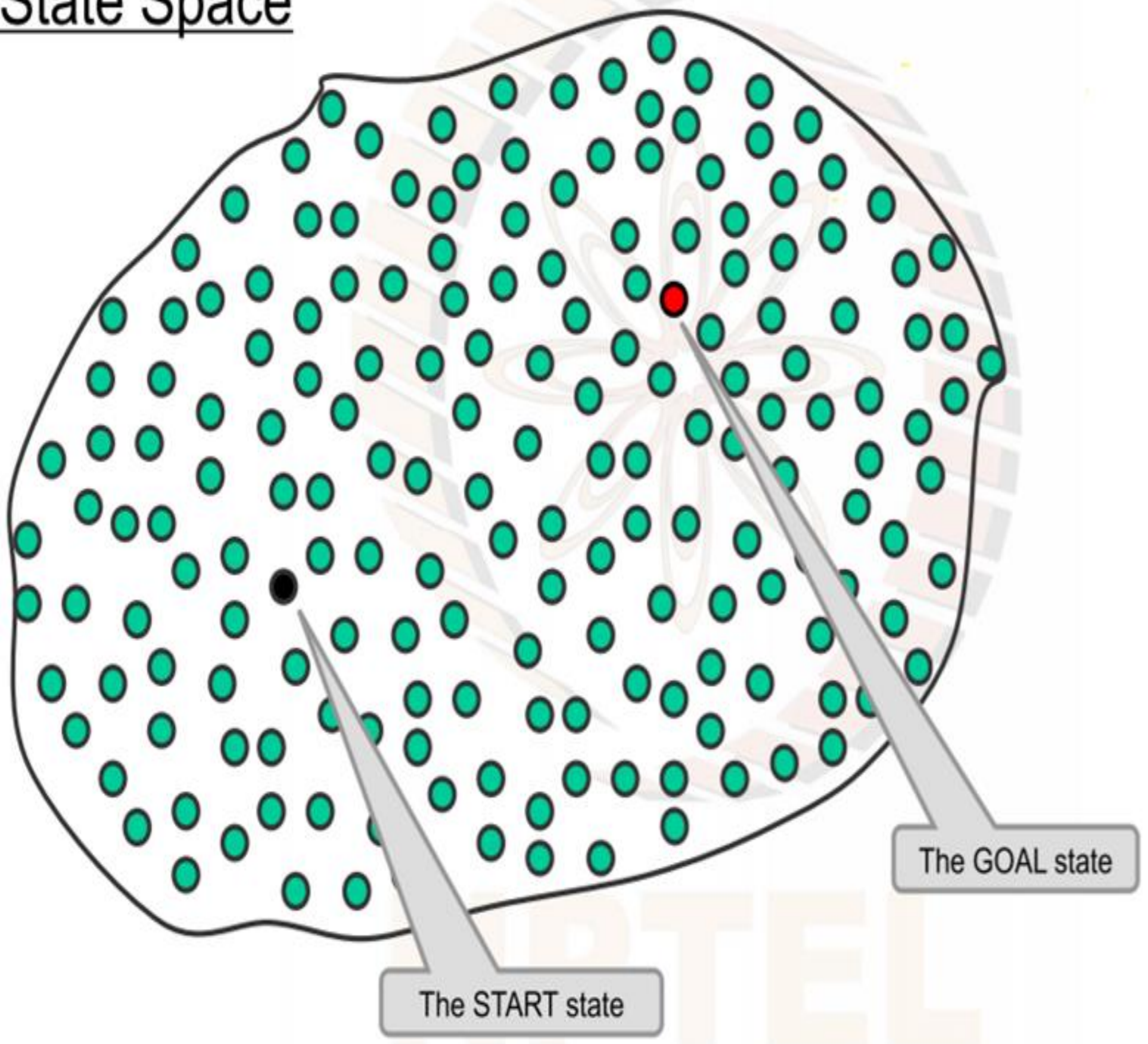
AI-Search Strategies

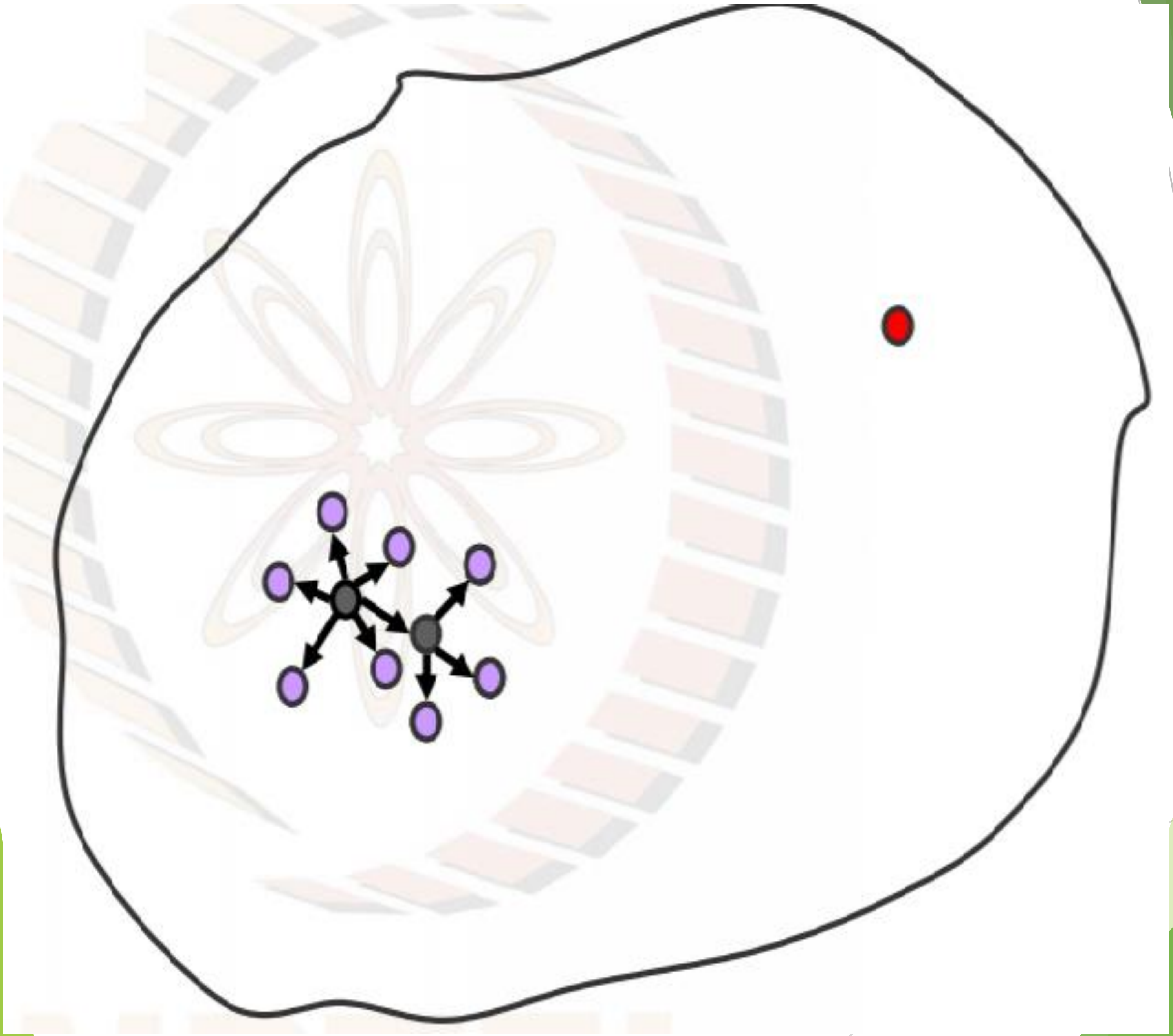
Dr. Dhanya K.M.
Associate Professor (IT)
Government Engineering College
Palakkad

Search Strategies

- Depth-first search
- Breadth-first search
- Generate-and-Test
- Hill-climbing
- Best-first search
- Problem reduction
- Constraint satisfaction

The State Space





Generate and Test

- Algorithm : Generate-and-Test

1. Generate a possible solution.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise , return to step1.

Hill Climbing

- **Simple hill climbing** : It only evaluates the neighbour node state at a time and *selects the first one which optimizes current cost* and set it as a current state
- **Steepest-ascent hill climbing :** It examines all the neighbouring nodes of the current state and *selects one neighbour node which is closest to the goal state*
- **Stochastic hill climbing :** It *selects one neighbour node at random* and evaluate it as a current state or examine another state

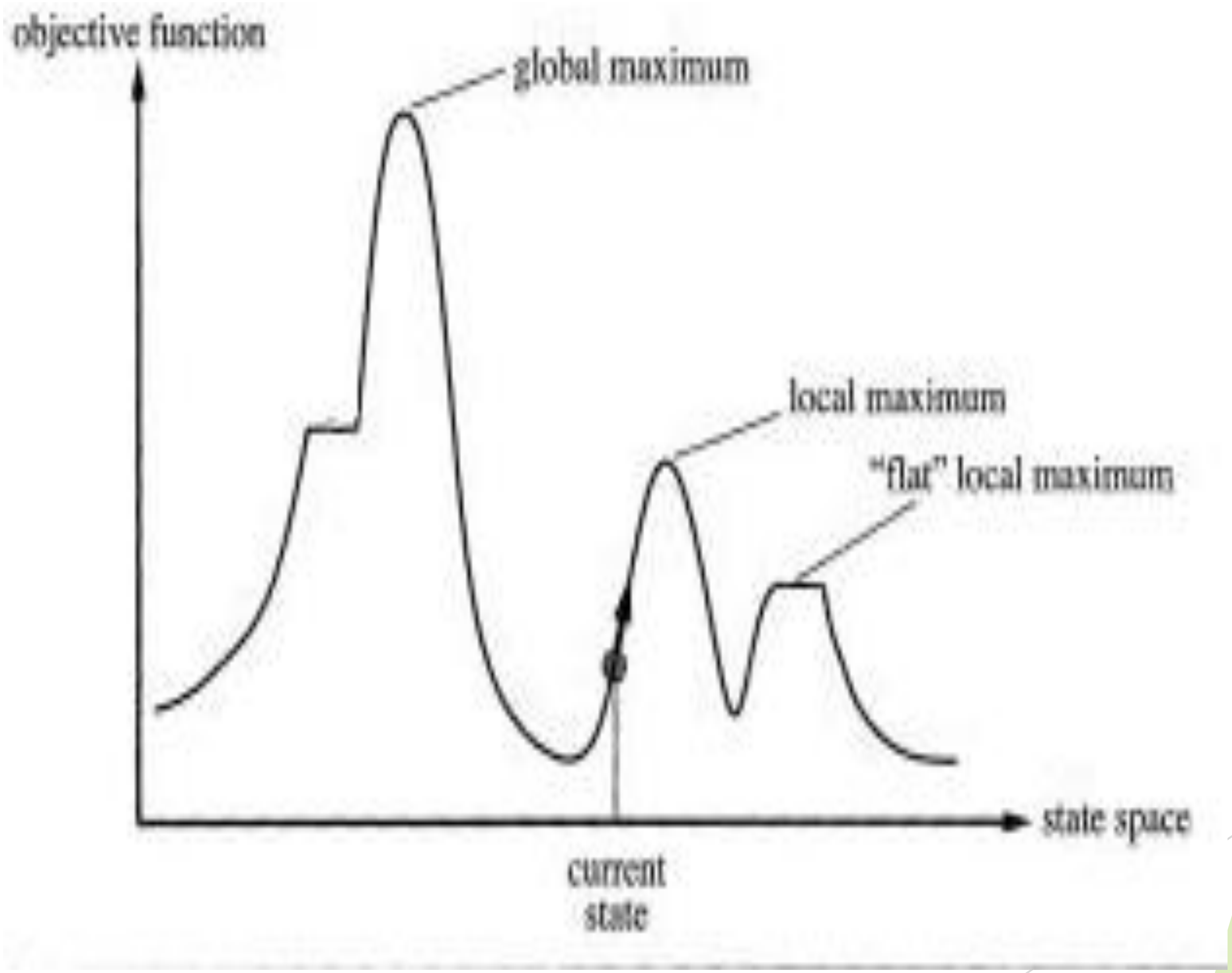
Simple Hill Climbing Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   a. Select an operator that has not yet been applied to the current state and apply it to produce a new state
   b. Evaluate the new state
      i. If it is a goal state, then return it and quit
      ii. If it is not a goal state but it is better than the current state, then make it the current state
      iii. If it is not better than the current state, then continue in the loop
3. Exit

Steepest Ascent Hill Climbing

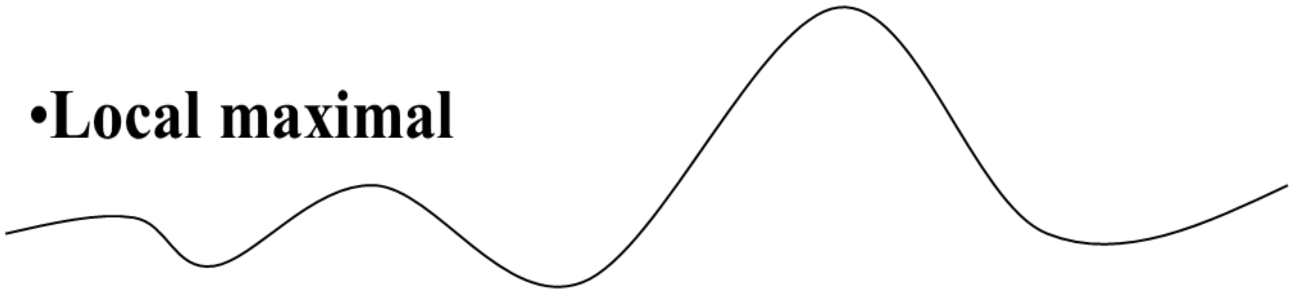
1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let it be a state such that any possible successor of the current state will be better than *SUCC*
   b. For each operator that applies to the current state do:
      i. Apply the operator and generate a new state
      ii. Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to *SUCC*. If it is better, then set *SUCC* to this state. If it is not better, leave *SUCC* alone
   c. If the *SUCC* is better than current state, then set current state to *SUCC*
3. Exit

State Space Diagram-Hill Climbing

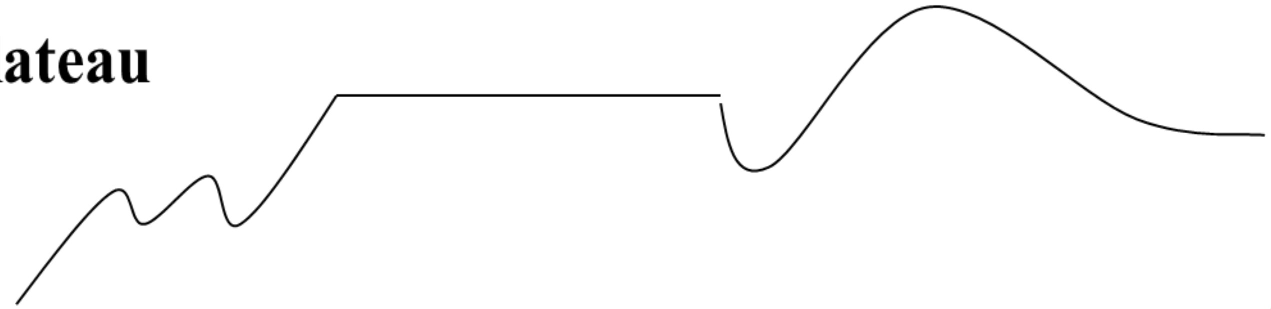


Hill Climbing Problems

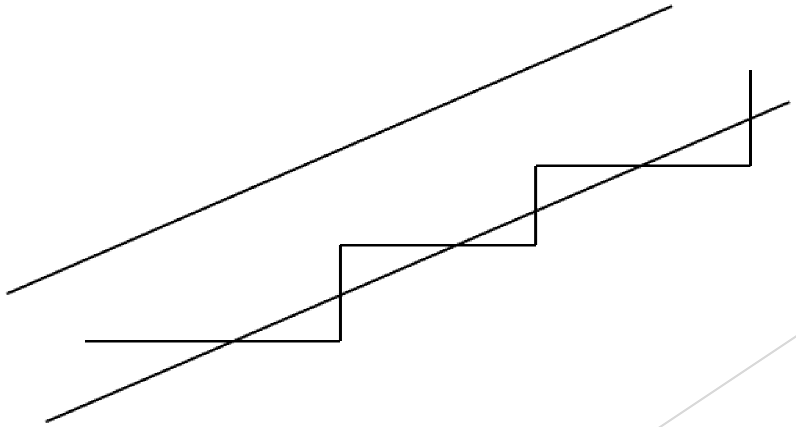
•**Local maximal**



•**Plateau**



•**Ridge**



Problems and Remedies of Hill Climbing

- Possible problems :
  1. Local optimal
  2. Plateau
  3. Ridge

- Possible solutions :
  1. Keep a list of plausible move and backtrack when a dead-end is met
  2. Make a big jump or move the same direction several times
  3. Try different directions (applying two or more rules) before test
- Hill climbing is basically a “local “ heuristics

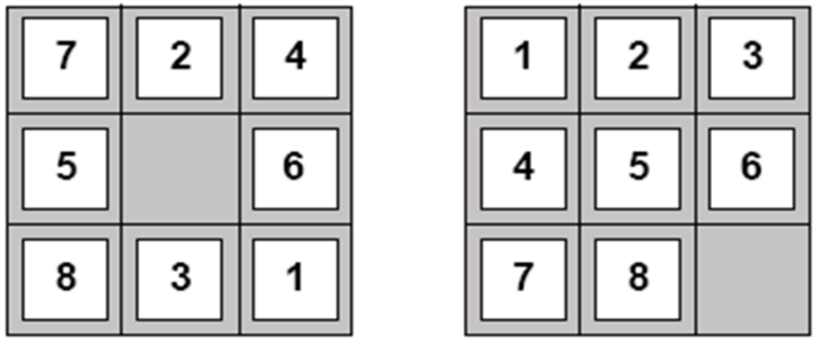
Best-First Search

- Use **heuristic function to choose a best move** out of several alternatives
- Keep exploring the best path until it turns less promising than a previous path
- Return to explore the previous path that has become most promising

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)



Start State   Goal State

$h_1(S) =$?? 7
$h_2(S) =$?? $4+0+3+3+1+0+2+1 = 14$

A* Search algorithm

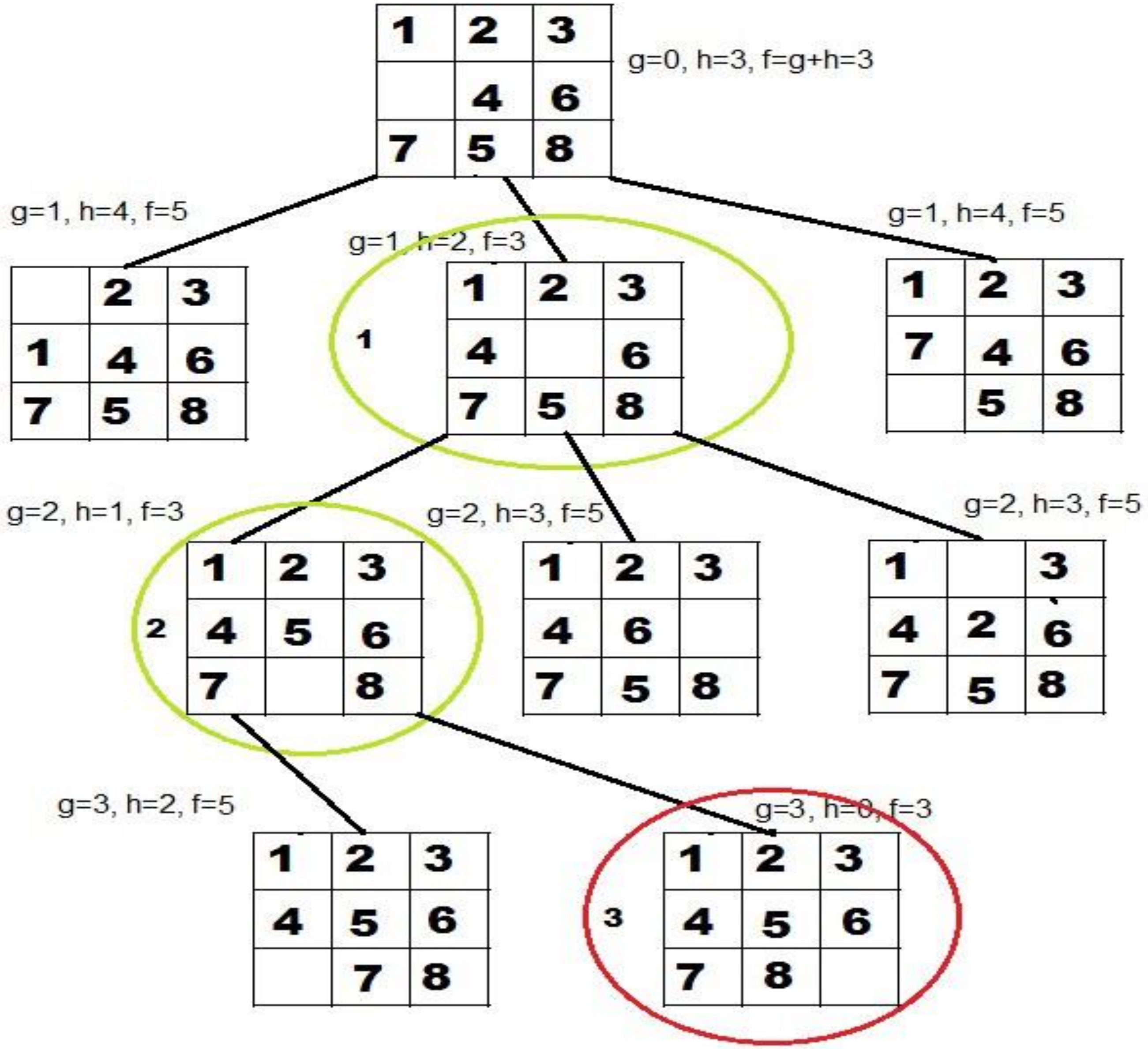
- Idea: avoid expanding paths that are already expensive

- The best search is based on the heuristic function for a given search node is

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

- g(n) is the cost from the starting node to the current node n
- h(n) is the estimate of the cost from the current node n to the goal node
- f(n) = estimated total cost of path through n to goal

A* on 8-Puzzle Problem



- Notation:
  - c(n , n’) is the cost of arc (n , n’)
  - g(n) is the cost of current path from start to node n in the search tree.
  - h(n) is the estimate of the cheapest cost of a path from n to a goal.
  - f(n) estimates the cheapest cost solution path that goes through n.
  - h*(n) is the true cheapest cost from n to a goal.
  - g*(n) is the true shortest path from the start s to n.

- If the heuristic function, h always underestimate the true cost (h(n) is smaller than h*(n)), then A* is guaranteed to find an optimal solution.

Admissibility of A*

- Whenever an A* algorithm terminates to find a solution and the solution is an optimal path solution from starting node s to goal node, then A* is admissible
- If h(n) underestimates h for all nodes ($h(n) \leq h^*(n)$), then A* is admissible
- If h(n) overestimates h*(n) for some node, then no guarantee the A* will find the optimal solution

A* Algorithm

- Input: a search graph problem with cost on the arcs
- Output: the minimal cost path from start node to a goal node.

1. Put the start node s on OPEN.
2. If OPEN is empty, exit with failure
3. Remove from OPEN and place on CLOSED a node n having minimum f.
4. If n is a goal node exit successfully with a solution path obtained by tracing back the pointers from n to s.
5. Otherwise, expand n generating its children and directing pointers from each child node to n.

   For every child node n' do

   - evaluate h(n') and compute f(n') = g(n') + h(n') = g(n)+c(n,n')+h(n')
   - If n' is already on OPEN or CLOSED compare its new f with the old f and attach the lowest f to n'.
   - put n' with its f value in the right order in OPEN
6. Go to step 2.

Performance Measures

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution?
- **Time Complexity**: How long does it take to find a solution?
- **Space Complexity**: How much memory is needed to perform the search?

Constraint Satisfaction Problem

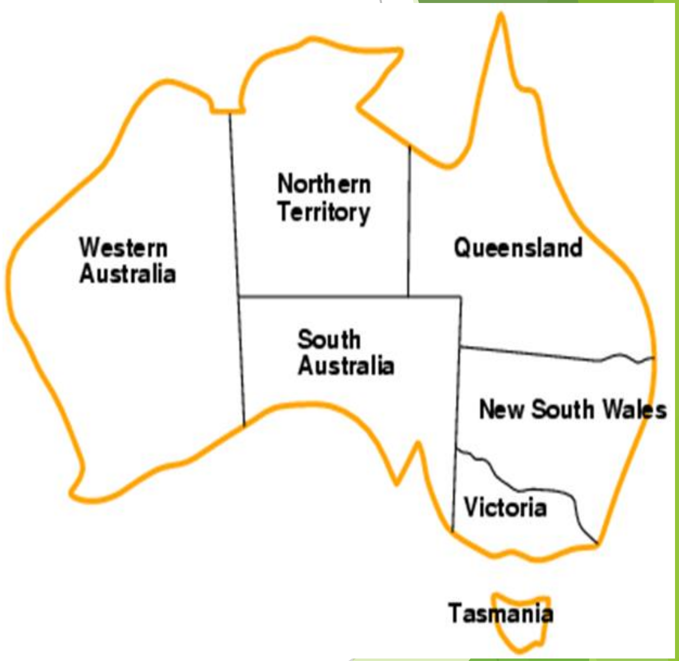
- State is defined by *variables* Xi with values from *domain* Di
- Goal test is *a set of constraints* specifying allowable combinations of values for subsets of variables

Eg: Map Colouring

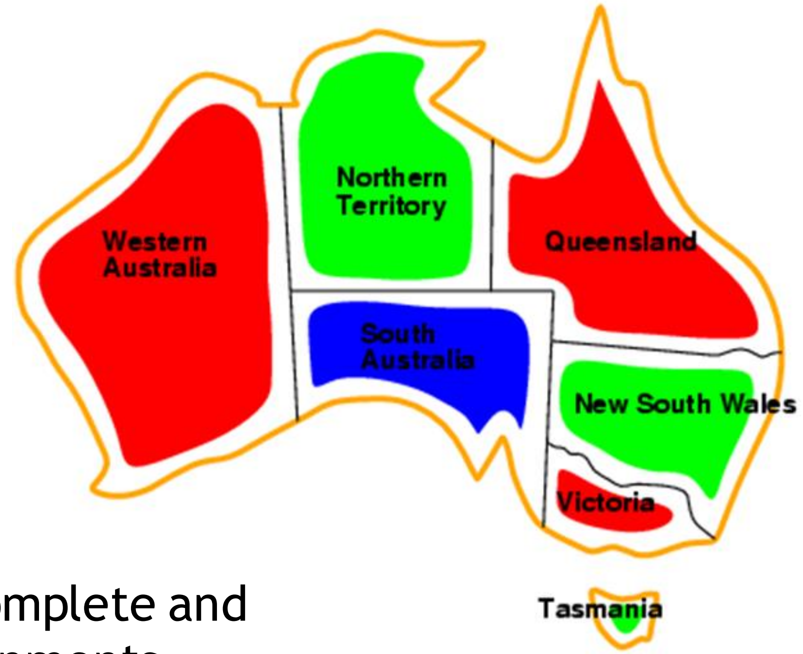
Variables Xi : WA, NT, Q, NSW, V, SA, T

Domains Di = {red,green,blue}

Constraints: Adjacent regions must have different colours



Map-Colouring



Solutions are complete and consistent assignments,

WA = red, NT = green,
Q = red, NSW = green,
V = red, SA = blue, T = green

Cryptarithmetic Problem-Example1

- **Constraints:**
  - No two letters have the same value
  - The sums of the digits must be as shown in the problem
- **Goal State:**
  - All letters have been assigned a digit in such a way that all the initial constraints are satisfied

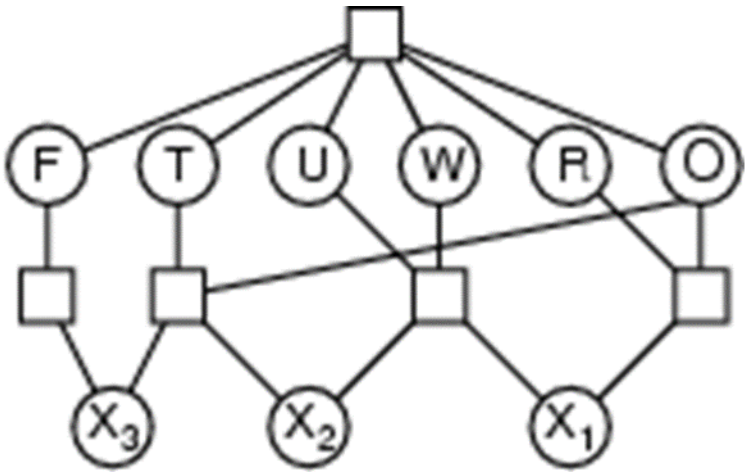
```
    S E N D
+   M O R E
-----------
  M O N E Y
```

Solution:

```
   9567
+  1085
--------
  10652
```

Cryptarithmetic Problem-Example2

```
    T W O
+   T W O
---------
  F O U R
```



- Variables: $F\ T\ U\ W\ R\ O$ $\qquad X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$ $\qquad \{0,1\}$
- Constraints: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

Example2- Solution

- According to Naoyuki Tamura, there are seven solutions to the problem:

- 938+938=1876
- 928+928=1856
- 867+867=1734
- 846+846=1692
- 836+836=1672
- 765+765=1530
- 734+734=1468

Constraint Satisfaction Algorithm

1. Propagate available constraints. To do this first set OPEN to set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty:

    a. Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.
    b. If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
    c. Remove OB from OPEN.

2. If the union of the constraints discovered above defines a solution, then quit and report the solution.

3. If the union of the constraints discovered above defines a contradiction, then return the failure.

4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this loop until a solution is found or all possible solutions have been eliminated:
    a. Select an object whose value is not yet determined and select a wayof strengthening the constraints on that object.
    b. Recursively invoke constraint satisfaction with the current set of constraints augmented by strengthening constraint just selected.

References

- Elaine Rich and Kevin Knight, “Artificial Intelligence”, Tata McGraw-Hill Publishing Company Ltd., New Delhi, Third Edition, ISBN: 13:978-0-07-008770-5, 2010
- Stuart Russell, Peter Norvig, “Artificial Intelligence- A modern approach”, Pearson Education Asia, Second Edition, ISBN:81-297-0041-7
- Amit Konar, Artificial Intelligence and Soft Computing, CRC Press.
- Dan W. Patterson, “Introduction to Artificial Intelligence and Expert Systems”, Prentice Hall India Ltd., New Delhi, 2009, ISBN: 81-203-0777-1.
- Rajendra Akerkar, Introduction to Artificial Intelligence, PHI Learning Pvt. Ltd., 2005,ISBN: 81-203- 2864-7

THANK YOU