Introduction to Artificial Intelligence: AO* and Hill Climbing algorithms

Dr. Tapas Kumar Mishra

Assistant Professor
Department of Computer Science and Engineering
SRM University
Amaravati-522502, Andhra Pradesh, India
Email: tapaskumar.m [at] srmap [dot] edu [dot] in

Heuristic Function

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$h_1(S) =$?? 6
$h_2(S) =$?? $4+0+3+3+1+0+2+1 = 14$

Heuristic Function

- A typical solution is about **20 steps**, although this of course varies depending on the **initial state**.
- The **branching factor** is about **3** (when the **empty tile** is in the **middle**, there are **four possible moves**; when it is in a **corner** there are **two**; and when it is along an **edge** there are **three**).
- An exhaustive search to depth **20** would look at about $3^{20} = 3.5 \times 10^9$ states.
- There are only $9! = 362{,}880$ different arrangements of 9 squares.
- Manhattan/City block distance: $|x_1 - x_2| + |y_1 - y_2|$

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 dominates h_1 and is better for search

Typical search costs:

$d = 14$ IDS = 3,473,941 nodes
A*(h_1) = 539 nodes
A*(h_2) = 113 nodes
$d = 24$ IDS $\approx$ 54,000,000,000 nodes
A*(h_1) = 39,135 nodes
A*(h_2) = 1,641 nodes

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

Relaxed problems

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the 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

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

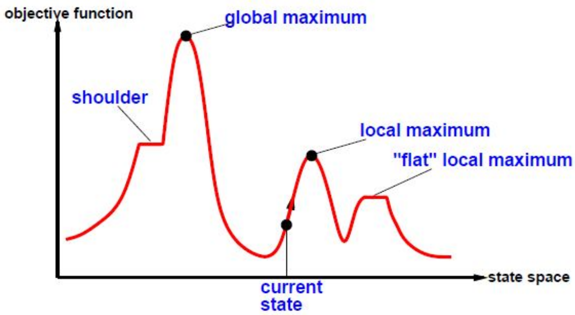
Local Search Algorithms and Optimization Problems

- **Local search algorithms** operate using a single **current state** (rather than **multiple paths**) and generally **move only to neighbors** of that state.
- Typically, the **paths** followed by the search are **not retained**.
- Although local search algorithms are **not systematic**, they have two key advantages:
  1. They use **very little memory**-usually a constant amount
  2. They can often **find reasonable solutions** in large or infinite (continuous) state spaces for which **systematic algorithms are unsuitable**.

Local Search Algorithms and Optimization Problems

- Local search algorithms are useful for solving pure **optimization problems**, in which the aim is to find the best state according to an **objective function**.
- A landscape has both "**location**" (defined by the **state**) and "**elevation**" (defined by the value of the **heuristic cost function** or **objective function**).
  - If elevation corresponds to **cost**, then the aim is to find the **lowest valley-a global minimum**
  - If elevation corresponds to an **objective function**, then the aim is to find the **highest peak**-a **global maximum**.

Hill-Climbing Search

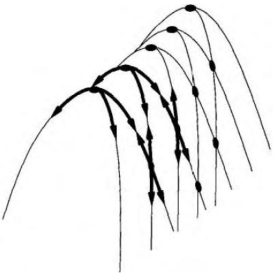


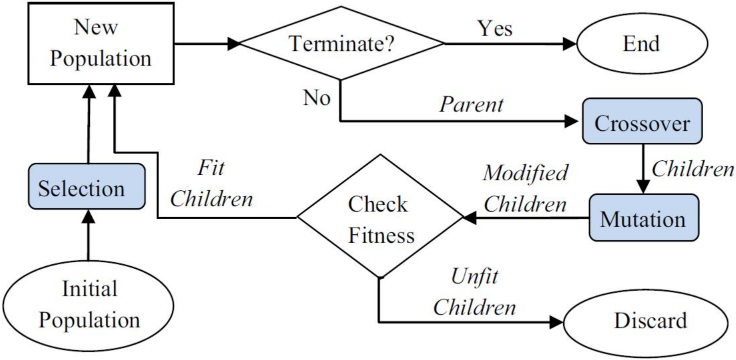
- It is simply a loop that **continually** moves in the direction of **increasing value**-that is, **uphill**.
- It terminates when it reaches a "**peak**" where **no neighbor** has a **higher value**.
- Hill-climbing algorithms typically choose randomly among the set of best successors, if there is more than one.

Hill-Climbing Search

Hill climbing often gets stuck for the following reasons

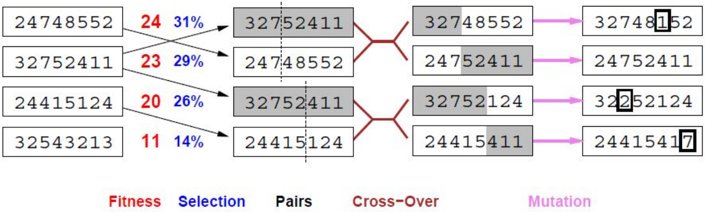
1. **Local maxima:** a **local maximum** is a **peak** that is **higher** than each of its **neighboring states**, but **lower** than the **global maximum**.
2. **Ridges:** Ridges result in **a sequence of local maxima** that is very difficult for **greedy algorithms** to **navigate**.
3. **Plateaux:** A plateau is an area of the state space landscape where the **evaluation function is flat**. It can be a **flat local maximum**, from which no uphill exit exists, or a **shoulder**, from which it is possible to make progress.



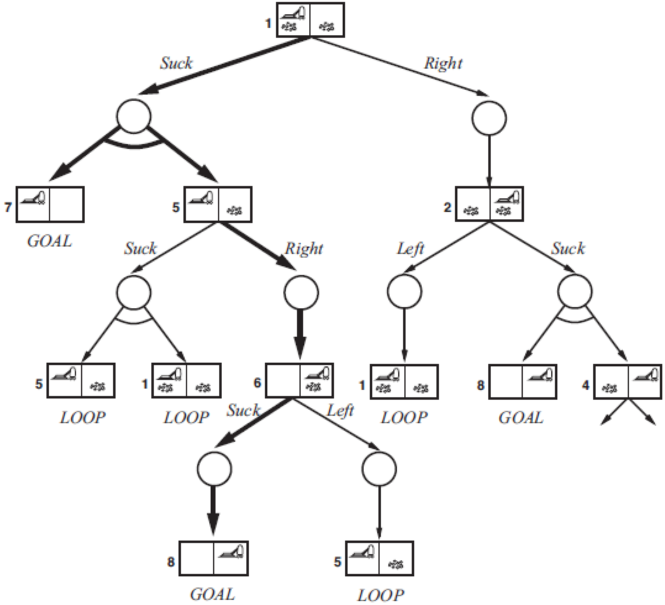

New Population
Terminate?
Yes
End
No
Parent
Crossover
Children
Mutation
Modified Children
Check Fitness
Fit Children
Selection
Initial Population
Unfit Children
Discard

Genetic algorithms

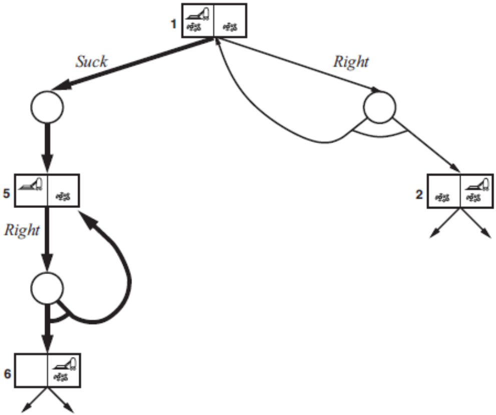
= stochastic local beam search + generate successors from **pairs** of states



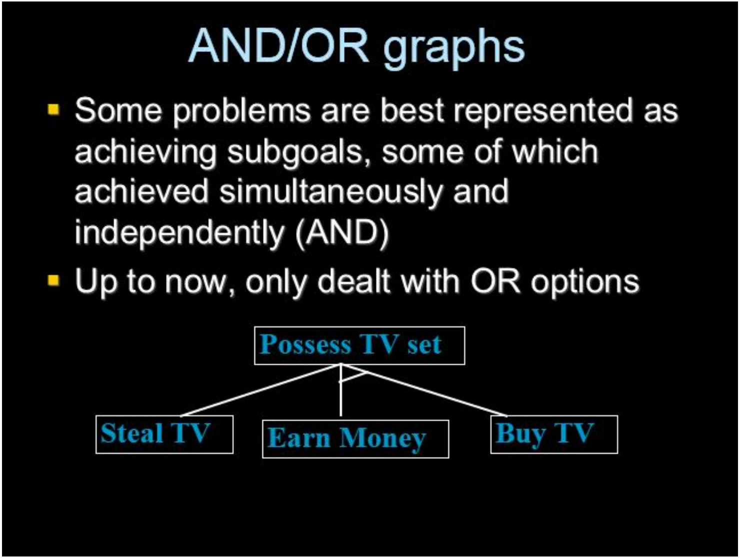
AND OR Search



AND OR Search



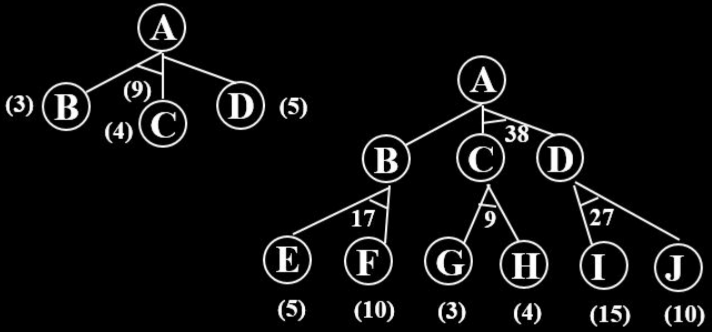
AO* Search


AND/OR graphs
Some problems are best represented as achieving subgoals, some of which achieved simultaneously and independently (AND)
Up to now, only dealt with OR options
Possess TV set
Steal TV
Earn Money
Buy TV

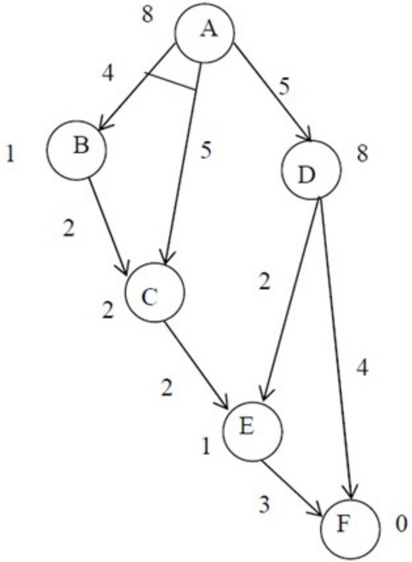
AO* Search

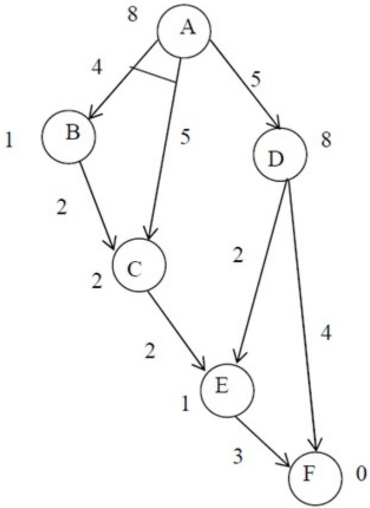
AND/OR search

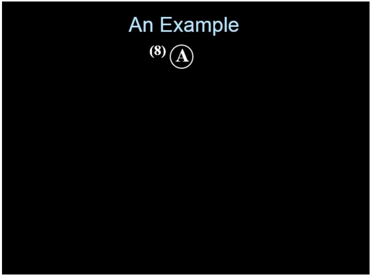
- We must examine several nodes simultaneously when choosing the next move

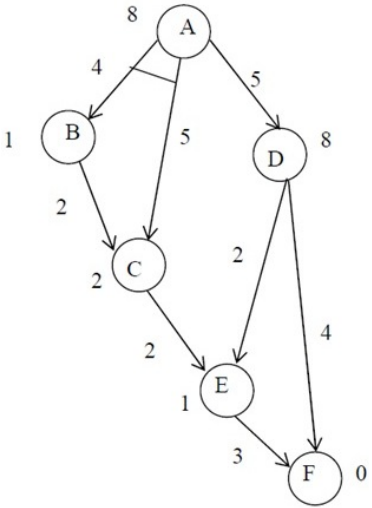


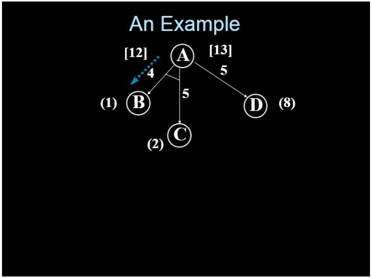
AO* Search

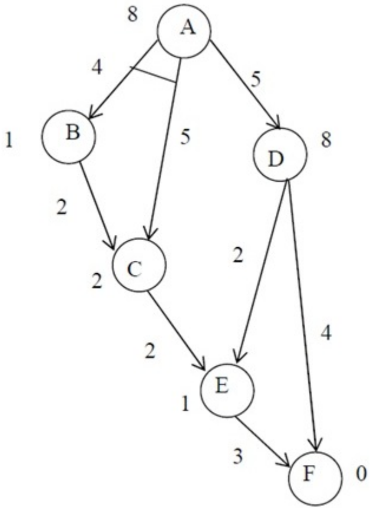


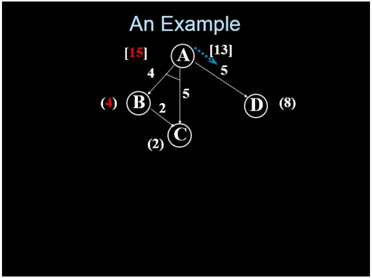

8
A
4
5
B
1
5
D
8
2
C
2
2
2
4
E
1
3
F
0


An Example
(8) A

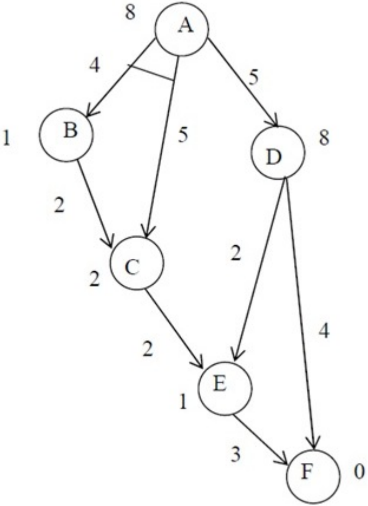

A
B
C
D
E
F
8
4
5
5
1
8
2
2
2
2
4
1
3
0

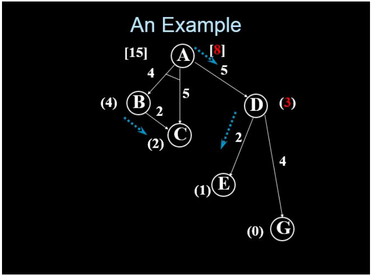

An Example
[12]
[13]
A
4
5
5
(1)
B
D
(8)
(2)
C

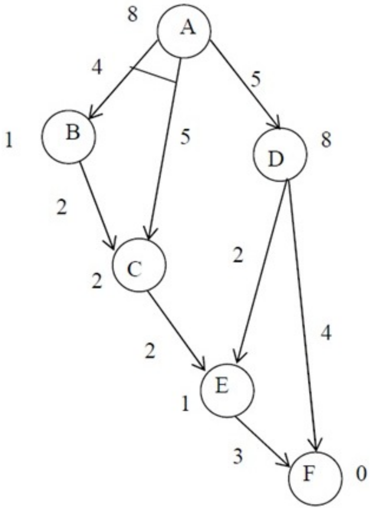

8
A
4
5
1
B
5
D
8
2
C
2
2
2
4
E
1
3
F
0


An Example
[15]
A
[13]
4
5
5
(4)
B
2
D
(8)
C
(2)

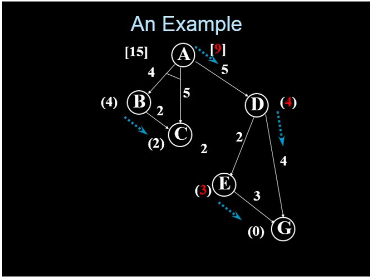
AO* Search





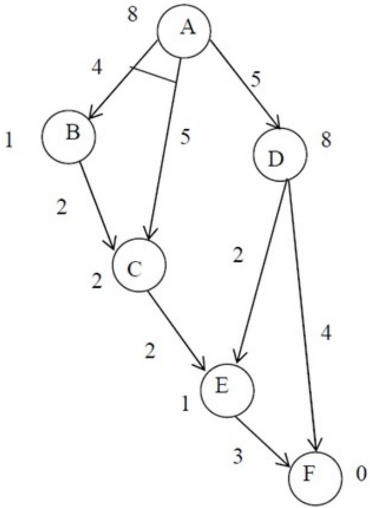


A
8
4
5
B
1
5
D
8
2
2
C
2
2
4
E
1
3
F
0

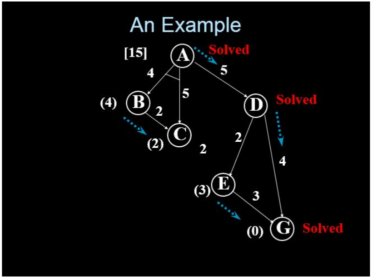


An Example
[15]
A
[9]
4
5
5
(4)
B
2
D
(4)
C
(2)
2
2
4
(3)
E
3
(0)
G

AO* Search



8
A
4
5
1
B
5
D
8
2
2
C
2
4
2
E
1
3
F
0



An Example
[15]
A
Solved
4
5
(4)
B
5
2
D
Solved
C
(2)
2
2
4
(3)
E
3
(0)
G
Solved

AO* Search

1. Place the starting node s on open.
2. Using the search tree constructed thus far, compute the most promising solution tree T_0.
3. Select a node n that is both on open and a part of T_0. Remove n from open and place it on closed.
4. If n is a terminal goal node, label n as solved. If the solution of n results in any of n's ancestors being solved, label all the ancestors as solved. If the start node s is solved, exit with success where T_0 is the solution tree. Remove from open all nodes with a solved ancestor.
5. If n is not a solvable node (operators cannot be applied), label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. If any of n's ancestors become unsolvable because n is, label them unsolvable as well. Remove from open all nodes with unsolvable ancestors.

AO* Search

6. Otherwise, expand node n generating all of its successors. For each such successor node that contains more than one subproblem, generate their successors to give individual subproblems. Attach to each newly generated node a back pointer to its predecessor. Compute the cost estimate h^* for each newly generated node and place all such nodes that do not yet have descendants on open. Next, recompute the values of h^* at n and each ancestor of n.
7. Return to step 2.

Thank You!