is a
queue
(BFS)

OL is
stack
(DFS)

OL is accessed by
using a functions
 $f = g + h$
(Algorithm A)



$$h(n_1) \leq C(n_1, n_2) + h(n_2)$$

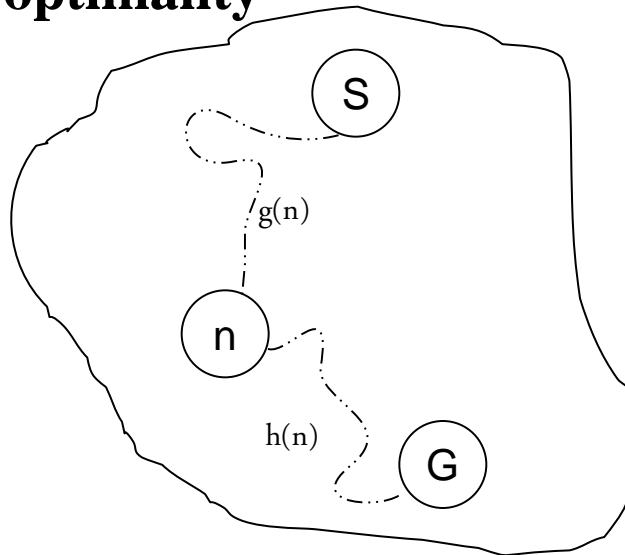
Algorithm A

- A function f is maintained with each node
 $f(n) = g(n) + h(n)$, n is the node in the open list
- Node chosen for expansion is the one with least f value
- For BFS: $h = 0$, g = number of edges in the path to S
- For DFS: $h = 0$, $g = \frac{1}{\text{No of edges in the path to } S}$

Algorithm A*

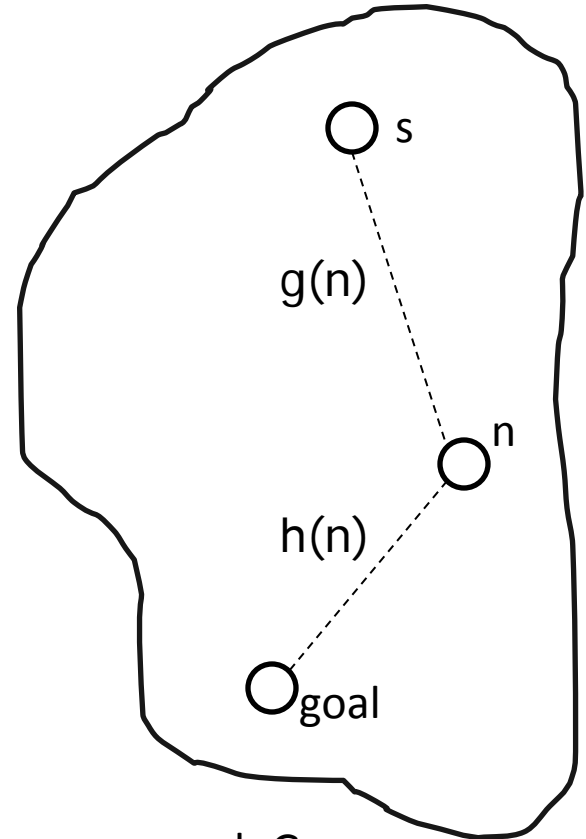
- One of the most important advances in AI
- $g(n)$ = least cost path to n from S found so far
- $h(n) \leq h^*(n)$ where $h^*(n)$ is the actual cost of optimal path to G (node to be found) from n

“Optimism leads to optimality”



A* Algorithm – Definition and Properties

- $f(n) = g(n) + h(n)$
- The node with the least value of f is chosen from the *OL*.
- $f^*(n) = g^*(n) + h^*(n)$,
where,
 $g^*(n)$ = actual cost of the optimal path (s, n)
 $h^*(n)$ = actual cost of optimal path (n, g)
- $g(n) \geq g^*(n)$
- By definition, $h(n) \leq h^*(n)$



State space graph G

8-puzzle: heuristics

Example: 8 puzzle

2	1	4
7	8	3
5	6	

s

1	6	7
4	3	2
5		8

n

1	2	3
4	5	6
7	8	

g

$h^*(n)$ = actual no. of moves to transform n to g

1. $h_1(n)$ = no. of tiles displaced from their destined position.
2. $h_2(n)$ = sum of Manhattan distances of tiles from their destined position.

$$h_1(n) \leq h^*(n) \text{ and } h_2(n) \leq h^*(n)$$

h^*	
h_2	
h_1	

Comparison

A* critical points

- **Goal**

1. Do we know the goal?
2. Is the distance to the goal known?
3. Is there a path (known?) to the goal?

A* critical points

- **About the path**

Any time before A* terminates there exists on the OL, a node from the optimal path all whose ancestors in the optimal path are in the CL.

This means,

\exists in the OL always a node 'n' s.t.

$$g(n) = g^*(n)$$

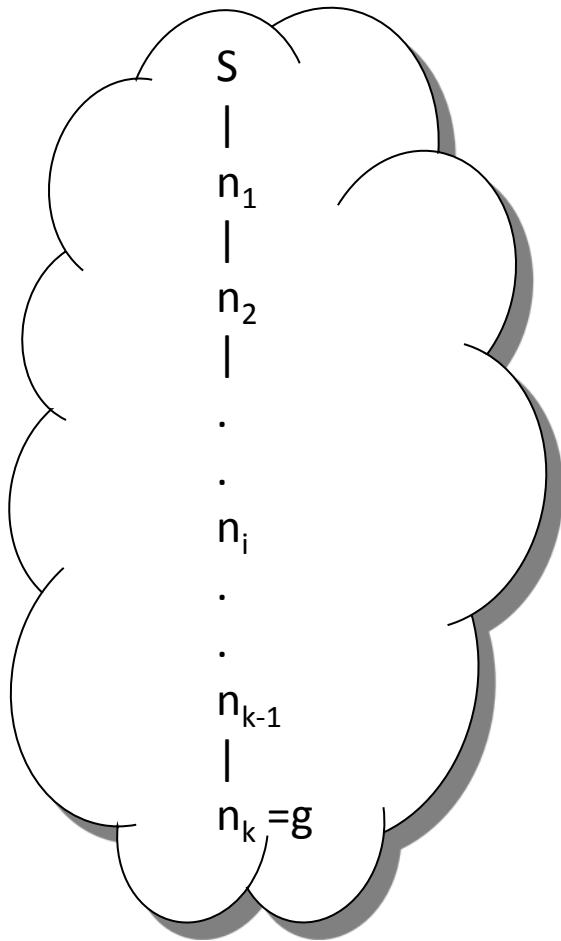
Key point about A^* search

Statement:

Let $S - n_1 - n_2 - n_3 \dots n_i \dots - n_{k-1} - n_k (=G)$ be an optimal path.

At any time during the search:

1. There is a node n_i from the optimal path in the OL
2. For n_i all its ancestors $S, n_1, n_2, \dots, n_{i-1}$ are in CL
3. $g(n_i) = g^*(n_i)$



Proof of the statement

Proof by induction on iteration no. j

Basis : $j = 0$, S is on the OL, S satisfies the statement

Hypothesis : Let the statement be true for $j = p$ (p^{th} iteration)

Let n_i be the node satisfying the statement

Proof (continued)

Induction : Iteration no. $j = p+1$

Case 1 : n_i is expanded and moved to the closed list

Then, n_{i+1} from the optimal path comes to the OL

Node n_{i+1} satisfies the statement
(note: if n_{i+1} is in CL, then n_{i+2} satisfies the property)

?? at a given time there should be only one node in the successor of n_i which should be on the optimal path
this statement seems flawed

Case 2 : Node $x \neq n_i$ is expanded

Here, n_i satisfies the statement

A* Algorithm- Properties

- **Admissibility:** An algorithm is called admissible if it always terminates and terminates in optimal path
- **Theorem:** A* is admissible.
- **Lemma:** Any time before A* terminates there exists on OL a node n such that $f(n) \leq f^*(s)$
- **Observation:** For optimal path $s \rightarrow n_1 \rightarrow n_2 \rightarrow \dots \rightarrow g$,
 1. $h^*(g) = 0$, $g^*(s) = 0$ and
 2. $f^*(s) = f^*(n_1) = f^*(n_2) = f^*(n_3) \dots = f^*(g)$

A* Properties (*contd.*)

$$f^*(n_i) = f^*(s), \quad n_i \neq s \text{ and } n_i \neq g$$

Following set of equations show the above equality:

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

$$f^*(n_{i+1}) = g^*(n_{i+1}) + h^*(n_{i+1})$$

$$g^*(n_{i+1}) = g^*(n_i) + c(n_i, n_{i+1})$$

$$h^*(n_{i+1}) = h^*(n_i) - c(n_i, n_{i+1})$$

Above equations hold since the path is optimal.

Admissibility of A*

A* always terminates finding an optimal path to the goal if such a path exists.

Intuition

Lemma 1 : This is the first lemma

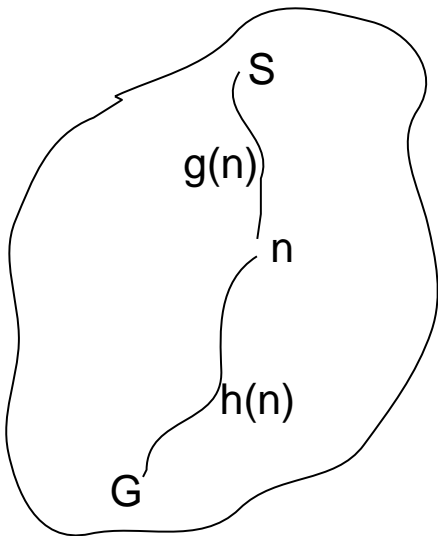
(1) In the open list there always exists a node n such that $f(n) \leq f^*(S)$.

(2) If A* does not terminate, the f value of the nodes expanded become **unbounded**.

what is meant by this statement

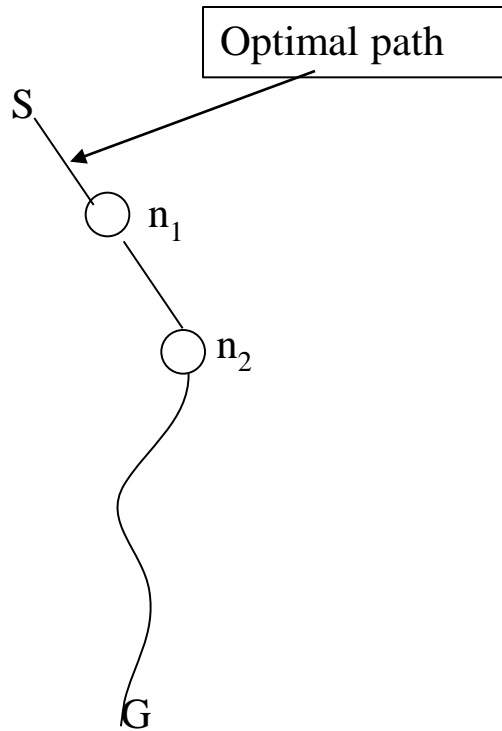
1) and 2) are together inconsistent

Hence A* must terminate



Lemma

Any time before A^* terminates there exists in the open list a node n' such that $f(n') \leq f^*(S)$



For any node n_i on optimal path,

$$f(n_i) = g(n_i) + h(n_i) \\ \leq g^*(n_i) + h^*(n_i)$$

$$\text{Also } f^*(n_i) = f^*(S)$$

Let n' be the first node in the optimal path that is in OL. Since all parents of n' in the optimal have gone to CL,

As all parent of the n' are in the closed path of the optimal path the current g value of the n' node is optimal in the sense

$$g(n') = g^*(n') \text{ and } h(n') \leq h^*(n') \\ \Rightarrow f(n') \leq f^*(S)$$

If A^* does not terminate

Let e be the least cost of all arcs in the search graph.

Then $g(n) \geq e \cdot l(n)$ where $l(n) = \#$ of arcs in the path from S to n found so far. If A^* does not terminate, $g(n)$ and hence $f(n) = g(n) + h(n)$ [$h(n) \geq 0$] will become unbounded.

There is an hidden assumption that the $g(n)$ values in the open list will become unbounded then as there in the openlist which has a bound hence this becomes contradictory

This is not consistent with the lemma. So A^* has to terminate.

2nd part of admissibility of A*

The path formed by A* is optimal when it has terminated

Proof

Suppose the path formed is not optimal

Let G be expanded in a non-optimal path.

At the point of expansion of G ,

$$\begin{aligned} f(G) &= g(G) + h(G) \\ &= g(G) + 0 \\ &> g^*(G) = g^*(S) + h^*(S) \\ &= f^*(S) [f^*(S) = \text{cost of optimal path}] \end{aligned}$$

The point to be mentioned over here is that, $f(G) > f^*(G)$ whereas there always exists a node on the opmelist such that $f(n) < f^*(G)$ hence no node shuold habve been popped before this node was popped

Cross check this point

This is a contradiction

So path should be optimal

Summary on Admissibility

- 1. A^* algorithm halts
- 2. A^* algorithm finds optimal path
- 3. If $f(n) < f^*(S)$ then node n has to be expanded before termination
- 4. If A^* does not expand a node n before termination then $f(n) \geq f^*(S)$

Exercise-1

Prove that if the distance of every node from the goal node is "known", then no "search:" is necessary

Ans:

- For every node n , $h(n)=h^*(n)$. The algo is A^* .
- Lemma proved: any time before A^* terminates, there is a node m in the OL that has $f(m) \leq f^*(S)$, S = start node (m is the node on the optimal path all whose ancestors in the optimal path are in the closed list).
- For m , $g(m)=g^*(m)$ and hence $f(m)=f^*(S)$.
- Thus at every step, the node with $f=f^*$ will be picked up, and the journey to the goal will be completely directed and definite, with no "search" at all.
- Note: when $h=h^*$, f value of any node on the OL can never be less than $f^*(S)$.
which is in lieu of the above statement that all the nodes having $f(n) < f^*(S)$ will definitely be expanded before the termination of the algorithm

Exercise-2

If the h value for every node over-estimates the h^* value of the corresponding node by a constant, then the path found need not be costlier than the optimal path by that constant. Prove this.

revise

Ans:

- Under the condition of the problem, $h(n) \leq h^*(n) + c$.
- Now, any time before the algo terminates, there exists on the OL a node m such that $f(m) \leq f^*(S) + c$.
- The reason is as follows: let m be the node on the optimal path all whose ancestors are in the CL (there *has to be* such a node).
- Now, $f(m) = g(m) + h(m) = g^*(m) + h(m) \leq g^*(m) + h^*(m) + c = f^*(S) + c$
- When the goal G is picked up for expansion, it must be the case that
- $f(G) \leq f^*(S) + c = f^*(G) + c$
- i.e., $g(G) \leq g^*(G) + c$, since $h(G) = h^*(G) = 0$.

Better Heuristic Performs
Better

Theorem

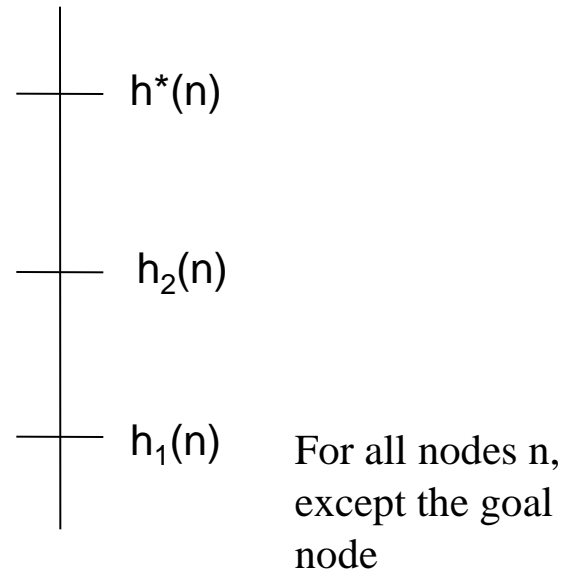
A version A_2^* of A^* that has a “better” heuristic than another version A_1^* of A^* performs at least “as well as” A_1^*

Meaning of “better”

$h_2(n) > h_1(n)$ for all n

Meaning of “as well as”

A_1^* expands at least all the nodes of A_2^*



Proof by induction on the search tree of A_2^* .

A^* on termination carves out a tree out of G

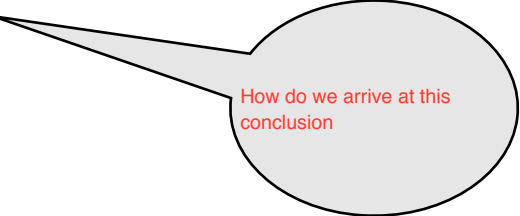
Induction

on the depth k of the search tree of A_2^* . A_1^* before termination expands all the nodes of depth k in the search tree of A_2^* .

$k=0$. True since start node S is expanded by both

Suppose A_1^* terminates without expanding a node n at depth $(k+1)$ of A_2^* search tree.

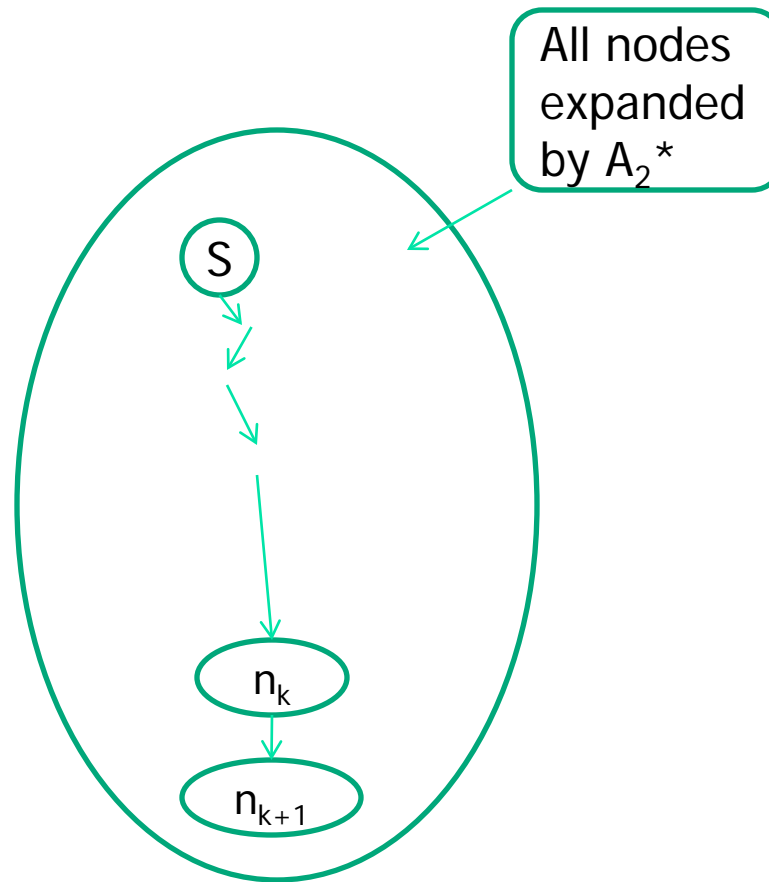
Since A_1^* has seen all the parents of n seen by A_2^*
 $g_1(n) \leq g_2(n) \quad (1)$



How do we arrive at this conclusion

Proof for $g_1(n_{k+1}) \leq g_2(n_{k+1})$

(1/3)



Proof for $g_1(n_{k+1}) \leq g_2(n_{k+1})$ (2/3)

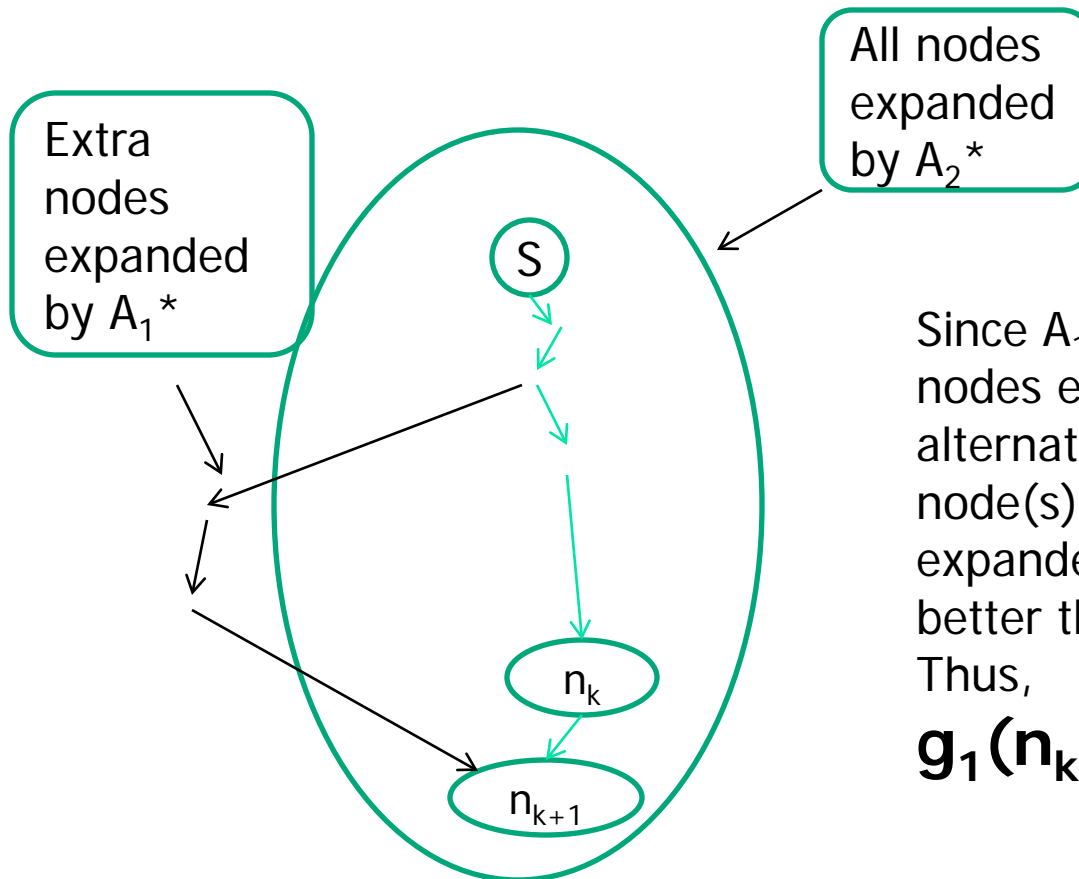
Case 1: n_k is the parent of n_{k+1} even in A_1^*

$$\begin{aligned} g_1(n_{k+1}) &= g_1(n_k) + \text{cost}(n_k, n_{k+1}) \\ &\leq g_2(n_k) + \text{cost}(n_k, n_{k+1}) \dots\dots\dots(1) \\ &\leq g_2(n_{k+1}) \end{aligned}$$

(1) $\rightarrow g_1(n_k)$ has to be less than $g_2(n_k)$ as all nodes expanded by A_2^* have been expanded by A_1^* too. Therefore, it can only find a path better than A_1^* to n_k .

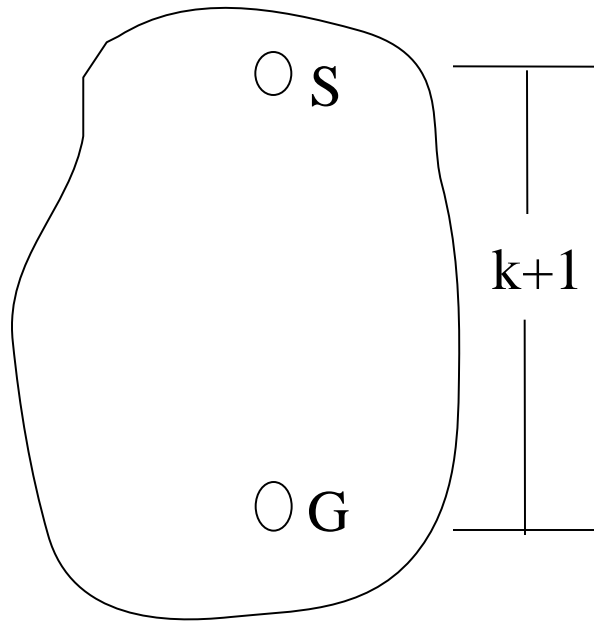
Proof for $g_1(n_{k+1}) \leq g_2(n_{k+1})$ (3/3)

Case 2: n_k is not the parent of n_{k+1} in A_1^*



Since A_1^* has already expanded all nodes expanded by A_2^* , if it finds an alternative path through some node(s) other than the ones expanded by A_2^* , then it has to be better than the path as per A_2^* . Thus,

$$g_1(n_{k+1}) \leq g_2(n_{k+1})$$



Since A_1^* has terminated without expanding n ,

$$f_1(n) \geq f^*(S) \quad (2)$$

here n should have been $n(k+1)$

Any node whose f value is strictly less than $f^*(S)$ has to be expanded.

Since A_2^* has expanded n

$$f_2(n) \leq f^*(S) \quad (3)$$

From (1), (2), and (3)

$h_1(n) \geq h_2(n)$ which is a contradiction. Therefore, A_1^* has to expand all nodes that A_2^* has expanded.

Exercise

If better means $h_2(n) > h_1(n)$ for some n and $h_2(n) = h_1(n)$ for others, then Can you prove the result ?

for some $h_1(n) = h_2(n)$ we can say that still it will be a contradiction will be atleast as better as h_2

Monotonicity

Steps of GGS

(principles of AI, Nilsson,)

- 1. Create a search graph G , consisting solely of the start node S ; put S on a list called $OPEN$.
- 2. Create a list called $CLOSED$ that is initially empty.
- 3. Loop: if $OPEN$ is empty, exit with failure.
- 4. Select the first node on $OPEN$, remove from $OPEN$ and put on $CLOSED$, call this node n .
- 5. if n is the goal node, exit with the solution obtained by tracing a path along the pointers from n to s in G . (ointers are established in step 7).
- 6. Expand node n , generating the set M of its successors that are not ancestors of n . Install these memes of M as successors of n in G .

GGs steps (contd.)

- 7. Establish a pointer to n from those members of M that were not already in G (*i.e.*, not already on either *OPEN* or *CLOSED*). Add these members of M to *OPEN*. For each member of M that was already on *OPEN* or *CLOSED*, 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.
- 8. Reorder the list *OPEN* using some strategy.
- 9. Go *LOOP*.

Illustration for CL parent pointer redirection recursively

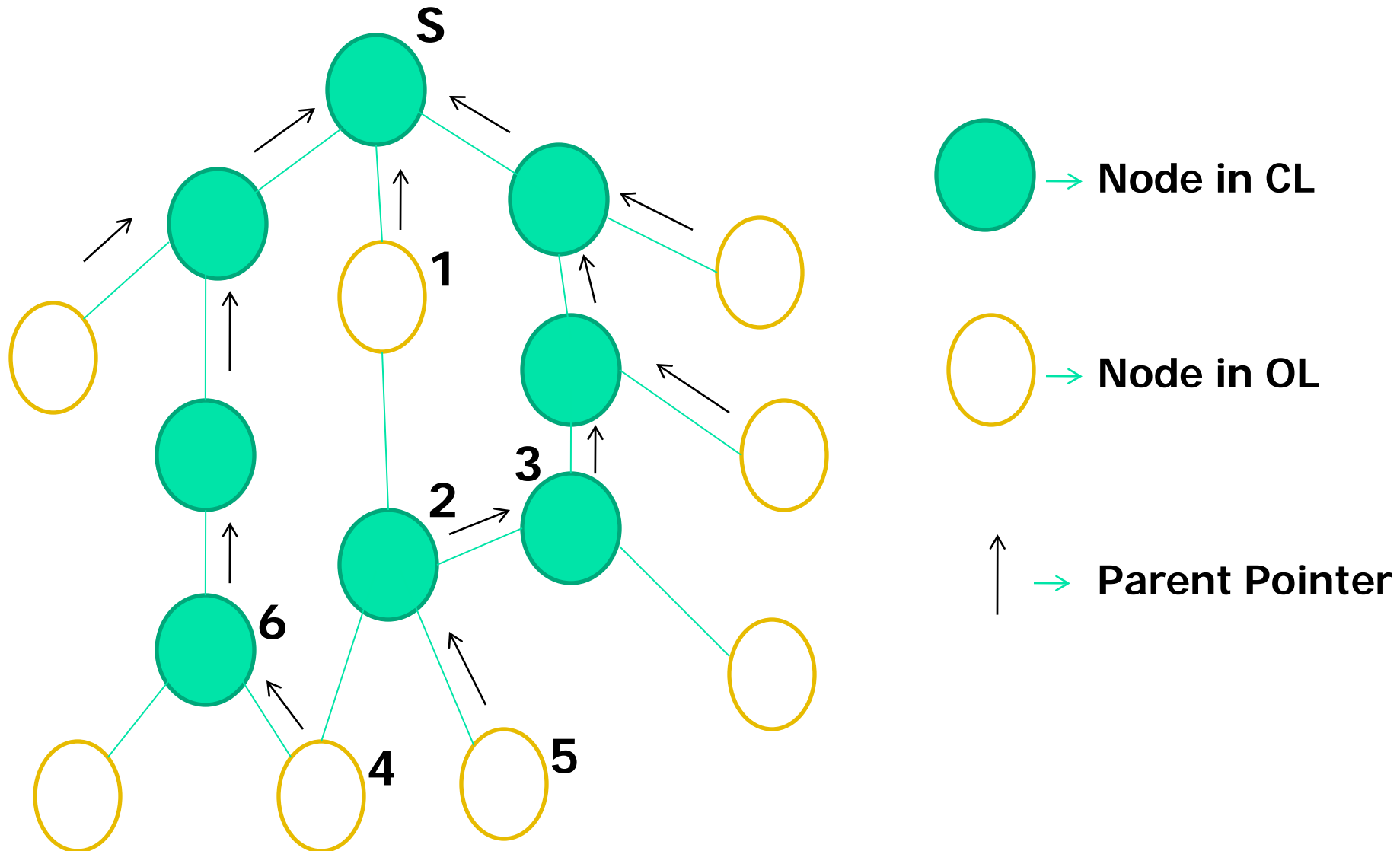
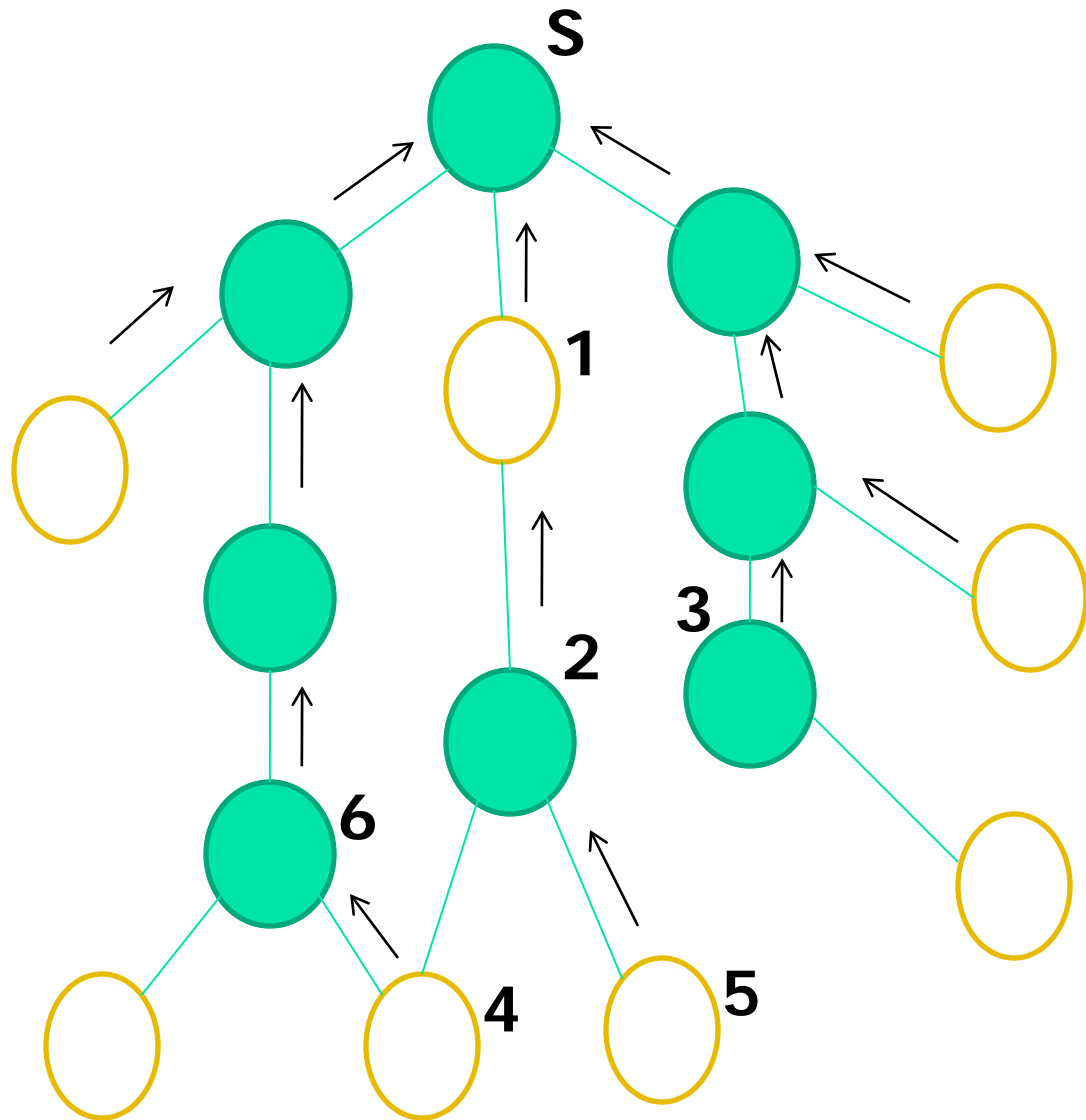


Illustration for CL parent pointer redirection recursively



Stage 1 :

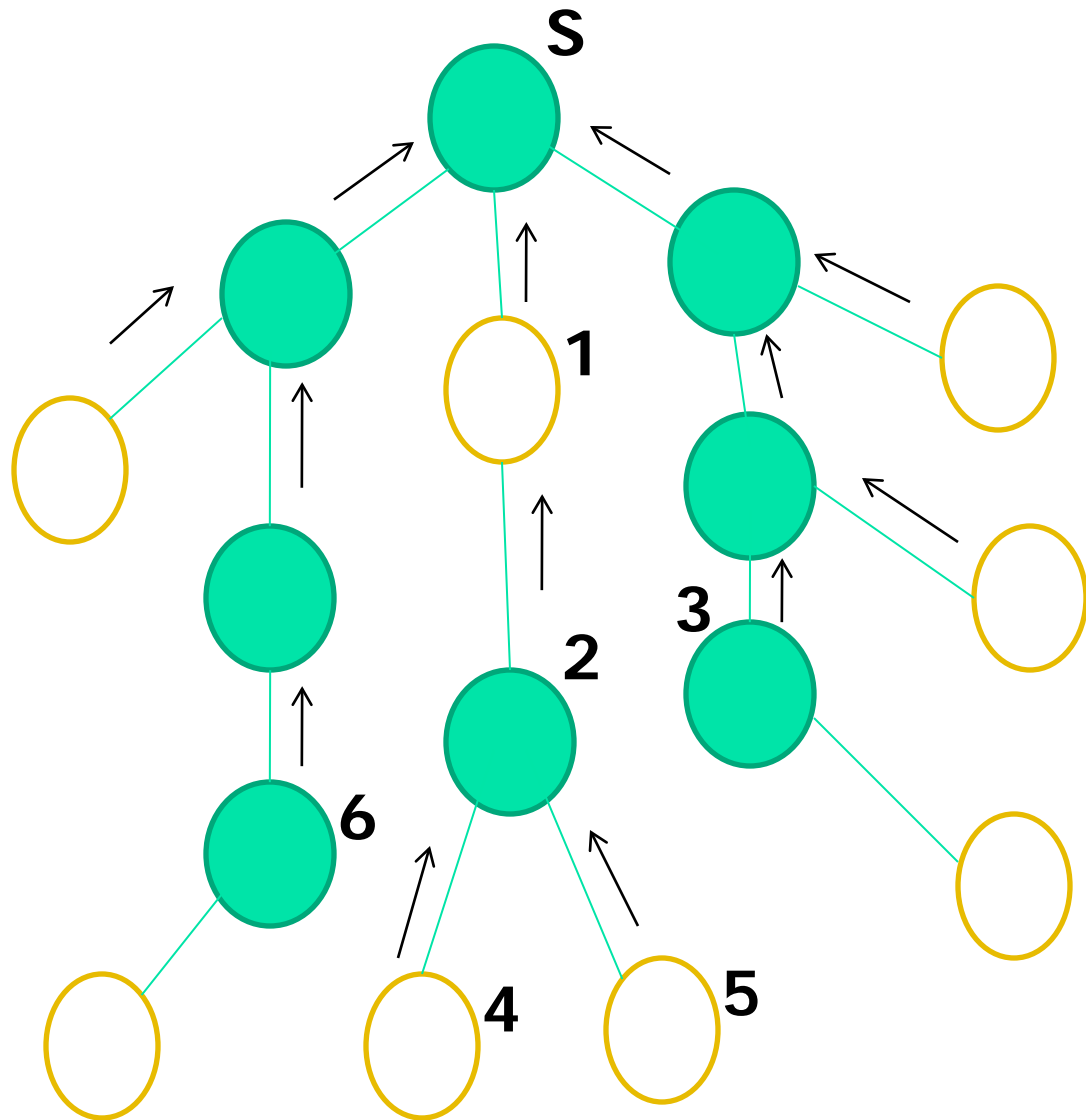
**Parent
Pointer
change from**

**2 - > 3 (Cost
= 4)**

to

**2 - > 1 (Cost
= 2)**

Illustration for CL parent pointer redirection recursively



Stage 2 :

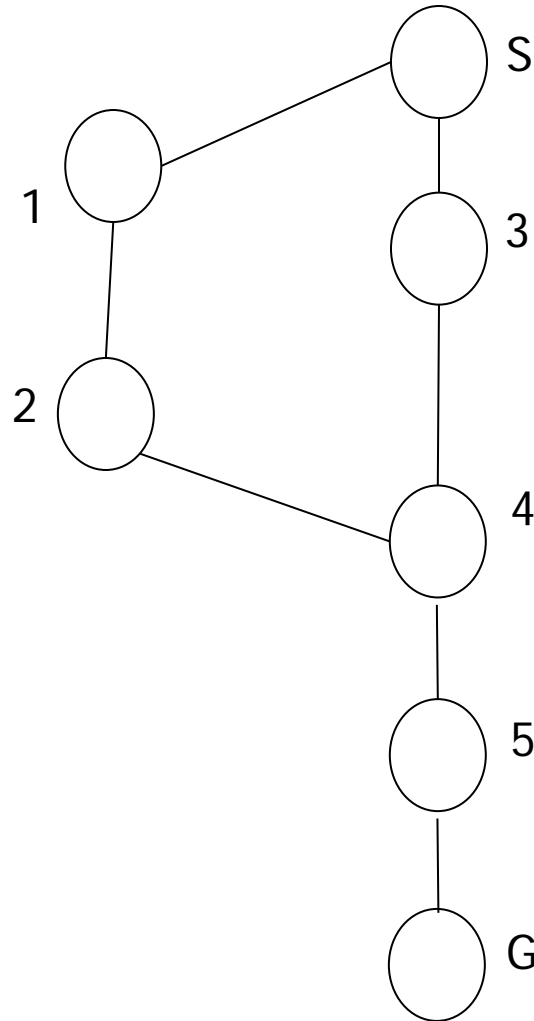
**Parent
Pointer
change from**

**4 - > 6 (Cost
= 4)**

to

**4 - > 2 (Cost
= 3)**

Another graph



Each arc cost
1 unit

$h=0$ for all nodes,
Except 3, which is
3, i.e., $h(3)=3$

3 violates MR
 $h(3)=3$
 $h(4)=0$

Sequence of expansions:
S-1-2-4-...

Definition of monotonicity

- A heuristic $h(p)$ is said to satisfy the monotone restriction, if for all ' p ',
 $h(p) \leq h(p_c) + cost(p, p_c)$, where ' p_c ' is the child of ' p '.

Theorem

- If monotone restriction (also called triangular inequality) is satisfied, then for nodes in the closed list, redirection of parent pointer is not necessary. In other words, if any node ' n ' is chosen for expansion from the open list, then $g(n)=g^*(n)$, where $g(n)$ is the cost of the path from the start node ' s ' to ' n ' at that point of the search when ' n ' is chosen, and $g^*(n)$ is the cost of the optimal path from ' s ' to ' n '

Grounding the Monotone Restriction

7	3	
1	2	4
8	5	6

n



7	3	4
1	2	
8	5	6

n'

1	2	3
4	5	6
7	8	

G

$h(n)$ -: number of displaced tiles

Is $h(n)$ monotone ?

$$h(n) = 8$$

$$h(n') = 8$$

$$C(n, n') = 1$$

Hence monotone

Monotonicity of # of Displaced Tile Heuristic

- $h(n) \leq h(n') + c(n, n')$
- Any move changes $h(n)$ by at most 1
- $c = 1$
- Hence, $h(\text{parent}) \leq h(\text{child}) + 1$
- If the empty cell is also included in the cost, then h need not be monotone (try!)

Monotonicity of Manhattan Distance Heuristic (1/2)

- *Manhattan distance* = *X-dist* + *Y-dist* from the target position
- Refer to the diagram in the first slide:
- $h_{mn}(n) = 1 + 1 + 1 + 2 + 1 + 1 + 2 + 1 = 10$
- $h_{mn}(n') = 1 + 1 + 1 + 3 + 1 + 1 + 2 + 1 = 11$
- $Cost = 1$
- Again, $h(n) \leq h(n') + c(n, n')$

Monotonicity of Manhattan Distance Heuristic (2/2)

- Any move can either increase the h value or decrease it by **at most 1**.
- Cost again is 1.
- Hence, this heuristic also satisfies Monotone Restriction
- If empty cell is also included in the cost then manhattan distance does not satisfy monotone restriction (try!)
- Apply this heuristic for Missionaries and Cannibals problem

Relationship between Monotonicity and Admissibility

- Observation:

Monotone Restriction \rightarrow Admissibility
but not vice-versa

- Statement: *If $h(n_i) \leq h(n_j) + c(n_i, n_j)$
for all i, j
then $h(n_i) \leq h^*(n_i)$ for all i*

Proof of Monotonicity \rightarrow admissibility

Let us consider the following as the optimal path starting with a node $n = n_1 - n_2 - n_3 \dots n_i - \dots n_m = g_i$

Observe that

$$h^*(n) = c(n_1, n_2) + c(n_2, n_3) + \dots + c(n_{m-1}, g_i)$$

Since the path given above is the optimal path from n to g_i

Now,

$$h(n_1) \leq h(n_2) + c(n_1, n_2) \text{ ----- Eq 1}$$

$$h(n_2) \leq h(n_3) + c(n_2, n_3) \text{ ----- Eq 2}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$h(n_{m-1}) = h(g_i) + c(n_{m-1}, g_i) \text{ ----- Eq (m-1)}$$

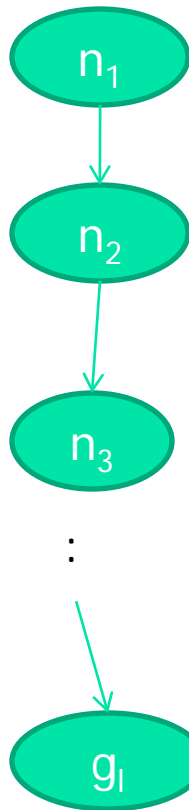
Adding Eq 1 to Eq (m-1) we get

$$h(n) \leq h(g_i) + h^*(n) = h^*(n)$$

Hence proved that $MR \rightarrow (h \leq h^*)$

Proof (continued...)

Counter example for vice-versa



$$h^*(n_1) = 3$$

$$h^*(n_2) = 2$$

$$h^*(n_3) = 1$$

$$\vdots$$

$$h^*(g_1) = 0$$

$$h(n_1) = 2.5$$

$$h(n_2) = 1.2$$

$$h(n_3) = 0.5$$

$$\vdots$$

$$h(g_1) = 0$$

$h < h^*$ everywhere but MR is not satisfied

Proof of MR leading to optimal path for every expanded node (1/2)

Let $S-N_1-N_2-N_3-N_4\ldots N_m\ldots N_k$ be an optimal path from S to N_k (all of which might or might not have been explored). Let N_m be the **last** node on this path which is on the open list, i.e., all the ancestors from S up to N_{m-1} are in the closed list.

For every node N_p on the optimal path,

$$\begin{aligned} g^*(N_p) + h(N_p) &\leq g^*(N_p) + C(N_p, N_{p+1}) + h(N_{p+1}), \text{ by monotone restriction} \\ g^*(N_p) + h(N_p) &\leq g^*(N_{p+1}) + h(N_{p+1}) \text{ on the optimal path} \\ g^*(N_m) + h(N_m) &\leq g^*(N_k) + h(N_k) \text{ by transitivity} \end{aligned}$$

Since all ancestors of N_m in the optimal path are in the closed list,

$$g(N_m) = g^*(N_m).$$

why do we require this statement

$$\Rightarrow f(N_m) = g(N_m) + h(N_m) = g^*(N_m) + h(N_m) \leq g^*(N_k) + h(N_k)$$

Proof of MR leading to optimal path for every expanded node (2/2)

Now if N_k is chosen in preference to N_m ,

$$\begin{aligned}f(N_k) &\leq f(N_m) \\g(N_k) + h(N_k) &\leq g(N_m) + h(N_m) \\&= g^*(N_m) + h(N_m) \\&\leq g^*(N_k) + h(N_k) \\g(N_k) &\leq g^*(N_k)\end{aligned}$$

But $g(N_k) \geq g^*(N_k)$, by definition

Hence $g(N_k) = g^*(N_k)$

This means that if N_k is chosen for expansion, the optimal path to this from S has already been found.

The key point here is that if MR is satisfied, nodes in an optimal path have to get expanded in the order of their distance from the start node.

TRY proving by induction on the length of optimal path

Monotonicity of $f(.)$ values

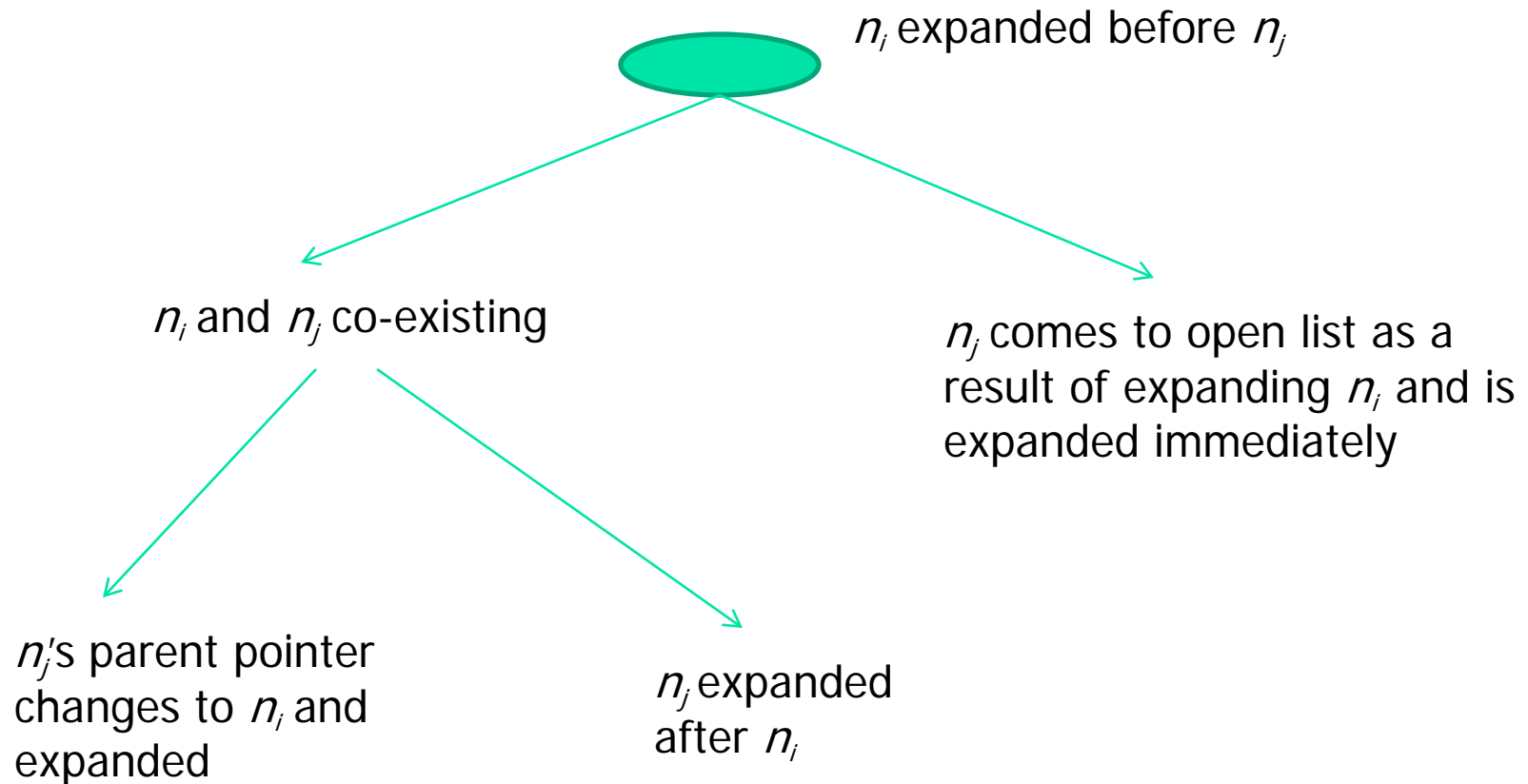
Statement:

f values of nodes expanded by A^*
increase monotonically, if h is
monotone.

Proof:

Suppose n_i and n_j are expanded with
temporal sequentiality, *i.e.*, n_j is
expanded after n_i

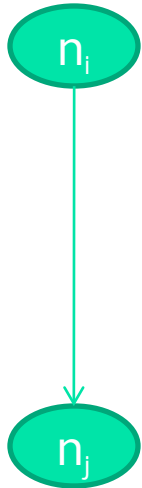
Proof (1/3)...



Proof (2/3)...

- All the previous cases are forms of the following two cases (think!)
- CASE 1:
 n_j was on open list when n_i was expanded
Hence, $f(n_i) \leq f(n_j)$ by property of A^*
- CASE 2:
 n_j comes to open list due to expansion of n_i

Proof (3/3)...



Case 2:

$$f(n_i) = g(n_i) + h(n_i) \quad (\text{Defn of } \hat{f})$$

$$f(n_j) = g(n_j) + h(n_j) \quad (\text{Defn of } \hat{f})$$

$$f(n_i) = g(n_i) + h(n_i) = g^*(n_i) + h(n_i) \quad \text{---Eq 1}$$

(since n_i is picked for expansion n_i is on optimal path)

With the similar argument for n_j we can write the following:

$$f(n_j) = g(n_j) + h(n_j) = g^*(n_j) + h(n_j) \quad \text{---Eq 2}$$

Also,

$h(n_i) \leq h(n_j) + c(n_i, n_j) \quad \text{---Eq 3 (Parent- child relation)}$

$$g^*(n_j) = g^*(n_i) + c(n_i, n_j) \quad \text{---Eq 4 (both nodes on optimal path)}$$

From Eq 1, 2, 3 and 4

$$f(n_i) \leq f(n_j)$$

Hence proved.

Better way to understand monotonicity of $f()$

- Let $f(n_1), f(n_2), f(n_3), f(n_4) \dots f(n_{k-1}), f(n_k)$ be the f values of k expanded nodes.
- The relationship between two consecutive expansions $f(n_i)$ and $f(n_{i+1})$ nodes always remains the same, *i.e.*, $f(n_i) \leq f(n_{i+1})$
- This is because
 - $f(n_i) = g(n_i) + h(n_i)$ and
 - $g(n_i) = g^*(n_i)$ since n_i is an expanded node (proved theorem) and this value cannot change
 - $h(n_i)$ value also cannot change Hence nothing after n_{i+1} 's expansion can change the above relationship.

Monotonicity of $f()$

$f(n_1), f(n_2), f(n_3), \dots, f(n_i), f(n_{i+1}), \dots, f(n_k)$

Sequence of expansion of $n_1, n_2, n_3 \dots n_i \dots n_k$

f values increase monotonically

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

Consider two successive expansions - $> n_i, n_{i+1}$

Case 1:

n_i & n_{i+1} Co-existing in OL

n_i precedes n_{i+1}

By definition of A^*

$$f(n_i) \leq f(n_{i+1})$$

Monotonicity of $f()$

Case 2:

$ni+1$ came to OL because of expanding ni and $ni+1$ is expanded

$$\begin{aligned} f(ni) &= g(ni) + h(ni) \\ &\leq g(ni) + c(ni, ni+1) + h(ni+1) \\ &= g(ni) + h(ni+1) \\ &= f(ni+1) \end{aligned}$$

Case 3:

$ni+1$ becomes child of ni after expanding ni and $ni+1$ is expanded. Same as case 2.

A list of AI Search Algorithms

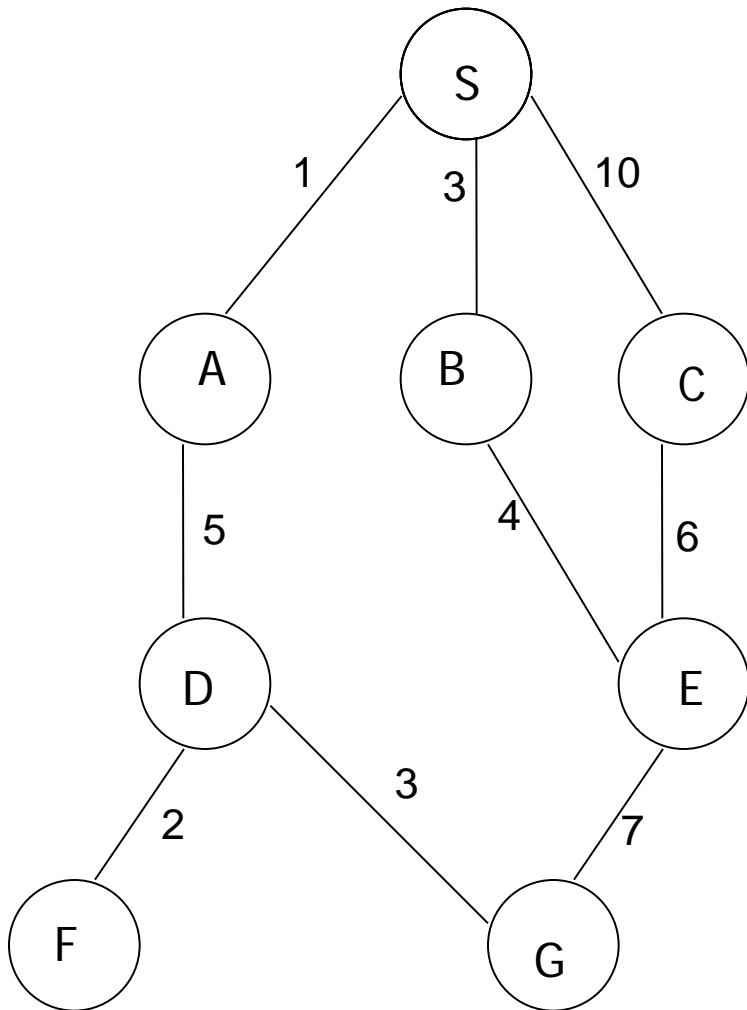
- A^*
 - AO^*
 - IDA^* (Iterative Deepening)
- Minimax Search on Game Trees
- Viterbi Search on Probabilistic FSA
- Hill Climbing
- Simulated Annealing
- Gradient Descent
- Stack Based Search
- Genetic Algorithms
- Memetic Algorithms

CS561

Additional slides from Lecture 1

6 Nov 2017

Graph $G = (V, E)$



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

$g(n)$ = Cost of optimal path from source to node 'n'

$$h(n) = 0$$

1) Open List : $S^{(\emptyset, 0)}$

Closed list : \emptyset

2) OL : $A^{(S,1)}, B^{(S,3)}, C^{(S,10)}$

CL : S

3) OL : $B^{(S,3)}, C^{(S,10)}, D^{(A,6)}$

CL : S, A

4) OL : $C^{(S,10)}, D^{(A,6)}, E^{(B,7)}$

CL: S, A, B

5) OL : $C^{(S,10)}, E^{(B,7)}, G^{(D, 9)}$

CL : S, A, B , D

6) OL : $C^{(S,10)}, F^{(D,8)}, G^{(D, 9)}$

CL : S, A, B, D, E

7) OL : $C^{(S,10)}, G^{(D,9)}$

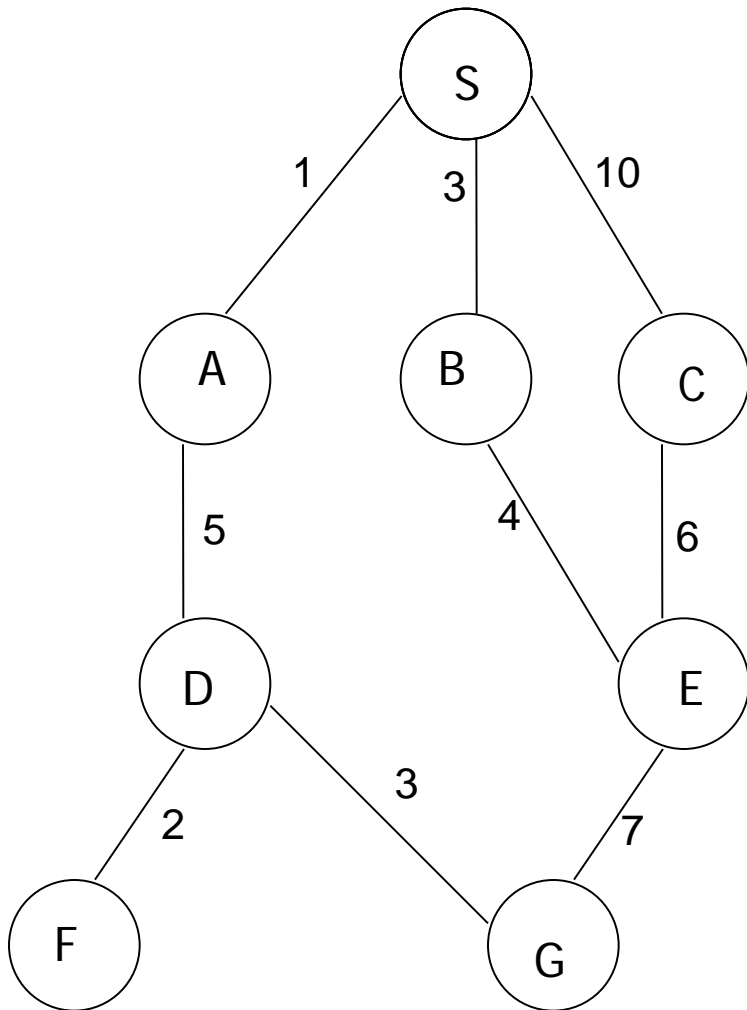
CL : S, A, B, D, E, F

8) OL : $C^{(S,10)}$

CL : S, A, B, D, E, F, G

Termination with success

Graph $G = (V, E)$



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

$g(n)$ = Cost of optimal path from source to node 'n'

$h(n)$ = Heuristic knowledge

Let

$$h(S) = 9,$$

$$h(A) = 8$$

$$h(B) = 11,$$

$$h(C) = 13$$

$$h(D) = 3,$$

$$h(E) = 7$$

$$h(F) = \text{infinity},$$

$$h(G) = 0$$

1) Open List : $S (\emptyset, 0, f=9)$

Closed list : \emptyset

2) OL : $A^{(S,1,f=9)}, B^{(S,3,f=14)}, C^{(S,10,f=23)}$

CL : S

3) OL : $B^{(S,3,f=14)}, C^{(S,10,f=23)}, D^{(A,6,f=9)}$

CL : S, A

4) OL : $B^{(S,3,f=14)}, C^{(S,10,f=23)}, G^{(D,9,f=9)}, F^{(D,8,f=infinity)}$

CL : S, A, D

5) OL : $B^{(S,3,f=14)}, C^{(S,10,f=23)}, F^{(D,8,f=infinity)}$

CL : S, A, D, G

Termination with success