Artificial Intelligence

Satvik Gupta

February 14, 2023

Intelligence: Ability to understand and react.

Allows:

1. React dynamically and quickly.
2. Recognize the relative importance of different elements of the situation.
3. Handling ambiguity, contradictions, incomplete and uncertain information.

AI

Artificial Intelligence (AI) refers to the development of abilities of natural intelligence to an extent ia machine.

Definitions

John McCarthy

The term "AI" was coined by John McCarthy (MIT, 1956). AI being a branch of computer science, is concerned with making computers behave like humans.

Rich and Knight

"AI is the study of how to make computers do things which at the moment people are better at doing" – Rich and Knight

Akerkar and Sajia

"AI is the branch of computer science that attempts to solve problems by mimicking the human thought process using heuristics and a symbolic, non-algorithmic approach" - Akerkar and Sajia

John Durkin

"AI is the field of study in computer science that pursues the goal of making a computer reason in a manner similar to humans" - John Durkin

Approaches

Thinking Humanly:

- Cognitive modeling approach
- This approach tries to solve a problem by seeing how human brains work and tries to incorporate the human problem-solving process into a machine by one of three ways:
  - Introspection (Observing one's own thoughts)
  - Psychological experiments (Observing someone else in action)
  - Brain imaging (Observing the brain in action)

Thinking Rationally:

- Laws of thought approach
- This approach is based on logic and the process of reasoning.
- It uses syllogisms which when provided patterns for argument structures always yield conclusions given correct premises.

Acting Humanly:

- Turing test approach
- In this the interrogator asks questions to both a human and machine. If he is unable to discriminate who is who, the machine passes the Turing test.
- To pass the Turing test, a machine must be equipped with techniques for natural language processing (NLP), knowledge representation, automated reasoning and machine learning

Acting Rationally:

- Rational agent approach
- A rational agent is one, so as to achieve the best outcome or the best expected outcome when there is uncertainty
- It is more generalized than the Laws of thought approach

Task Domains

1. Mundane tasks:
   - Perception
   - Communication (NLP)
   - Common sense reasoning
2. Formal Tasks:
   - Game Playing
   - Mathematical reasoning
3. Expert tasks:
   - Engineering tasks
   - Medical tasks

AI Techniques

1. Search based Techniques
   1.1. Brute Force Search (Uninformed)
   1.2. Informed Search (Heuristic)
2. Knowledge
3. Abstraction

Search techniques:

1. Search provides a way of solving problems for which no more direct approach is available. It also provides a framework into which any direct techniques that are available, can be embedded.
2. A search program finds the solution of a problem by trying various sequences of actions until a solution is found.

Advantages:

- Search techniques might be the best way when all other options are exhausted.
- The search process itself finds the sequence of actions to achieve the goal.

Disadvantages:

- In real world problems, the search space is so large that it is impossible or impractical to explore the entire search space.

Knowledge based Techniques:

The use of knowledge provides a way of solving complicated problems by manipulating the structures of concerned objects. Knowledge is indispensable but also voluminous and constantly changing (dynamic), hard to characterize accurately. Also, the organization of knowledge greatly impacts its usage and efficiency of the technique using it.

An AI technique, is a method that exploits knowledge that should be represented in such a way that:

- The knowledge captures generalization so that a separate representation of individual situations is not required.
- Important properties of situations are grouped together else the knowledge reduces to simply a large amount of data.
- It is understandable by the people involved in the project.
- It is easily modifiable to reflect changing situations.
- It is usable, even when information is not accurate or complete.
- It should be able to narrow down the range of search possibilities.
- It finds a way of separating important features and notifications from the unimportant ones, that would confuse any process.

AI Solutions

Two categories:

1. Weak Solutions/Weak AI
    - General search methods.
    - Not motivated by achieving human level performance.
    - Primary aim for these solutions is ***NOT*** to model how a human thinks.
    - They require more computations and less knowledge.
    - These methods aim to solve any problem.
    - Not very effective in specific tasks.
    - For e.g. A* search.
2. Strong solutions/Strong AI

    AKA Knowledge-based, or intensive solutions

    - Used for expert systems.
    - More knowledge, hence less computations.
    - Achieve better performance in specific tasks.
    - Can be guided by weak AI.

Phases of AI Solutions:

1. Search and Logic
2. Probability Based
3. Neural Networks

Problem Formulation

State - Information about environment. It is a general representation of a problem. Technically, the state should contain *all* information about its environment. While describing a problem, we generally only include all the information necessary to make a decision for the task at hand.

State Space - The set of all possible states. Each state represents a possible configuration of the problem. The problem can make a transition from one state to another by using one of the actions or operators available. An operator refers to some representation of an action. An operator usually includes information about what must be true in the world before the operator is applied, and how the world is changed after the operator is applied.

State Space Search

The state space search problem solving method is a way of defining a given problem as a problem of moving around in a state space, where each state corresponds to a legal position in the problem.

The problem solving progresses by starting at an initial state, using a set of rules to move from one state to another, and attempting to end up in a final state.

Types of State Views

1. Atomic View States are numbered to be differentiated, but we cannot view the details of each state, or the values of the variables in the state.
2. Propositional/Factored View We can view inside the state.
3. Relational View State objects are represented in how they relate to each other.
4. First Order View (Relational + Functions)

Question - **Consider a problem where we have 2 rooms and 1 vacuum cleaner. Each room may or may not have dirt in it. The vacuum cleaner can only be in one room at a time.How will this problem be represented in all the 4 views?**

R1 - Room1

R2 - Room2

D - Dirt

ND - No Dirt

V - has vacuum cleaner

Atomic View

1. (R1DV, R2D)
2. (R1D, R2DV)
3. (R1DV, R2ND)
4. (R1D, R2NDV)
5. (R2NDV,R1D)
6. (R2ND,R1DV)
7. (R1NDV,R2ND)
8. (R1ND,R2NDV)

Do rest later

Problem Solving

1. Define/Formulate the Problem.
2. Analyze
3. Task Knowledge
   - Isolation
   - Knowledge Representation
4. Choose an appropriate technique and apply it.

Steps to Define/Formulate a Problem

1. States
2. Initial State
3. Actions
4. Transition Model
5. Goal Test
6. Path Cost

Question Travel from City A to City B (formulate the problem using above 6 steps)

1. **States**

   Each state is a location between A and B.

2. **Initial State** - A

3. **Actions**

   - Choosing a mode of transport from A to some intermediate location, say C.

4. **Transition Model**

   - The state resulting from travelling from a previous location to the current location will have the current location as source, for the next segment of the journey.

5. **Goal Test**

   - Is current location same as specified destination location.

6. **Path Cost**

   - Path cost can be either the amount of fare expended, time taken, or distance traveled.

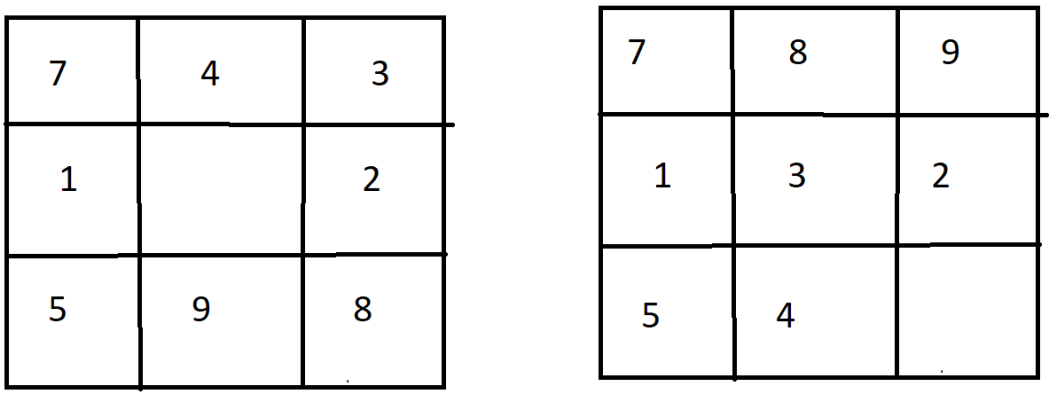
Chess Board

1. States 32 pieces, each of which can have 64 position. Each state will be defined by a 32-tuple containing the X and Y co-ordinate of each piece, and which player's turn it is (B/W). If a piece has been moved off the board, it's coordinates will be (-1,-1).
2. Initial State Each piece will be at the starting position defined in chess. Starting player is white.
3. Actions Each piece has certain rules for movement. Each player takes turns in moving, and can only move one piece at a time, following the legal rules for that piece. Pieces off the board cannot be moved.
4. Transition Model Each movement will result in a new state defined by the new positions of the pieces, and the turn will change. Movement for each piece will be according to the rules for that piece.
5. Goal Test
   - Checkmate from either side. (Win/Lose)
   - No possible moves from either side. (Draw)
6. Path cost can be either the time taken, the number of pieces lost, or the number of moves taken until the game ends.

***Question* 8 Puzzle Problem**

Consists of a 3x3 board with 8 numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The objective is to reach a specified goal state.

For example:



(Initial State on the left, Desired Final State on the Right)

1. **States**

   A state description specifies the location of each of the 8 tiles and the blank space in one of the nine squares.

2. **Initial State**

   Any possible configuration can be designated as initial state.

3. **Actions**

   Actions can be defined as *movements of the* ***blank*** space left,right,up or down. Different subsets of these are possible depending on where the blank space is.

4. **Transition Model**

   Given a state and action, this returns the resulting state. For example, if we apply the action *left* to the start state, the resulting state will have 1 and the blank space switched.

5. **Goal Test**

   In this we compare whether the current configuration matches the goal configuration.

6. **Path Cost**

   Let us assume each move costs 1. Path cost will be number of steps.

Production System

A production system provides a structure for the search process in AI system It is a program or system that facilitates in describing the search process and performing the search process

Production syntax

$$if \rightarrow then$$

$$antecedent \rightarrow consequence$$

Control Strategies

Control Strategies decide which rule to apply next during the search process It needs to deal with certain situations like:

- absence of any rules exactly matching the given fact
- more than one rule matching the given fact

  In this case a conflict resolution strategy must be used to decide on a single matching rule

A good control strategy saves time.

Characteristics of a Good Control Strategy

1. It should cause movement which takes us nearer to our goal. Each application of an if-else rule should take us closer to our goal.
2. It should be systematic.

Water Jug Problem

You are given 2 jugs 4L and 3L without any marking and a tap to fill the 2 containers with water.

The Goal is to get exactly 2L of water in the 4L water jug.

1. Formulate it as a space search Problem
2. Enlist all the possible rules
3. Find a possible solution
4. Apply BFS and DFS

X - water in 4L jug

Y - water in 3L jug

Initial State $(x, y) \rightarrow (0, 0)$

A*NS**WER***

(0,0)

(4,0)

(1,3)

(1,0)

(0,1)

(4,1)

(2,3)

Rules

$$(x, y) \rightarrow (0, 3)$$

$$(x, y) \rightarrow (4, 0)$$

$$(x, y) \rightarrow (x, 0)$$

$$(x, y) \rightarrow (0, y)$$

if (x+y>=4 and y>0) :

$$(x, y) \rightarrow (4, y - (4 - x))$$

if (x+y>=3 and x>0):

$$(x, y) \rightarrow (x - (3 - y), 3)$$

if(x+y<=4 and y>0):

$$(x, y) \rightarrow (x, y, 0)$$

if(x+y<=3 and x>0):

$$(x, y) \rightarrow (0, x + y)$$

Formulating with 6 steps

States

Both jug can be empty or may have certain amount of water in them. Represented as:

$(0...4, 0...3)$

Initial States

Both jugs are empty

$(0, 0)$

Actions

- Fill up either container.
- Empty either container.
- Pour contents of one container into another.

Transitions (same as rules mentioned above.)

Goal Test

We will have reached our goal if 4L jug has 2L water.

I.e, if $state == (2, x)$, where $x \in [0, 3]$

Path Cost

Path cost can be:

- Number of moves.
- Amount of Water taken from tap.

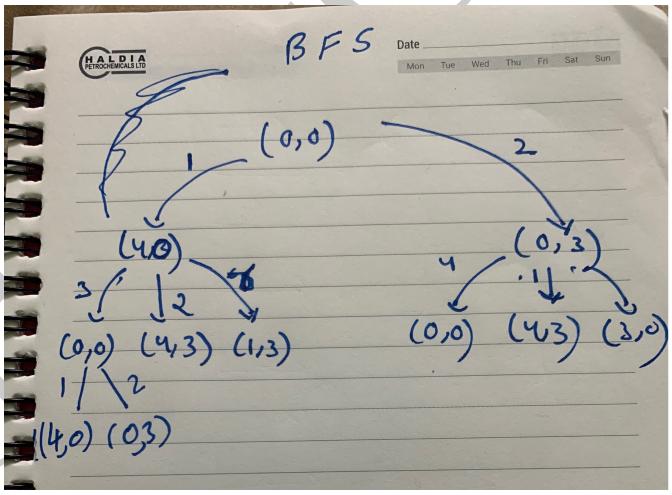


Figure 1: State Transition Tree for Water Jug Problem

Characteristics of Production System

Monotonic Production System

In this, the application of a rule never prevents the later application of another rule, that could also have been applied at the time the first rule was selected.

Non-monotonic production systems are those in which the above is not true.

Partially Commutative Prod System

If the application of a particular sequence of rules transforms a state $X \to Y$, then any allowable permutation of these rules also performs the transformation $X \to Y$

Commutative Prod System

Both Monotonic and Partially Commutative

	Monotonic	Non-Monotonic
Partially Comm.	Theorem Planning	Robot Navigation
Not Partially Comm.	Chemical Synthesis	Chess

Characteristics of a Problem

1. **Decomposability**
2. **Ignoring/Undoing steps**:
   - Ignorable (Theorem Proving)
   - Recoverable (8 Puzzle)
   - Irrecoverable (Chess)
3. **Predictability**:

   If we're able to predict whether we will reach our goal state, and how. In 8 Puzzle we can predict. In chess we cannot, since goal depends on the opponent's actions also.
4. **Nature of Solution**:

   Is there a single goal state or multiple goal states are acceptable?
5. **Solution Type**:
   - State
   - Path
6. **Role of Knowledge**:

   Is knowledge necessary (as in NLP), or is it just used to speed things up (as in chess engine, where we can brute force everything but knowledge helps optimization by a lot.)
7. **Interactive**
8. **Problem Classification**

Problems in Search Methodology

1. Search direction:
    - Forward
    - Backward
2. Search Strategy
3. Node Representation
4. Search Process Representation:
    - Tree
    - Graph

Heuristic Search Techniques

They're also called **informed search techniques**. These are useful when direct techniques are impractical or impossible. These methods are guided by direct methods. These are general purpose techniques independent of problem domain.

When applied to particular problems, they apply domain-specific knowledge. These are also called weak methods. They provide a framework into which domain specific knowledge can be placed. They form the core of most AI systems.

Generate and Test Technique

1. Generate a possible solution/path.
2. See if it is a goal state/ solution.
    - If yes, return and quit.
    - If not, backtrack to step 1.

Hill Climbing

Simple HC

1. Generate an initial state.
    - If it is a goal state, return and quit.
    - If not, call it as current state.
2. Apply an operator (that hasn't yet been applied) to current state and generate a new state.
3. If new state is a goal state, return and quit.

   Otherwise,compare new state with current state.
    - If new state is better than current state, make new state as current state.
    - If new state is not better than current state, go back to step 2.

Here, *better* is evaluated by a heuristic function. It can be cost, or number of steps, distance, etc. depending on the problem.

Steepest Ascent HC

Chooses the best successor as current state. It generates all successors from the current state.

Simple HC chooses the first better new state as current state.

1. Check if initial state is goal state.
    - If yes, return and quit.
    - If no, make initial state as current state.
2. Loop until a solution is found, or until a complete iteration produces no change in the current state (CS):

1. Let $SUCC$ be the worst possible successor to CS (i.e, no successor to CS will be worse than $SUCC$)
2. For each operator applicable to CS
    1. Apply the operator to CS and generate the new state (NS).
    2. Evaluate the NS. If it is a goal state, return and quit. If not, compare it with $SUCC$. If NS is better than $SUCC$, then set $SUCC = NS$. Otherwise, let $SUCC$ remain as it is.
    3. If $SUCC$ is better than CS, set $CS = SUCC$.

Issues with HC

Both types of HC may get stuck. Either algorithm may terminate, not by finding a goal state, but by getting to a state from which no better states can be generated. This can happen in 3 cases:

- **Local Maxima**

  It is a state which is better than all its neighbors, but there me a far off state that is better than it. All immediate moves from the local maxima seem to make things worse.

  Local maxima often occur almost within sight of a solution. In this case, they are called foothills.

  Solution:

  Backtrack and try another path.

- **Plateau**

  It is a flat area of search space, with all neighbors having the same value. All the immediate next moves seem same. No best move exists.

  Solution:

  Make a big jump in some random direction to try to get a new section of the search space. We can apply a simple one-move rule several times in the same direction to accomplish this.

- **Ridge**

  It is a special kind of local maxima. It is an area in the search space that is higher than surrounding areas, but it itself has a slope (that we wish to climb).

  Solution:

  Apply two or more rules before doing the test.

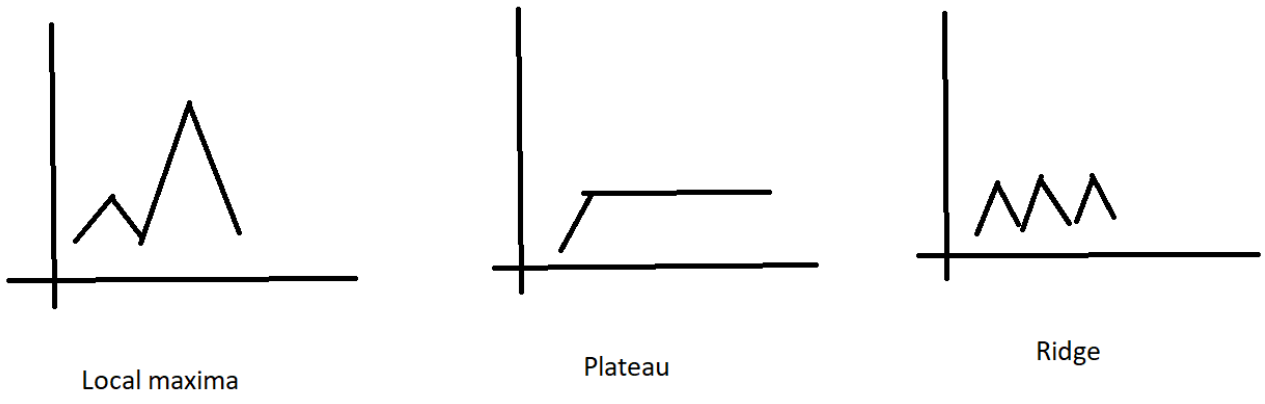


Figure 2: Issues in Hill Climbing Techniques

Simulated Annealing

Heuristic Function is referred to as the objective function. We try to minimize the value of the objective function .

Annealing is the process of melting metals, and gradually cooling them until a solid state is reached. Goal is to find a minimal energy final state.

Objective function is the energy level. There is some probability that a transition to a higher energy state will occur (even though, generally physical substances move towards lower energy configurations). The probability of such a transition to a higher energy state is

$$p = e^{\Delta E/kT}$$

Where ΔE is change in Energy level T is temperature and k is the Boltzmann constant.

Mapping this to AI and HC, we use the formula

$$p' = e^{\Delta E/T}$$

where T is the objective function.

A schedule for T must be maintained, called as annealing schedule. It should have - an initial value of T - a criteria to reduce T - amount by which T must be reduced - When to quit

Simulated annealing is generally used with problems having a large search space.

The annealing schedule must specify:

1. Initial Value of T.
2. By what amount to reduce T
3. Criteria for reducing T
4. When to Quit

Algorithm for Simulated Annealing

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. Initialize **BEST-SO-FAR** to the current state.
3. Initialize T according to the annealing schedule.
4. Loop until a solution is found or until there are no new operators left to be applied in the current state.
    1. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
    2. Evaluate the new state. Compute

    $$\Delta E = (value\ of\ current) - (value\ of\ new\ state)$$

    - If the new state is a goal state, then return it and quit.
    - If it is not a goal state but is better than the current state, then make it the current state. Also set **BEST-SO-FAR** to this new state.
    - If it is not better than the current state, then make it the current state with probability p' as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range $[0, 1]$. If the number is less than p', the move is accepted, otherwise do nothing

    3. Revise T as per annealing schedule.
5. Return **BEST-SO-FAR** as the answer.

Best First Search && A*

Factors used in determining complexity of search algorithms in graphs

1. b - branching factor

   The maximum number of successors that can be generated from a single node.

2. m - max length

   The length of the longest path from root to leaf.

3. d - depth of solution

   The level at which goal state exists.

Heuristics - Best First Search And A*

What is Heuristic Search?

- Heuristic is a technique which makes our search algorithm more efficient.
- Some heuristics help to guide a search process without sacrificing any claim to completeness and some sacrificing it.
- Heuristic is a problem specific knowledge that decreases expected search efforts.
- It is a technique which sometimes works but not always.
- Heuristic search algorithm uses information about the problem to help directing the path through the search space.

These searches uses some functions that estimate the cost from the current state to the goal presuming that such function is efficient.

A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number.

The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available.

- Generally heuristic incorporates domain knowledge to improve efficiency over blind search.
- In AI heuristic has a general meaning and also a more specialized technical meaning.
- Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case.
- Heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing.
- This technique is useful in solving tough problems which could not be solved in any other way, those whose solutions take an infinite time to compute.

Heuristic Strategies

Simple Techniques

- Hill Climbing
  - Simple
  - Steepest Ascent
  - Problems
  - Simulated Annealing

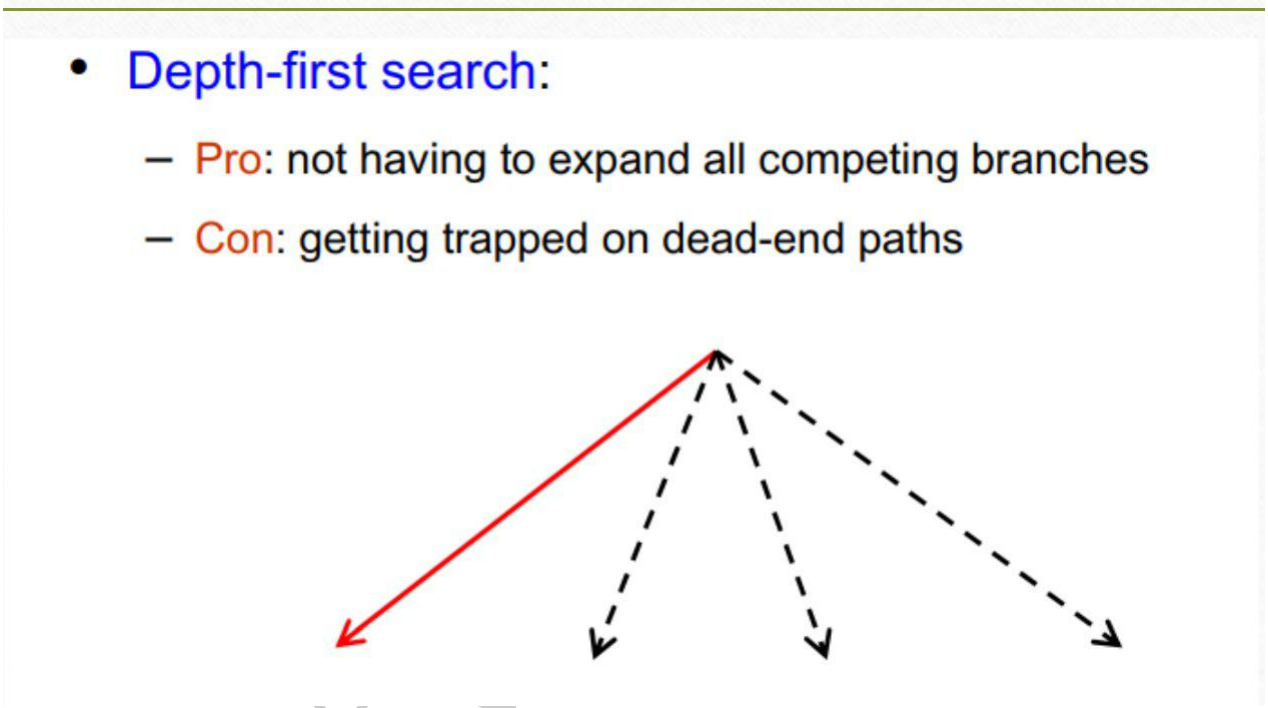
Improved Techniques

- Best First Search
  - OR graphs
  - A* algorithm
- Problem Reduction
  - AND-OR graphs
  - AO* algorithm
- Constraint Satisfaction
- Means-ends Analysis

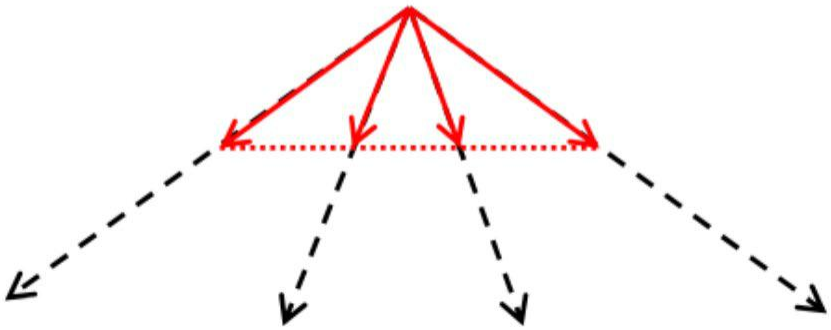
BEST FIRST SEARCH

Introduction

- Combines the advantages of breadth first search and depth first search.
- DFS is good as it allows a solution to be found without all competing branches having to be expanded at all levels.
- BFS is good because it does not get trapped on dead-end paths.
- In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function.
- The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore.



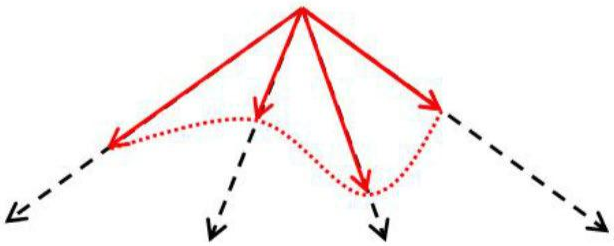
- **Breadth-first search:**
  - Pro: not getting trapped on dead-end paths
  - Con: having to expand all competing branches



- The best first search allows us to switch between paths thus gaining the benefit of both approaches.
- **At each step the most promising node is chosen.**
- If one of the nodes chosen generates nodes that are less promising it is possible to choose another at the same level and in effect the search changes from depth to breadth.
- If on analysis these are no better than this previously unexpanded node and branch is not forgotten and the search method reverts to the

Best-First Search

⇒Combining the two is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one.



- Best first search is an instance of graph search algorithm in which a node is selected for expansion based on evaluation function $f(n)$.
- Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation

measures distance to the goal.

- Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values.
- This search algorithm serves as combination of depth first and breadth first search algorithm.
- Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable

---

OR Graphs

- Best first search works on the type of problems that can be represented in the form of OR graphs.
- An OR graph is a graph whose various branches represents an alternative problem solving path.
- We can choose either one path OR another path OR another path to reach our goal.

Implementation

Done by using two lists of nodes:

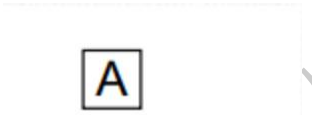
- **OPEN**

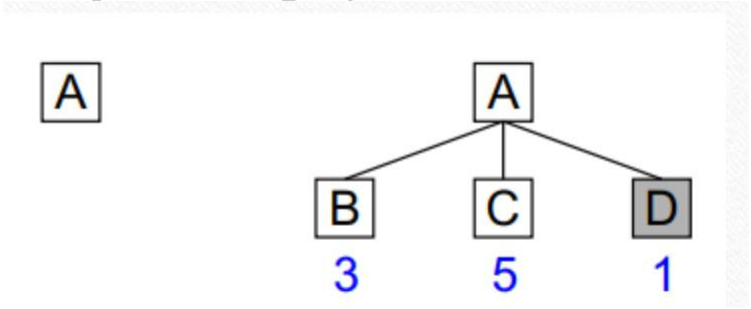
  OPEN is a priority queue of nodes that have been evaluated by the heuristic function but which have not yet been expanded into successors. The most promising nodes are at the front.

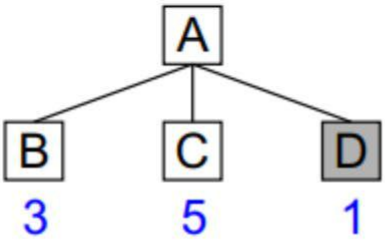
- **CLOSED**

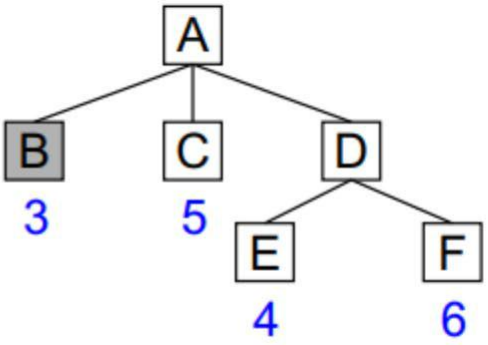
  CLOSED are nodes that have already been generated and these nodes must be stored because a graph is being used in preference to a tree.

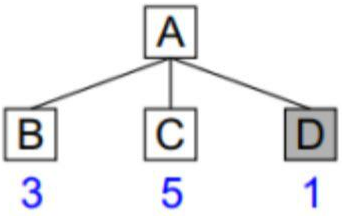
Example

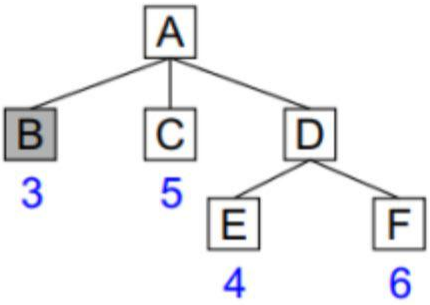


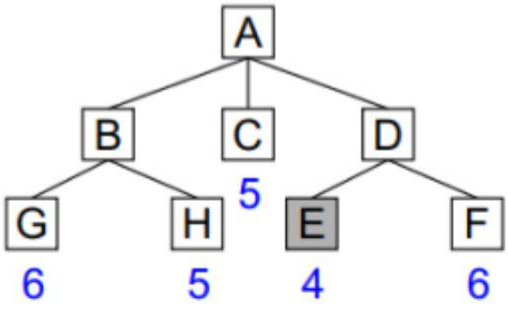


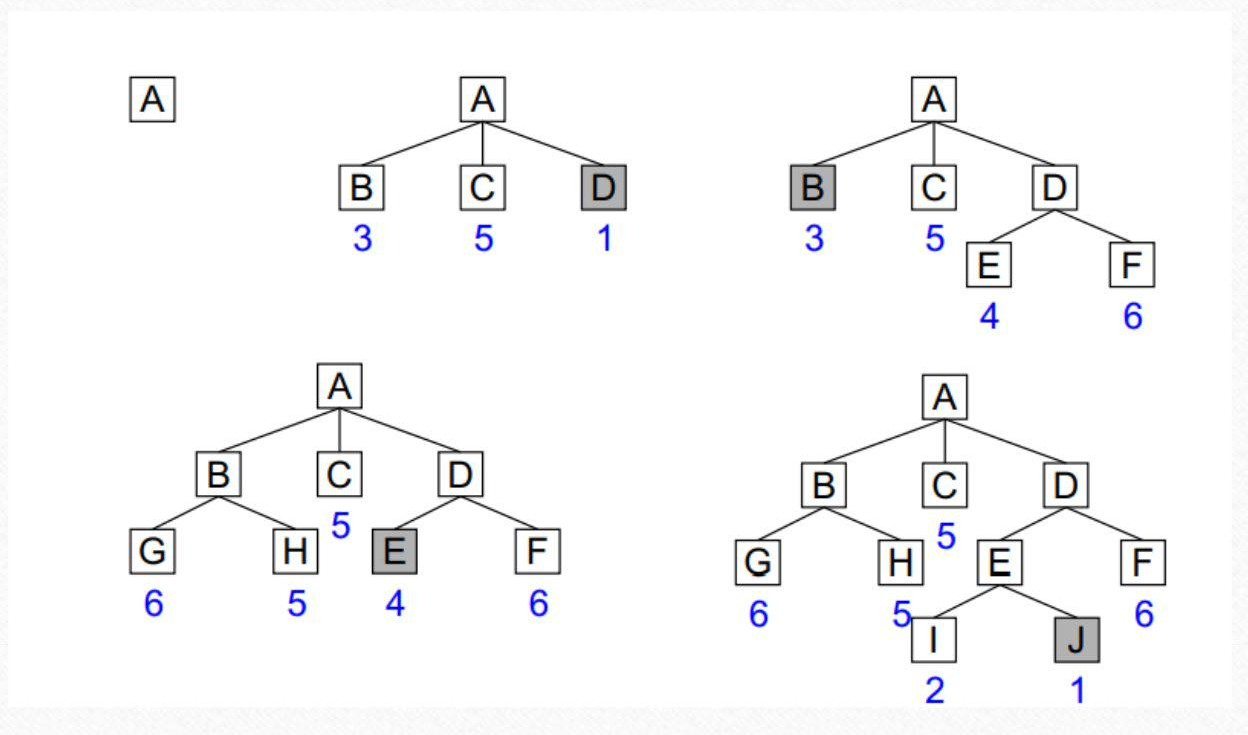












Algorithm

1. Start with OPEN holding the initial state
2. Until a goal is found or there are no nodes left on open do.
   - Pick the best node on OPEN
   - Generate its successors
   - For each successor Do
     - If it has not been generated before ,evaluate it ,add it to OPEN and record its parent
     - If it has been generated before change the parent if this new path is better and in that case update the cost of getting to any successor nodes.
3. If a goal is found or no more nodes left in OPEN, quit, else return to 2.

Disadvantages

1. It is not optimal.
2. It is incomplete because it can start down an infinite path and never return to try other possibilities.
3. The worst-case time complexity for greedy search is O (b^m), where m is the maximum depth of the search space.
4. Because greedy search retains all nodes in memory, its space complexity is the same as its time complexity

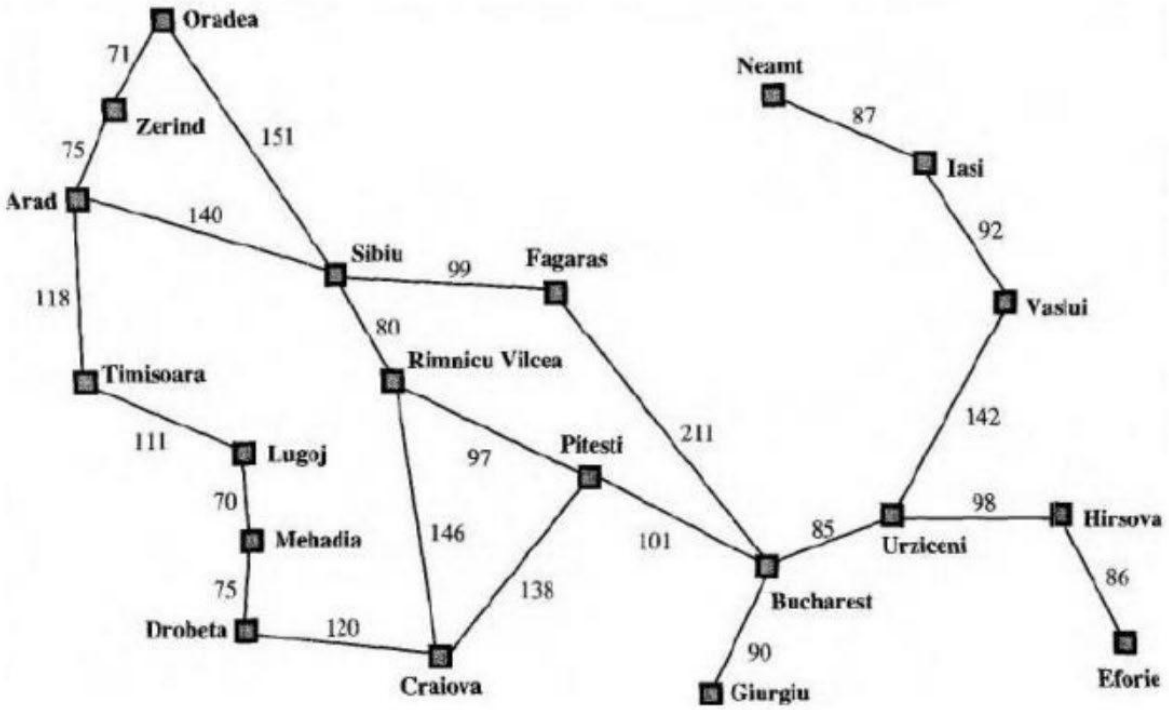
---

Question

- Taking the example of Route-finding problems in Romania, the goal is to reach Bucharest starting from the city Arad.

The heuristic costs from each city to Bucharest:

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

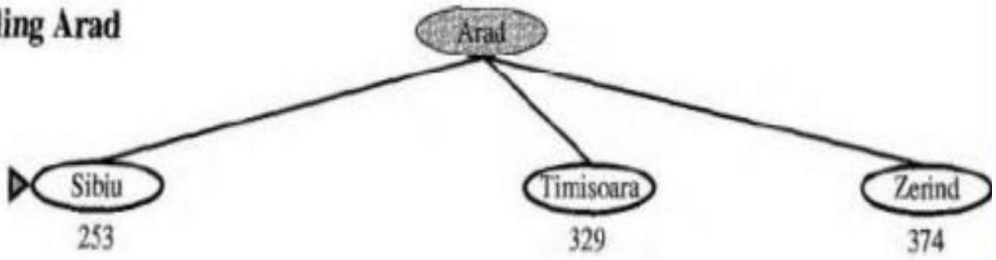


Solution

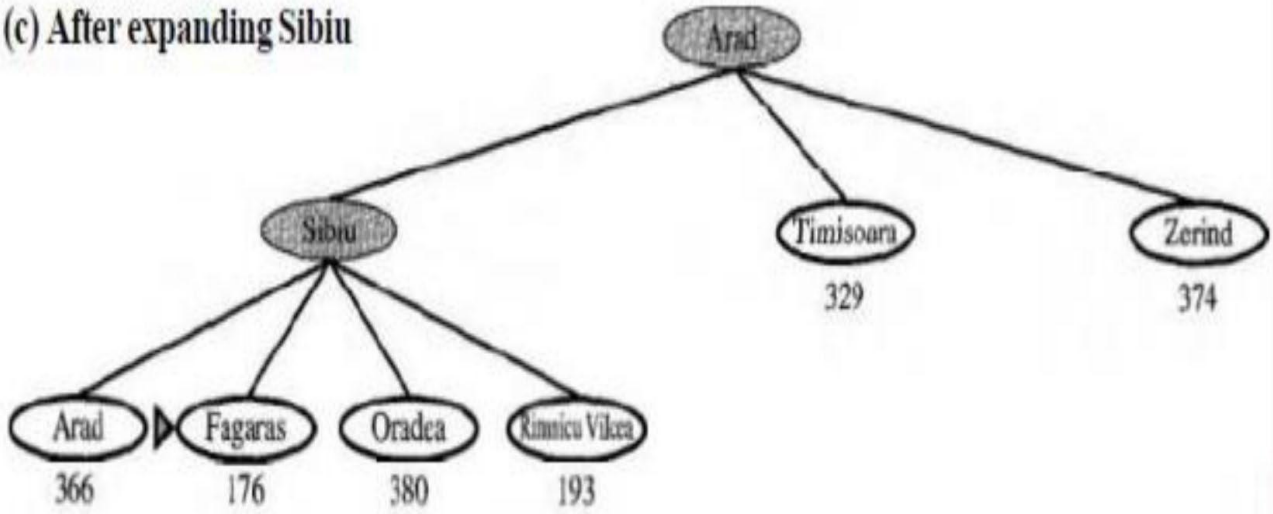
(a) The initial state

(a) The initial state

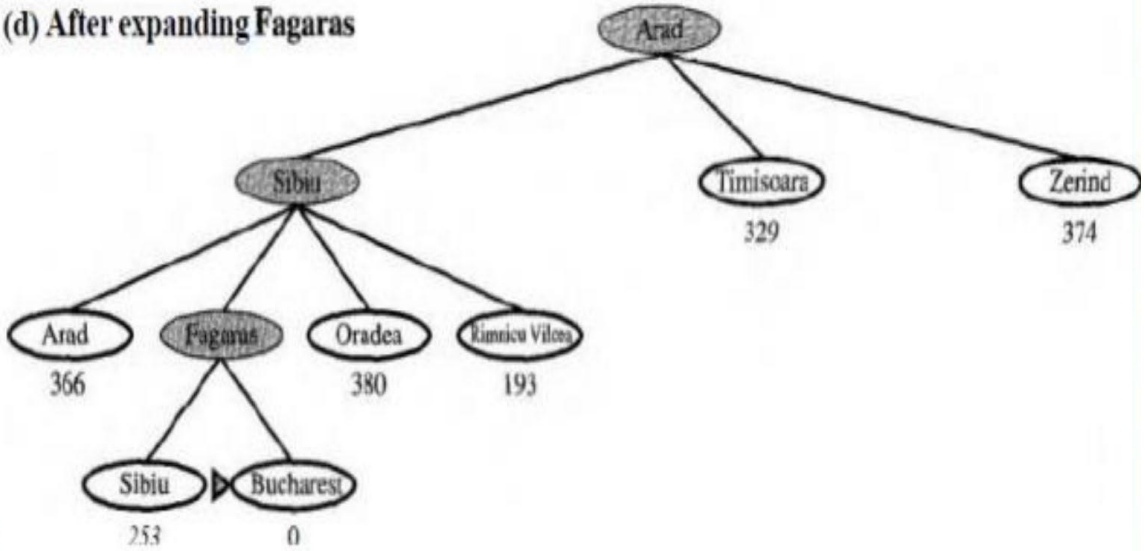
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Example:

The A* Algorithm

- The Best First algorithm is a simplified form of the A* algorithm.
- The A* search algorithm (pronounced "Ay-star") is a tree search algorithm that finds a path from a given initial node to a given goal node (or one passing a given goal test).
- It employs a "heuristic estimate" which ranks each node by an estimate of the best route that goes through that node.
- It visits the nodes in order of this heuristic estimate.
- Similar to greedy best-first search but is more accurate because A* takes into account the nodes that have already been traversed.
- A* search finds the shortest path through a search space to goal state using heuristic function.
- This technique finds minimal cost solutions and is directed to a goal state called A* search.
- In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost.
- A* requires heuristic function to evaluate the cost of path that passes through the particular state.
- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula.

Heuristic used

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

Where

g (n): The actual cost path from the start state to the current state.

h (n): The actual cost path from the current state to goal state.

f (n): The actual cost path from the start state to the goal state.

For implementation

- For the implementation of A* algorithm we will use two lists namely OPEN and CLOSED.
- **OPEN**: A list which contains the nodes that has been generated but has not been yet examined.
- **CLOSED**: A list which contains the nodes that have been examined.

Description

- A* begins at a selected node. Applied to this node is the "cost" of entering this node (usually zero for the initial node).

  A* then estimates the distance to the goal node from the current node. This estimate and the cost added together are the heuristic which is assigned to the path leading to this node.

  The node is then added to a priority queue, often called "open".

- The algorithm then removes the next node from the priority queue (because of the way a priority queue works, the node removed will have the lowest heuristic).

  If the queue is empty, there is no path from the initial node to the goal node and the algorithm stops.

  If the node is the goal node, A* constructs and outputs the successful path and stops.

- If the node is not the goal node, new nodes are created for all admissible adjoining nodes; the exact way of doing this depends on the problem at hand.

  For each successive node, A* calculates the "cost" of entering the node and saves it with the node.

  This cost is calculated from the cumulative sum of costs stored with its ancestors, plus the cost of the operation which reached this new node.

- The algorithm also maintains a 'closed' list of nodes whose adjoining nodes have been checked.

  If a newly generated node is already in this list with an equal or lower cost, no further processing is done on that node or with the path associated with it.

  If a node in the closed list matches the new one, but has been stored with a higher cost, it is removed from the closed list, and processing continues on the new node

- Next, an estimate of the new node's distance to the goal is added to the cost to form the heuristic for that node.

  This is then added to the 'open' priority queue, unless an identical node is found there.

- Once the above three steps have been repeated for each new adjoining node, the original node taken from the priority queue is added to the 'closed' list. The next node is then popped from the priority queue and the process is repeated

---

Algorithm

- a node consists of
  - state
  - g, h, f values
  - list of successors
  - pointer to parent
- OPEN is the list of nodes that have been generated and had h applied, but not expanded and can be implemented as a priority queue.
- CLOSED is the list of nodes that have already been expanded.

1) /* Initialization */

OPEN <- start node

Initialize the start node
g:
h:
f:

CLOSED <- empty list

2) repeat until goal (or time limit or space limit)

- if OPEN is empty, fail
- BESTNODE <- node on OPEN with lowest f
- if BESTNODE is a goal, exit and succeed
- remove BESTNODE from OPEN and add it to CLOSED
- generate successors of BESTNODE

for each successor s do
1. set its parent field
2. compute g(s)
3. if there is a node OLD on OPEN with the same state info as s
{ add OLD to successors(BESTNODE)
if g(s) < g(OLD), update OLD and
throw out s }

4. if (s is not on OPEN and there is a node OLD on CLOSED with the same state info as s
{ add OLD to successors(BESTNODE)
if g(s) < g(OLD), update OLD,
remove it from CLOSED
and put it on OPEN, throw out s
}

5. If s was not on OPEN or CLOSED
{ add s to OPEN
add s to successors(BESTNODE)
calculate g(s), h(s), f(s) }

end of repeat loop

Advantages

- It is complete and optimal.
- It is the best one from other techniques.
- It is used to solve very complex problems.
- It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages

- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- The speed execution of A* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute h (n).
- It has complexity problems.

The g function in A*

- g is a measure of how good a path is.
- Useful for solutions where we want to find the cheapest path.
- For problems where the path cost is not of any concern, search can be guided solely by h, i.e. set g=0 always
- If a path with shortest no. of steps is required, set cost of each branch =1.
- If cheapest path is required and some operators cost more than others, individual branches will have varying costs.
- Thus A* is useful to find both any path solutions as well as minimal path cost solutions.

The h (heuristic) function in A*

- h represents the distance of a node to the goal.
- It is an estimate only, often represented by h‘ where the ‘ represents that this is not an exact cost, but an estimation.
- It represents how good the node itself is.
- h can be either a perfect estimate, an underestimate or an overestimate.
- If h is always set to 0, the search is controlled by g.
- If both h and g are set to 0, it will be a random search.
- If h=0 and g=1, it reduces to breadth first search. All nodes on a lower level will have lower g (and hence f) values and thus will get expanded before nodes on the next level.

- If h is a perfect estimator of the true cost then A* will always pick the correct successor with no search.
- If h is admissible, A* with TREE-SEARCH is guaranteed to give the optimal solution.
- If h is consistent, too, then GRAPH-SEARCH is optimal.
- If h is not admissable, no guarantees, but it can work well if h is not often greater than the true cost.

The case when h‘ underestimates h

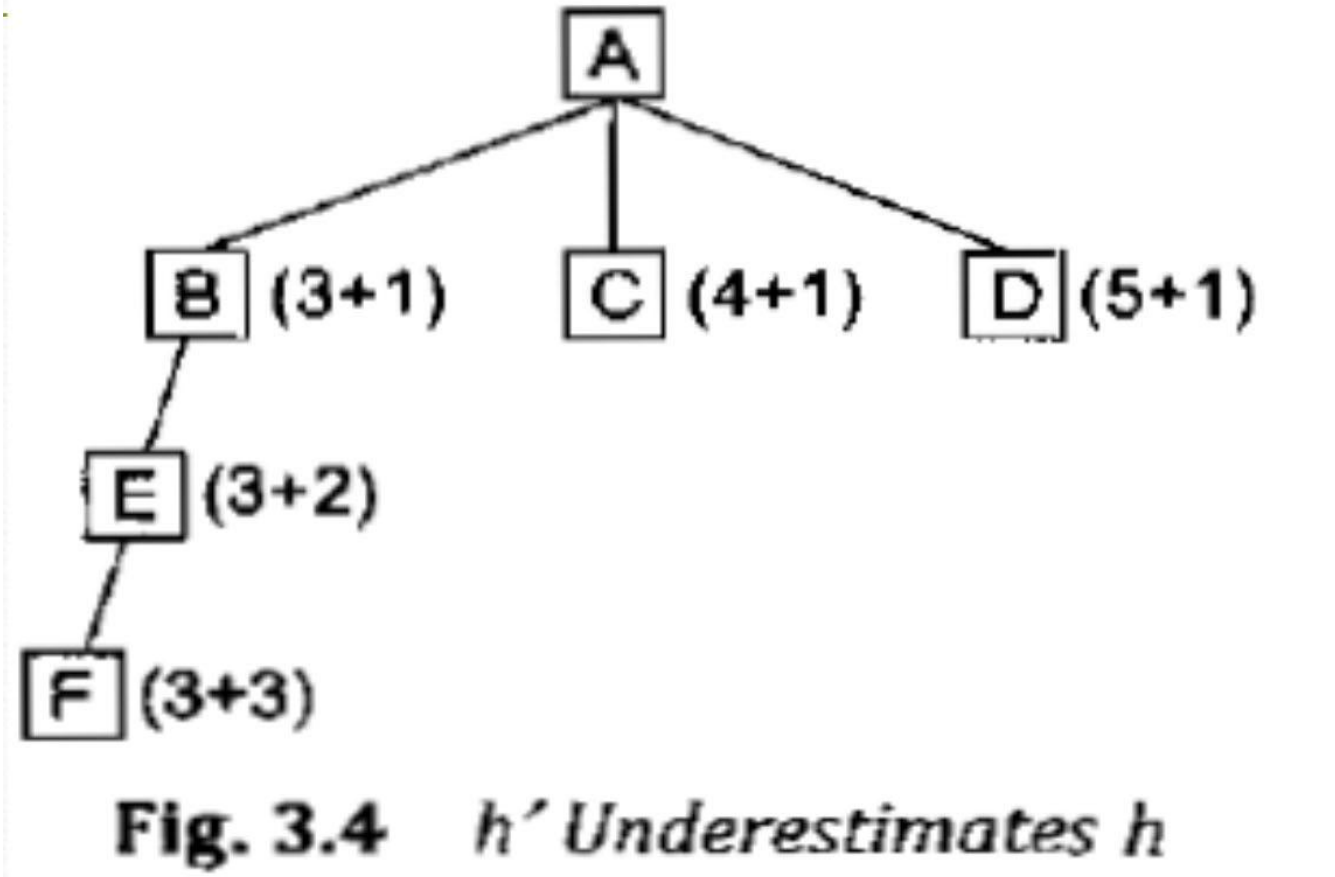


Fig. 3.4 *h′ Underestimates h*

- Wastes some effort.

Consider the situation shown in Fig. 3.4. Assume that the cost of all arcs is 1. Initially, all nodes except A are on *OPEN* (although the Fig. show the situation two steps later, after B and E have been expanded). For each node, f' is indicated as the sum of h' and g. In this example, node B has the lowest f', 4, so it is expanded first. Suppose it has only one successor E which also appears to be three moves away from a goal. Now $f'(E)$ is 5 the same as $f'(C)$. Suppose we resolve this in favor of the path we are currently following. Then we will expand E next. Suppose it too has a single successor F, also judged to be three moves from a goal. We are clearly using up moves and making no progress. But $f'(F) = 6$, which is greater than $f'(C)$. So we will expand C next. Thus we see that by underestimating $h'(B)$ we have wasted some effort. But eventually we discover that B was farther away than we thought and we go back and try another path.

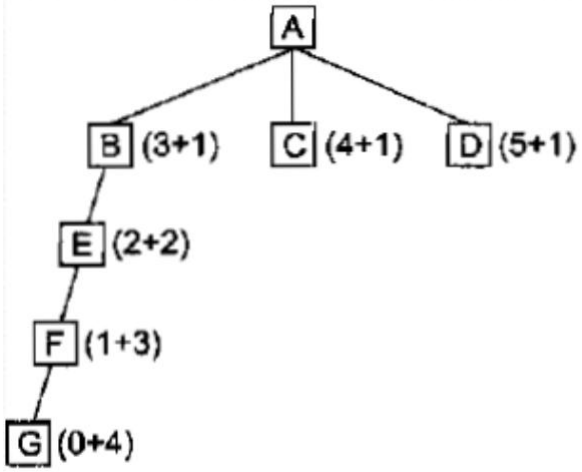


Fig. 3.5 *h' Overestimates h*

Now consider the situation shown in Fig. 3.5. Again we expand B on the first step. On the second step we again expand E. At the next step we expand F, and finally we generate G, for a solution path of length 4. But suppose there is a direct path from D to a solution, giving a path of length 2. We will never find it. By overestimating $h'(D)$ we make D look so bad that we may find some other, worse solution without ever expanding D. In general, if h' might overestimate h, we cannot be guaranteed of finding the cheapest path solution unless we expand the entire graph until all paths are longer than the best solution. An

---

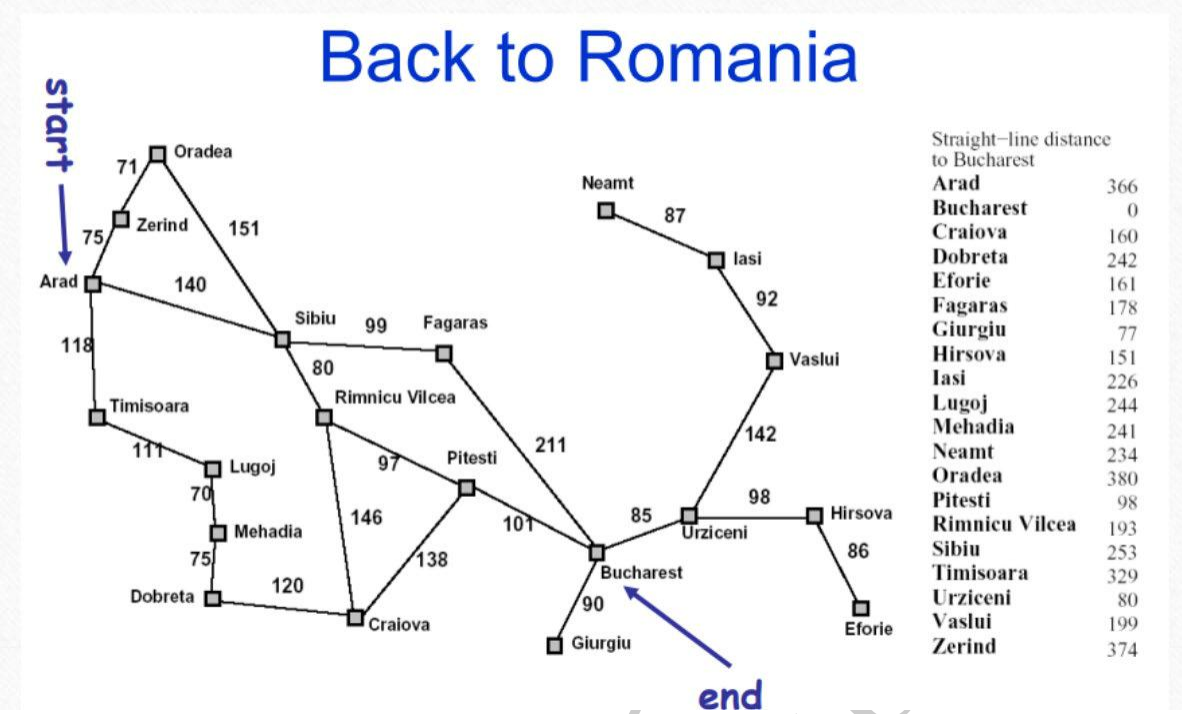
Admissibility

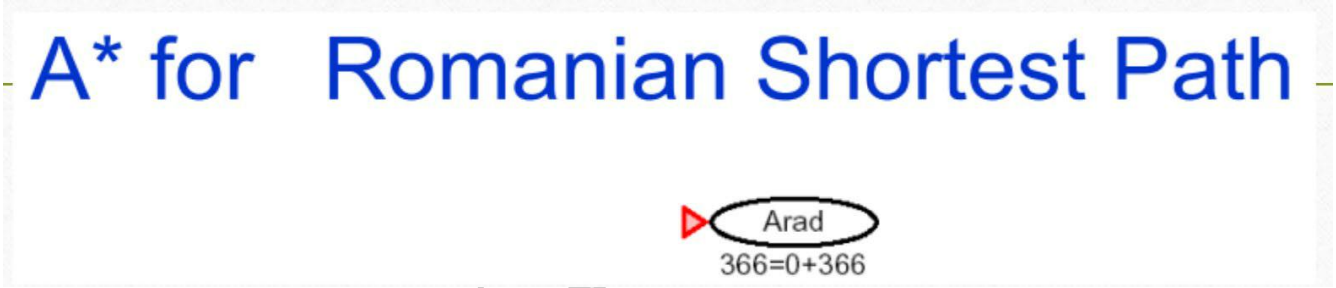
- The algorithm A* is admissible.
  - This means that provided a solution exists, the first solution found by A* is an optimal solution.
  - A* is admissible under the following conditions: Heuristic function: for every node n $h(n) \leq h^*(n)$.
- A* is also complete.
- A* is optimally efficient for a given heuristic.
- A* is much more efficient that uninformed search.

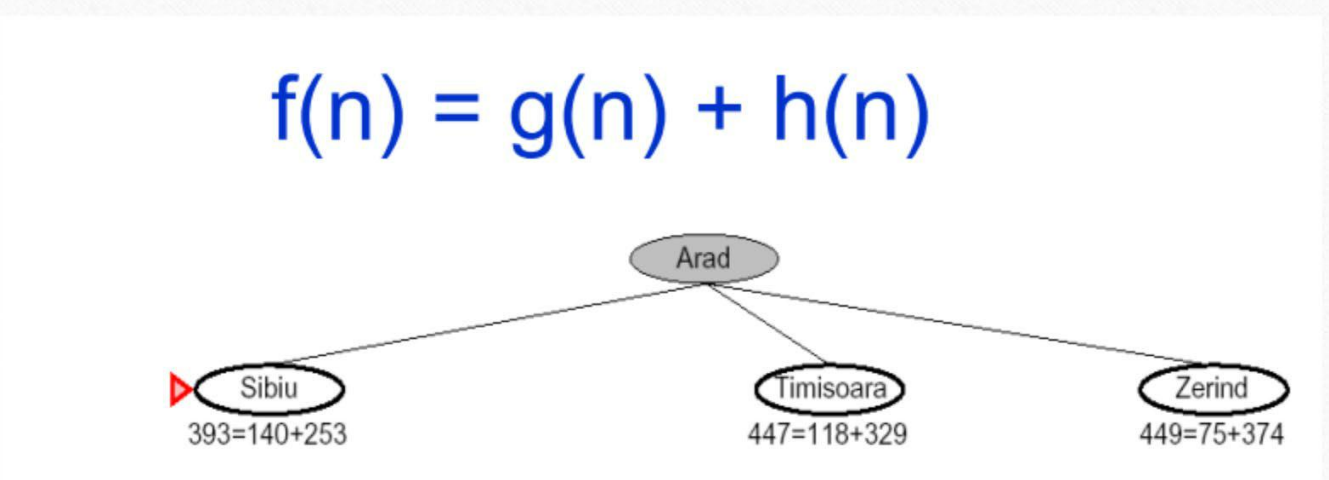
Graceful decay of admissibility

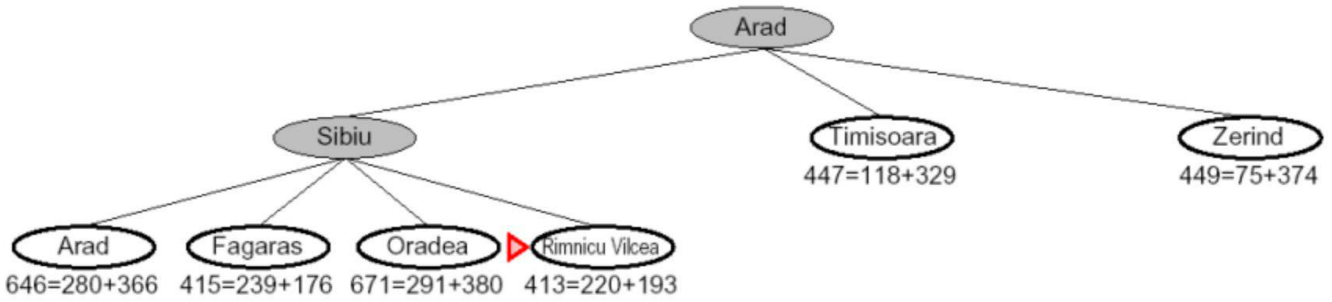
- If h' rarely overestimates h by more than d then the A* algorithm will rarely find a solution whose cost is d greater than the optimal solution.
- Most practical

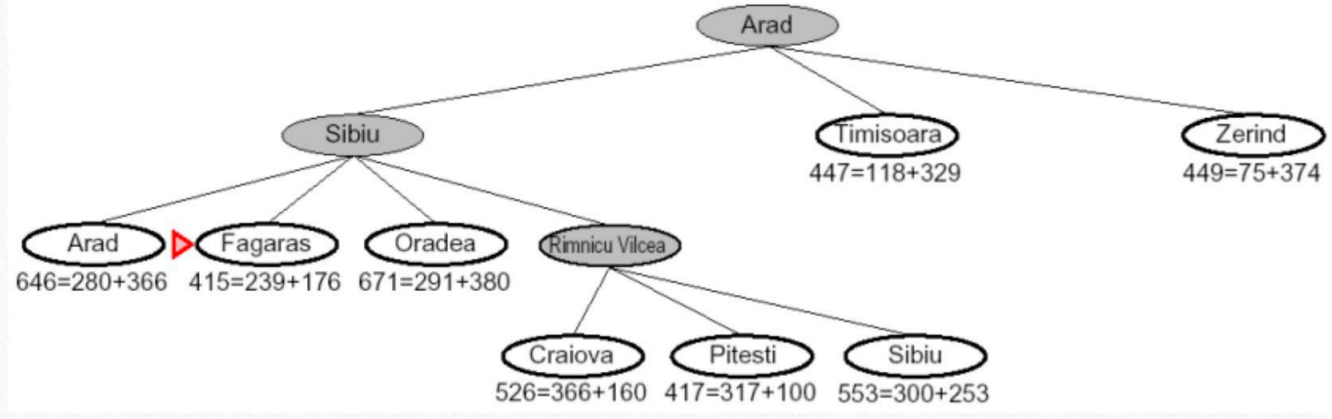
Question – Apply A* to the Route problem

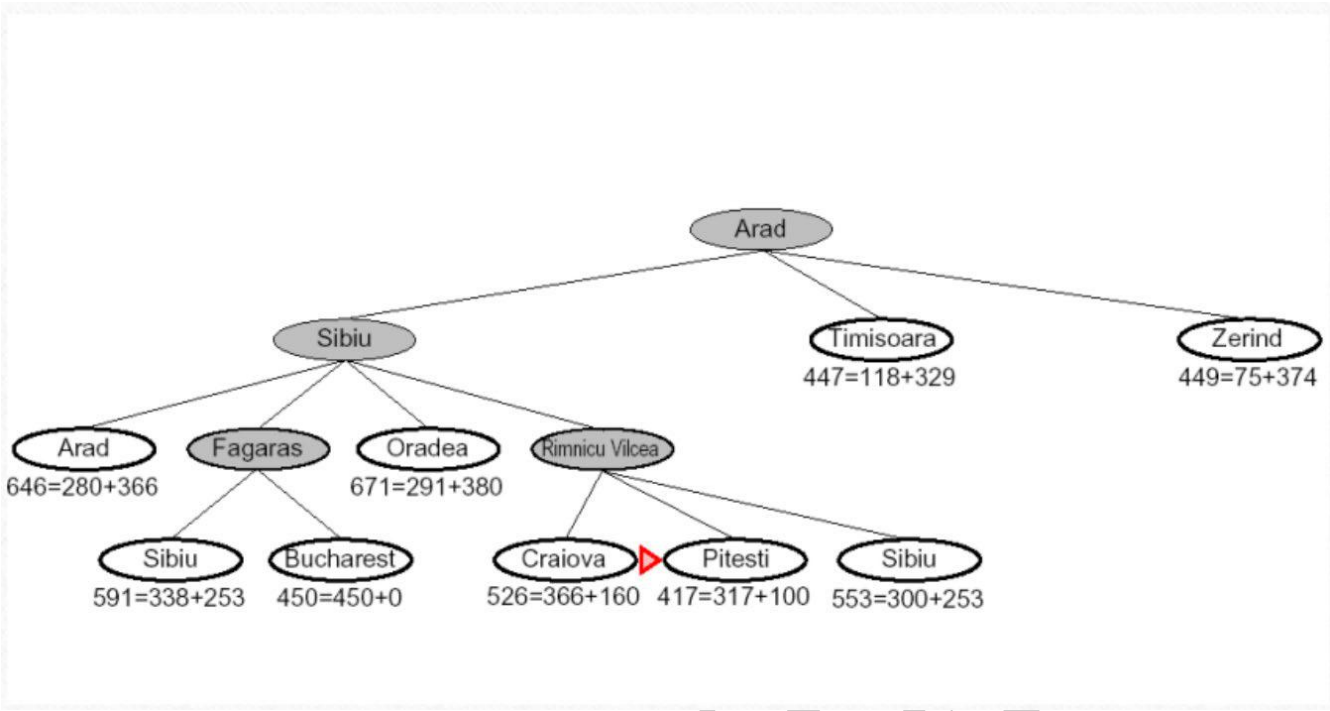


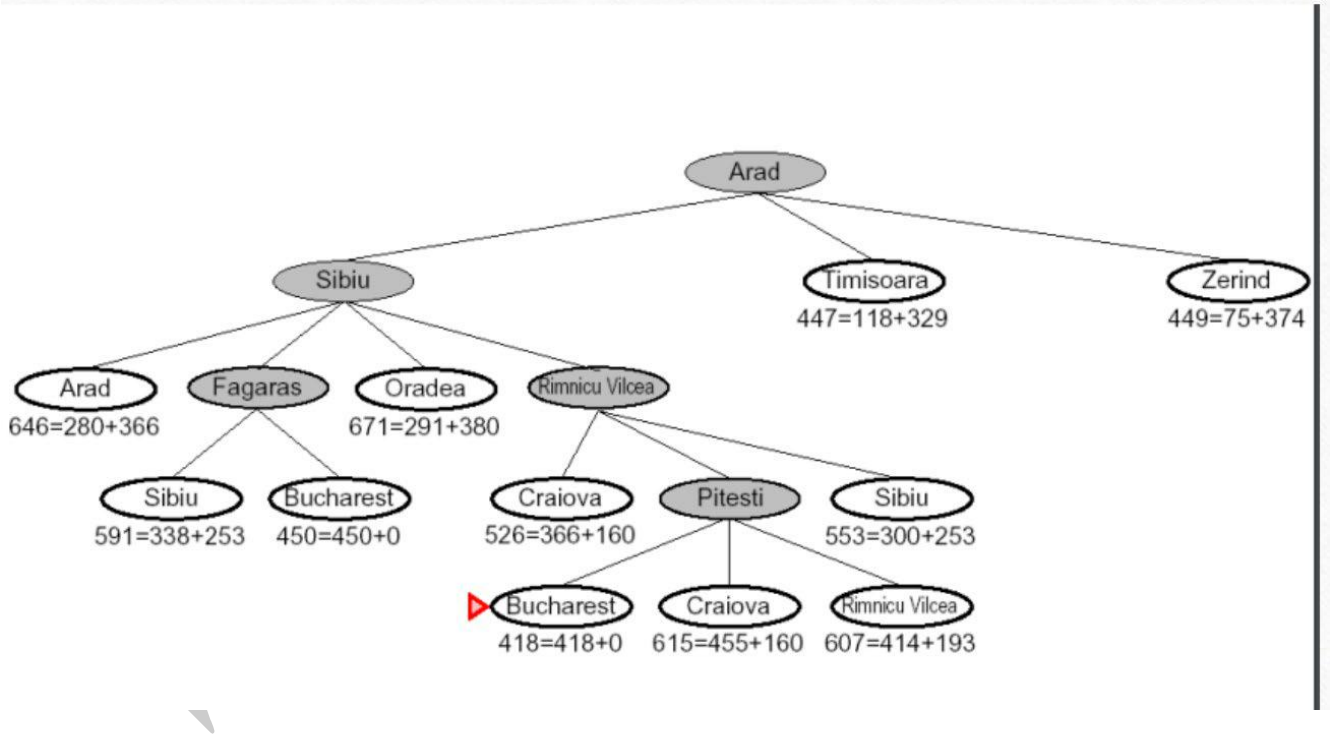





Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

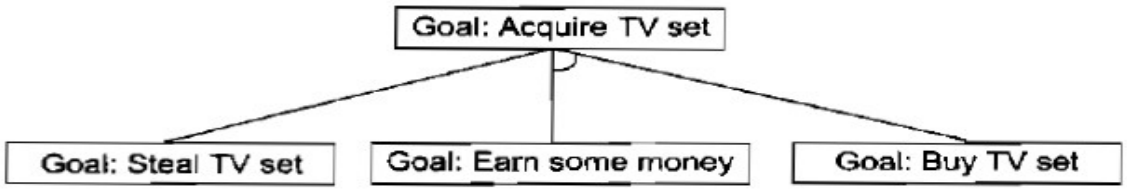
PROBLEM REDUCTION using AND-OR Graphs

So far, we have considered search strategies for OR graphs through which we want to find a single, path to a goal. Such structures represent the fact that we will know how to get from a node to a goal state if we can discover how to get from that node to a goal state along any one of the branches leaving it.

AND-OR Graphs

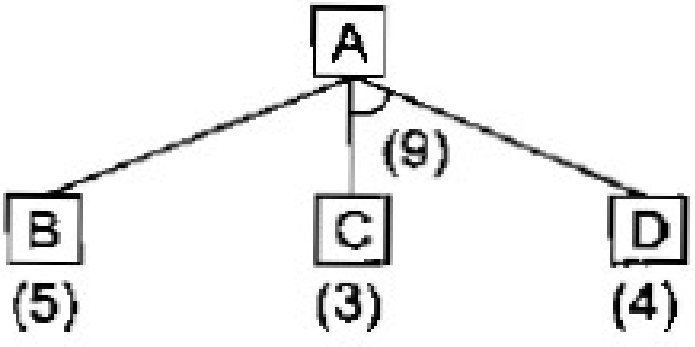
- When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution.
- The decomposition of the problem or problem reduction generates AND arcs.
- One AND are may point to any number of successor nodes. All these must be solved so that the arc will rise to many arcs, indicating several possible solutions.
- Hence the graph is known as AND - OR instead of AND. Figure shows an AND - OR graph.

A simple example of AND-OR graph



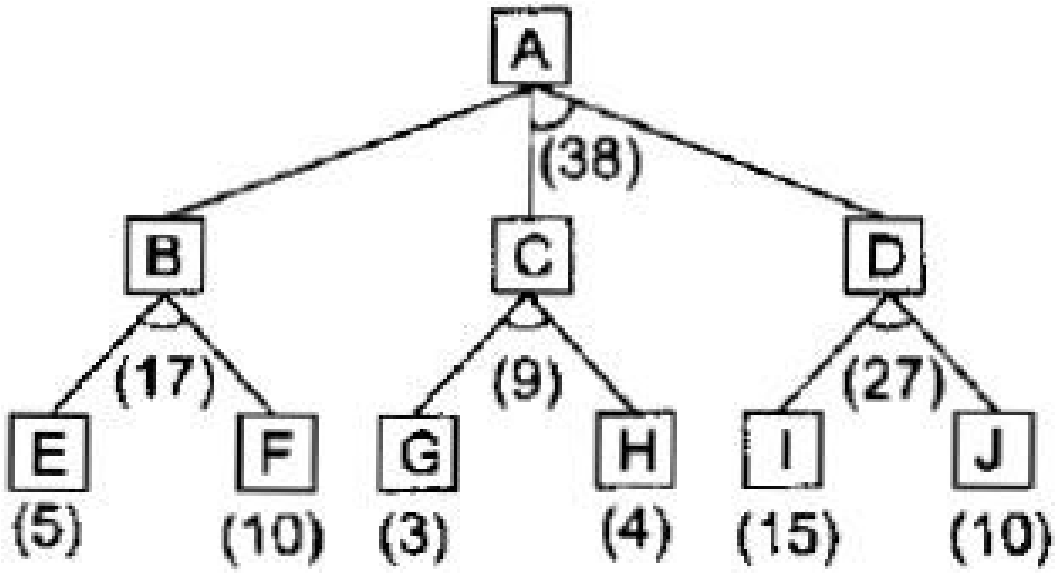
Is A* algorithm sufficient for AND- OR Graphs?

- An algorithm to find a solution in an AND - OR graph must handle AND area appropriately. A* algorithm can not search AND - OR graphs efficiently. This can be understand from the given figure.



- In figure (a) the top node A has been expanded producing two area one leading to B and leading to C-D . the numbers at each node represent the value of f ' at that node (cost of getting to the goal state from current state). For simplicity, it is assumed that every operation(i.e. applying a rule) has unit cost, i.e., each are with single successor will have a cost of 1 and each of its components. With the available information till now , it appears that C is the most promising node to expand since its f ' = 3 , the lowest but going through B would be better since to use C we must also use D' and the cost would be 9(3+4+1+1). Through B it would be 6(5+1).
- **Thus the choice of the next node to expand depends not only n a value but also on whether that node is part of the current best path form the initial mode.**

Another example



- The previous figure makes this clearer. In figure the node G appears to be the most promising node, with the least f ' value. But G is not on the current beat path, since to use G we must use GH with a cost of 9 and again this demands that arcs be used (with a cost of 27). The path from A through B, E-F is better with a total cost of (17+1=18)

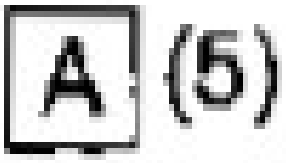
Searching AND-OR Graphs

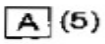
- Thus we can see that to search an AND-OR graph, the following three things must be done.
- 1. traverse the graph starting at the initial node and following the current best path, and accumulate the set of nodes that are on the path and have not yet been expanded.
- 2. Pick one of these unexpanded nodes and expand it. Add its successors to the graph and computer f ' (cost of the remaining distance) for each of them.
- 3. Change the f ' estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which of the current best path.

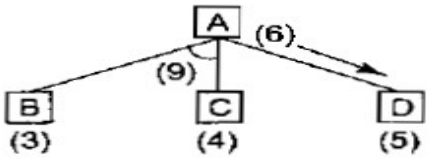
Difference between Best First Search and AO*

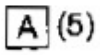
- The propagation of revised cost estimation backward is in the tree is not necessary in A* algorithm. This is because in AO* algorithm expanded nodes are re-examined so that the current best path can be selected.

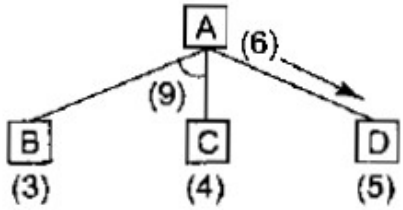
Problem Reduction - Example

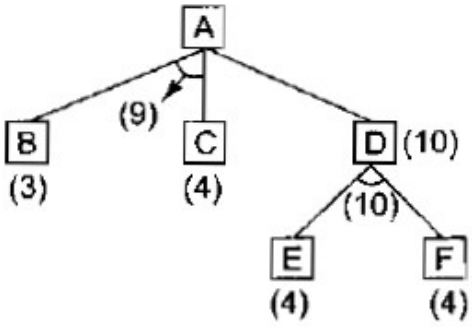


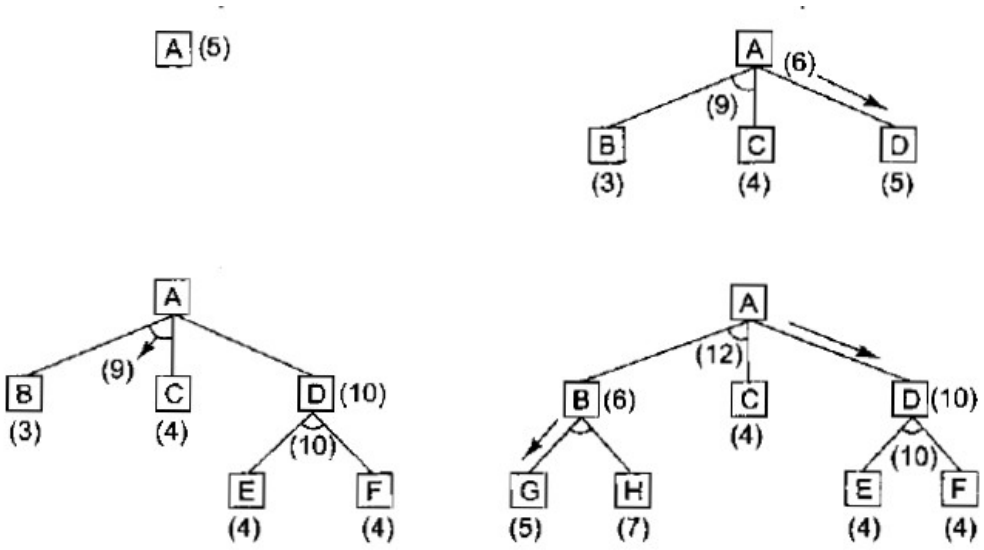










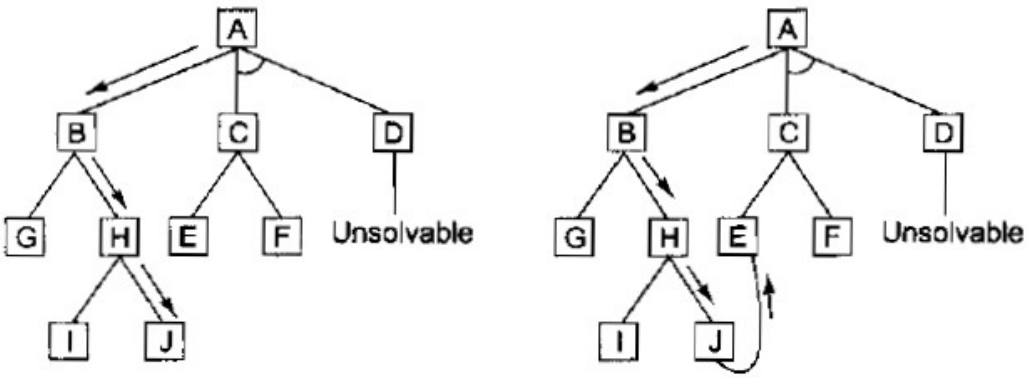


This process is illustrated in Fig. 3.8. At step 1, A is the only node, so it is at the end of the current best path. It is expanded, yielding nodes B, C, and D. The arc to D is labeled as the most promising one emerging from A, since it costs 6 compared to B and C, which costs 9. (Marked arcs are indicated in the Fig.s by arrows.) In step 2, node D) is chosen for expansion. This process produces one new arc, the AND arc to E and F, with a combined cost estimate of 10. So we update the f' value of D to 10. Going back one more level, we see that this makes the AND arc B-C better than the arc to D, so it is labeled as the current best path. At step 3, we traverse that arc from A and discover the unexpanded nodes B and C. If we are going to find a solution along this path, we will have to expand both B and C eventually, so let's choose to explore B first. This generates two new arcs, the ones to G and to H. Propagating their f' values backward, we update f' of B to 6 (since that is the best we think we can do, which we can achieve by going through G). This requires updating the cost of the AND arc B-C to 12 (6 + 4 + 2). After doing that, the arc to D is again the better path from A, so we record that as the current best path and either node E or node F will be chosen for expansion at step 4. This process continues until either a solution is found or all paths have led to dead ends, indicating that there is no solution.

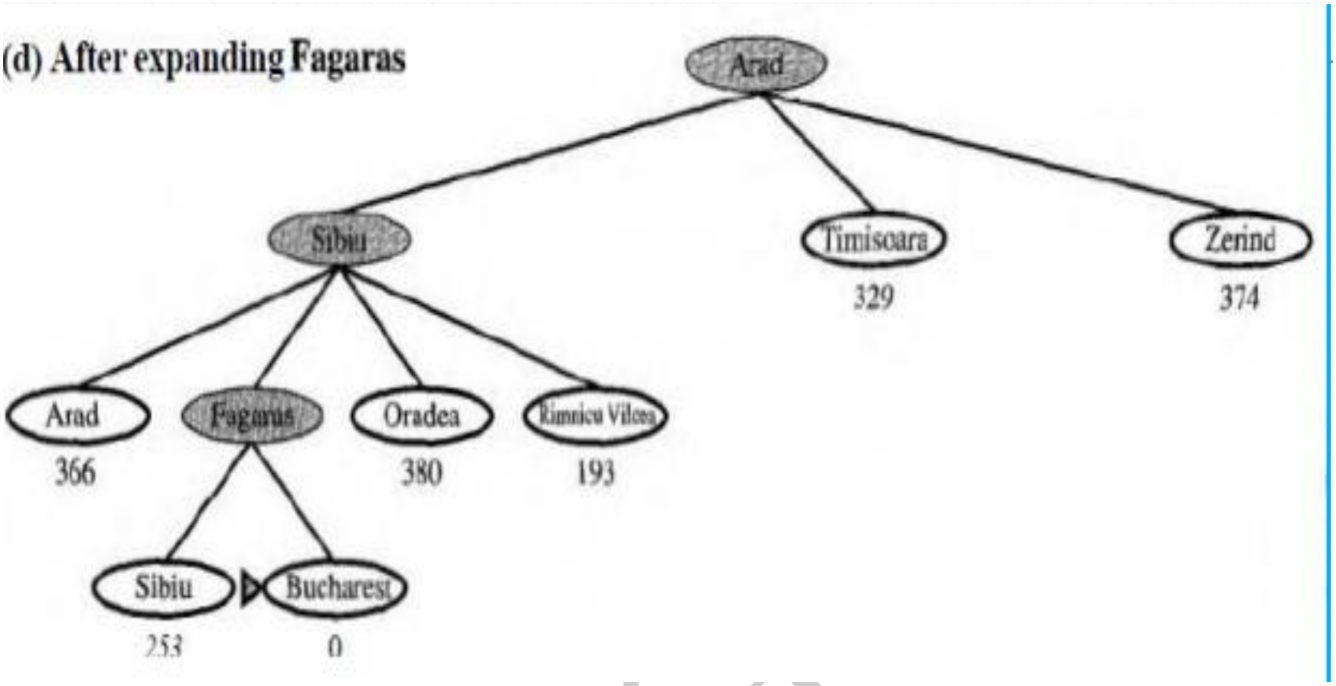
Another difference between BFS and AO*

- In best first search, the desired path from one node to another was always the one with the lowest cost. But this is not always the case with AND-OR graphs.

Example: a longer path may be better



Consider the example shown in the first fig. The nodes were generated in alphabetical order. Now suppose that node J is expanded at the next step and that one of its successors is node E, producing the graph shown next to it. This new path to E is longer than the previous path to E going through C. But since the path through C will only lead to a solution if there is also a solution to D, which we know there is not, the path through J is better.



1. Initialize the graph to the starting node.
2. Loop until the starting node is labeled *SOLVED* or until its cost goes above *FUTILITY*:
   (a) Traverse the graph, starting at the initial node and following the current best path, and accumulate the set of nodes that are on that path and have not yet been expanded or labeled as solved.
   (b) Pick one of these unexpanded nodes and expand it. If there are no successors, assign *FUTILITY* as the value of this node. Otherwise, add its successors to the graph and for each of them compute f' (use only h' and ignore g, for reasons we discuss below). If of any node is 0, mark that node as *SOLVED*.
   (c) Change the f' estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains a successor arc whose descendants are all solved, label the node itself as *SOLVED*. At each node that is visited while going up the graph, decide which of its successor arcs is the most promising and mark it as part of the current best path. This may cause the current best path to change. This propagation of revised cost estimates back up the tree was not necessary in the best-first search algorithm because only unexpanded nodes were examined. But now expanded nodes must be reexamined so that the best current path can be selected. Thus it is important that their f' values be the best estimates available.

AO*

- The algorithm for performing a heuristic search of an AND - OR graph is the AO* algorithm.
- Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G. G represents the part of the search graph generated so far. Each node in G points down to its immediate successors and up to its immediate predecessors, and also has with it the value of h' cost of a path from itself to a set of solution nodes.
- The cost of getting from the start nodes to the current node "g" is not stored as in the A* algorithm. This is because it is not possible to compute a single such value since there may be many paths to the same state. In AO* algorithm h‘ serves as the estimate of goodness of a node. Also a there should be value called FUTILITY. The estimated cost of a solution is greater than FUTILITY then the search is abandoned as too expansive to be practical.

Algorithm

- 1. Let G consists only to the node representing the initial state call this node $INIT$. Compute $h'(INIT)$.
- 2. Until $INIT$ is labeled $SOLVED$ or $h'(INIT)$ becomes greater than $FUTILITY$, repeat the following procedure.
  - (I) Trace the marked arcs from $INIT$ and select an unbounded node $NODE$.
  - (II) Generate the successors of $NODE$. if there are no successors then assign $FUTILITY$ as $h'(NODE)$. This means that $NODE$ is not solvable. If there are successors then for each one called $SUCCESSOR$, that is not also an ancestor of $NODE$ do the following
    - (a) add $SUCCESSOR$ to graph G
    - (b) if $SUCCESSOR$ is a terminal node, mark it solved and assign zero to its h' value.
    - (c) If $SUCCESSOR$ is not a terminal node, compute its h' value.
  - (III) propagate the newly discovered information up the graph by doing the following . let S be a set of nodes that have been marked $SOLVED$. Initialize S to $NODE$. Until S is empty repeat the following procedure;
    - (a) select a node from S call it $CURRENT$ and remove it from S.
    - (b) compute h' of each of the arcs emerging from $CURRENT$, Assign minimum h' to $CURRENT$.
    - (c) Mark the minimum cost path as the best out of $CURRENT$.
    - (d) Mark $CURRENT$ as $SOLVED$ if all of the nodes connected to it through the new marked have been labeled $SOLVED$.
    - (e) If $CURRENT$ has been marked $SOLVED$ or its h' has just changed, its new status must be propagate backwards up the graph . hence all the ancestors of $CURRENT$ are added to S.
