KNOWLEDGE, SEARCH, PROBLEM SOLVING

Introduction to ML, DL, AI and OpenVino

Session 03

Pramod Sharma
pramod.sharma@prasami.com

Agenda

2

- States
- States Search
- Goal Formulation
- Informed Search
- Uninformed Search
- Adversarial Search

3

Part I

pra-sâmi

4

What is State?

- All information about state…
- All information necessary to make decision about task at hand

pra-sâmi

5

Agent's Knowledge Representation

Atomic	States are indivisible; No internal structure	Search on atomic states; Heuristic and Blind searches
Propositional (aka Factored)	States are made of state variables that take values (Propositional or multi-values or Continuous)	Search + Inference in (prop logic) and probabilistic (Bayes Nets) representations
Relational	States Describe the objects in the world and their inter-relations	Search + Inference in predicate logic (or relational prob. Models)
First-order	States Describe the objects in the world and their inter-relations along with Functions over the objects	Search + Inference in first order logic
Probability Theory	Add degree of belief to the facts	Inference with some degree of belief

pra-sâmi

6

Defining the problem as State Space Search

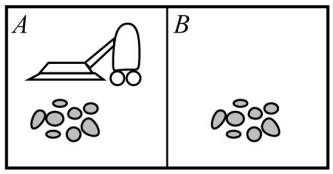
- The state space representation forms the basis of most of the AI methods.
- Its structure corresponds to the structure of problem solving in two important ways:
  - It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
  - It permits us to define the process of solving a particular problem as a combination of known techniques (each represented as a rule defining a single step in the space) and search, the general technique of exploring the space to try to find some path from current state to a goal state.
- Search is a very important process in the solution of problems for which no more direct techniques are available.

pra-sâmi

7

Toy Vacuum World

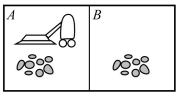
- Two Locations : squares A and B
- The vacuum agent perceives:
  - Which square it is in
  - Whether there is dirt in the square
- Actions: move left, move right, suck up the dirt, or do nothing
- Agent function:
  - if the current square is dirty, then suck;
  - otherwise, move to the other square

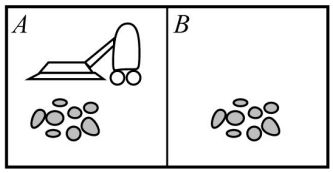


pra-sâmi

8

Toy Vacuum World – Possible states

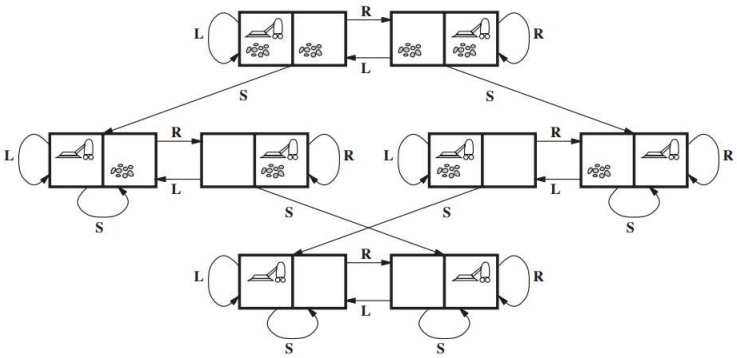




pra-sâmi

9

Toy Example : Vacuum World



The state space for the vacuum world.
Links denote actions L : Turn Left R : Turn Right S: Suck

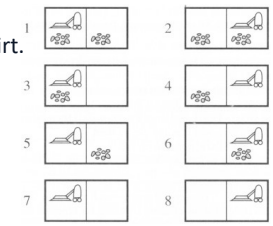
Compared with the real world, this toy problem has discrete locations, discrete dirt, reliable cleaning, and it never gets any dirtier

pra-sâmi

10

Toy Example : Vacuum World

- States: The state is determined by both the agent location and the dirt locations.
  - The agent could be in one of two locations, each of which might or might not contain dirt.
  - Thus, there are $2 \times 2^2 = 8$ possible world states.
  - A larger environment with n locations has $n \cdot 2^n$ states.



- Initial state: Any state can be designated as the initial state.
- Actions: In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments might also include Up and Down.
- Transition model: The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect.
- Goal test: This checks whether all the squares are clean.
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

pra-sâmi

11

Solving Problems by Searching

- How an agent can look ahead to find a sequence of actions that will eventually achieve its goal
- When the correct action to take is not immediately obvious, an agent may need to plan ahead
  - To consider a sequence of actions that form a path to a goal state
- Such an agent is called a problem-solving agent,
- The computational process it undertakes is called search

pra-sâmi

12

Correct and Complete Answer

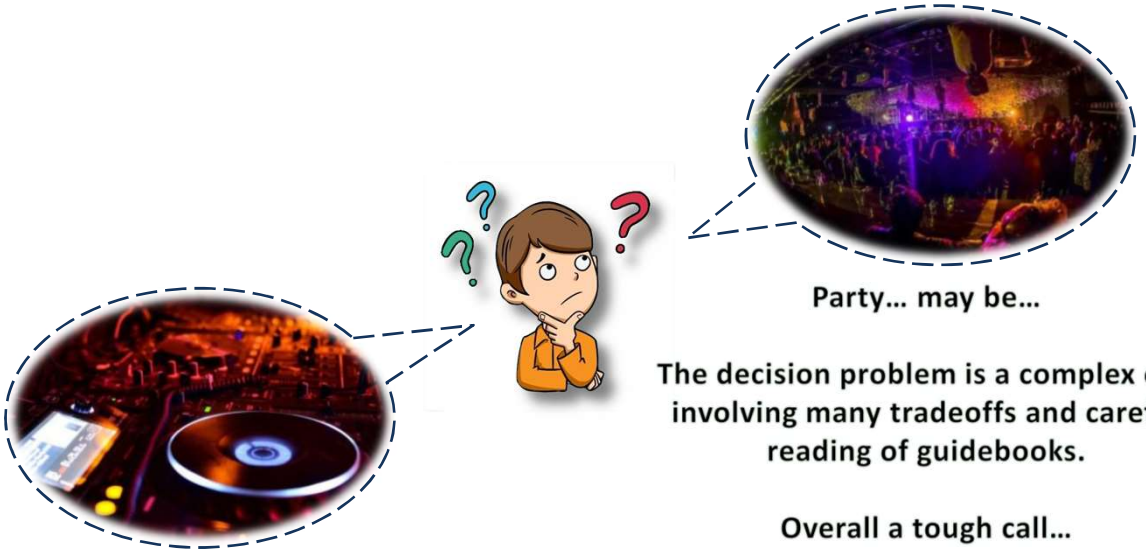
- Assume we have some way of defining the problem, let's look at the solution
- Answer returned is incorrect or incomplete?
  - How do you define complete or correct answer?
- There four common classes of solutions
  - Optimal solution
  - Satisficing solution
  - Approximately optimal solution
  - Probable solution

pra-sâmi

13 Goal Formulation

pra-sâmi

14 Goal Formulation



Party… may be…

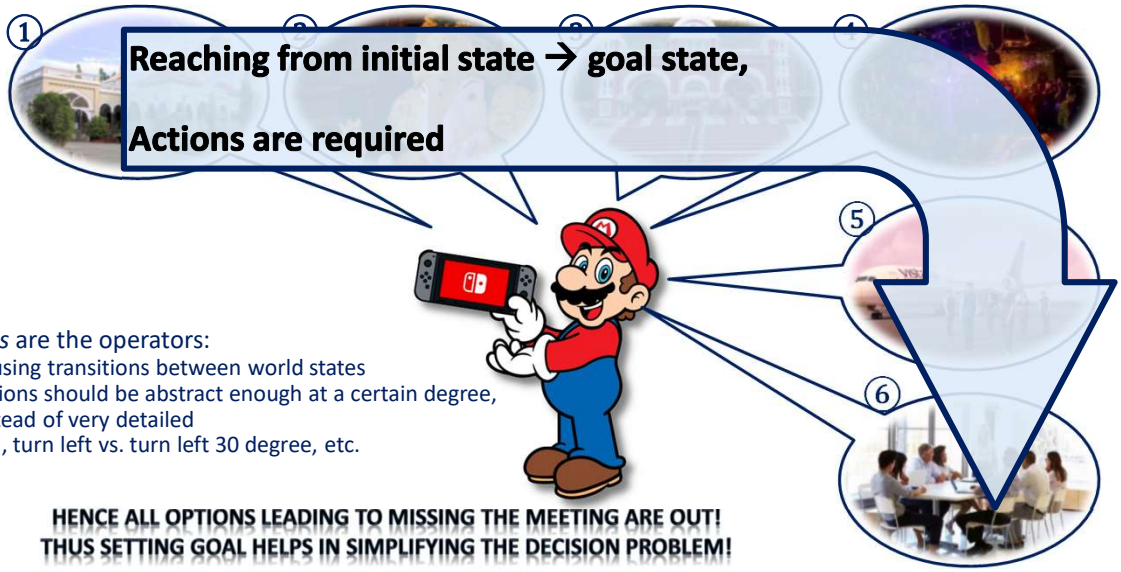
The decision problem is a complex one involving many tradeoffs and careful reading of guidebooks.

Overall a tough call…

pra-sâmi

15 Goal Formulation

16 Goal Formulation



Actions are the operators:
- Causing transitions between world states
- Actions should be abstract enough at a certain degree, instead of very detailed
- E.g., turn left vs. turn left 30 degree, etc.

HENCE ALL OPTIONS LEADING TO MISSING THE MEETING ARE OUT!
THUS SETTING GOAL HELPS IN SIMPLIFYING THE DECISION PROBLEM!

17

Recap - From our Example

- Formulate Goal
  - Be In the Meeting
- Formulate Problem
  - States : Locations of interest
  - Actions : reach location, enjoy the time there
- Find Solution
  - Sequence of activities

pra-sâmi

18

Search

- Because there are many ways to achieve the same goal
  - Together, those ways are expressed as a tree
  - Multiple options of unknown value at a point,
    - The agent can examine different possible sequences of actions, and choose the best
  - This process of looking for the best sequence is called search
  - The best sequence is then a list of actions, called solution
- Design of an agent
  - "Formulate, Search, Execute"

pra-sâmi

20 The Goal Test

- Applied to the current state to test
  - If the agent is in its goal state,
  - We don't want to continue to work if we have already reached the goal
- Sometimes there is an explicit set of possible goal states.
  - E.g. Reach the meeting in time
- Sometimes the goal is described by the properties
  - Instead of stating explicitly the set of states
  - Example: Chess
    - The agent wins if it can capture the KING of the opponent on next move (checkmate) no matter what the opponent does

21 A Path Cost Function

- Assigns a numeric cost to each path
- Select a performance measure denoted by g
- To distinguish the best path from others
- Usually the path cost is
  - The sum of the step costs of the individual actions (in the action list)

22

Well-defined Problems and Solutions

- Together a problem is defined by
  - Initial state (that the agent starts in)
  - Actions (The set of possible actions)
  - Transition model (description of what each action does)
    or
    Successor functions (refer to any state reachable from given state by a single action)
  - Goal test
  - Path in the state space:
    - Any sequence of states connected by a sequence of actions.
  - Path cost function

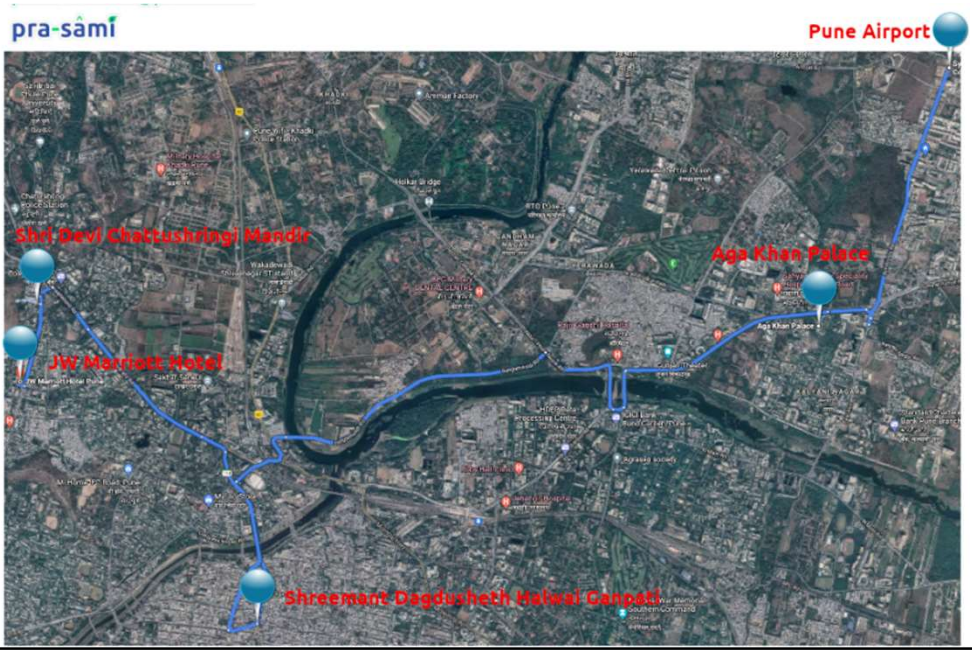
- The solution of a problem is then
  - A path from the initial state to a state satisfying the **goal test**

- Optimal solution
  - The solution with **lowest path cost** among all solutions

pra-sâmi

23

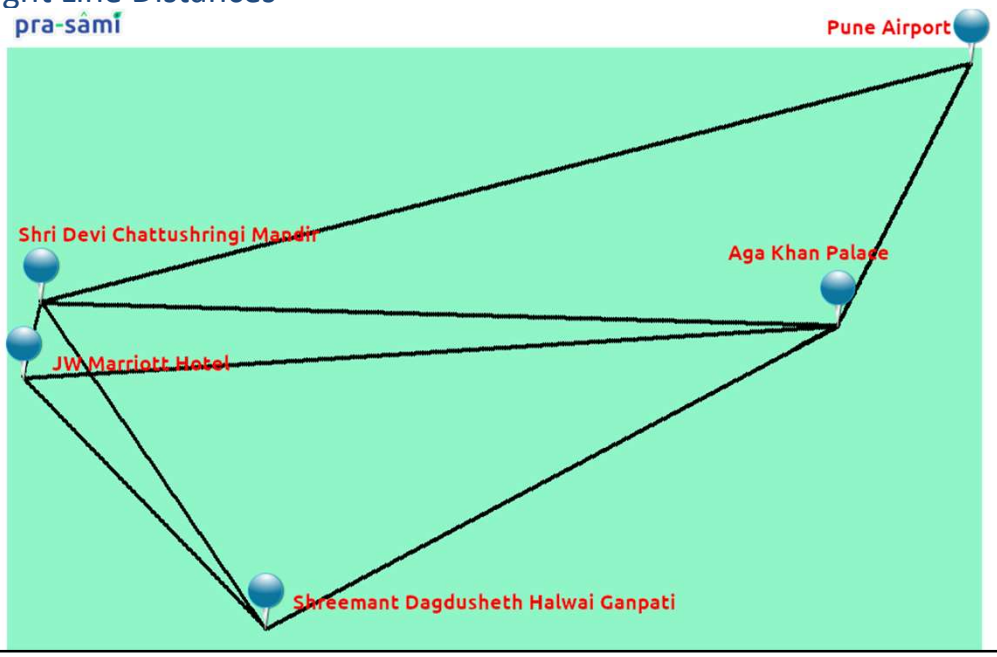
Locations



pra-sâmi

24

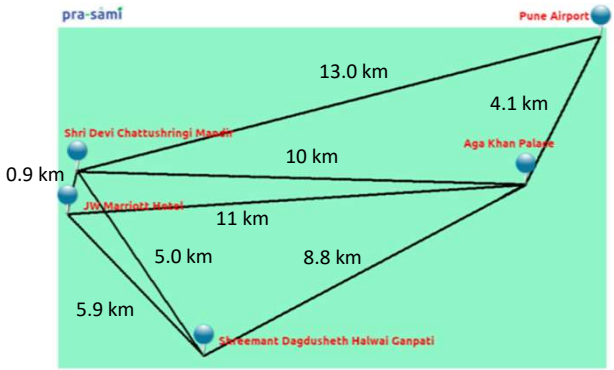
Straight Line Distances



pra-sâmi

25

Straight Line Distances



Actual Distance by Road

Locations		SLD (km)
From	to	
Airport	Aga Khan Palace	2.87
Airport	Shreemant Dagdusheth Halwai Ganpati	8.80
Airport	Shri Devi Chattushringi Mandir	9.32
Airport	JW Marriott	9.73
Aga Khan Palace	Shreemant Dagdusheth Halwai Ganpati	6.31
Aga Khan Palace	Shri Devi Chattushringi Mandir	7.78
Aga Khan Palace	JW Marriott	7.95
Shreemant Dagdusheth Halwai Ganpati	Shri Devi Chattushringi Mandir	3.07
Shreemant Dagdusheth Halwai Ganpati	JW Marriott	3.35
Shri Devi Chattushringi Mandir	JW Marriott	0.82

Straight Line Distance

pra-sâmi

26

Abstraction

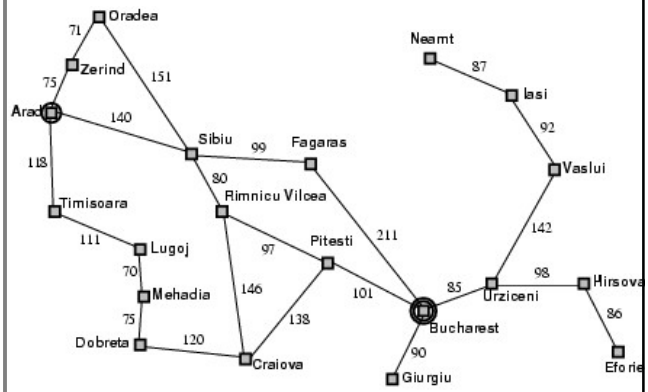
- While planning a day in Pune, we focus on the destinations and time it will take:
  - En route
  - At the destination
- Besides the four components (Initial state, Actions, Successor function, Goal test) for problem formulation
  - Anything else?
- We are not worried about other aspect of the journey
  - Cab? Which one? How many red lights? Cops on the way? etc.
- Abstraction
  - The process to take out the irrelevant information
  - Leave the most essential parts to the description of the states (Remove detail from representation)
  - Conclusion: Use the most important parts that are contributing to search only

pra-sâmi

27

Searching For Solutions

- On holiday in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest
- Formulate goal:
  - Be in Bucharest
- Formulate problem:
  - States: various cities
  - Actions: drive between cities
- Find solution:
  - Sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

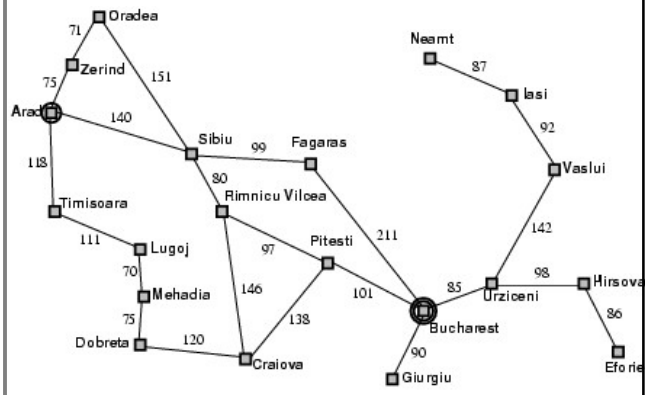


pra-sâmi

28

Single-state problem formulation

- A problem is defined by four items:
  - Initial state e.g., "at Arad“
  - Actions or successor function S(x) = set of action–state pairs
    - e.g., S(Arad) = {<Arad → Zerind, Zerind >, … }
  - Goal test, can be explicit or implicit .
    - Explicit in this case, e.g., x = "at Bucharest“
  - Path cost (additive)
    - e.g., sum of distances,
- A solution is a sequence of actions leading from the initial state to a goal state

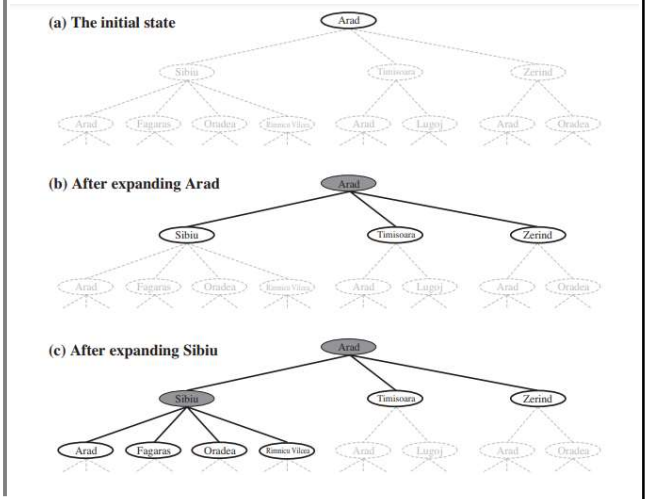


pra-sâmi

29

Search Tree

- Initial state
  - The root of the search tree is a search node
- Expanding
  - Applying successor function to the current state
  - Thereby generating a new set of states
- Leaf nodes
  - The states having no successors
- Fringe
  - Nodes that have not been expanded yet.



pra-sâmi

30

Components of a Node

- State: which state it is in
- Parent Node: from which node it is generated
- Action: which action applied to its parent-node to generate it
- Path Cost: the cost, g(n), from initial state to the node n itself
- Depth: number of steps along the path from the initial state

pra-sâmi

32

Search Strategies

Uninformed Search (Blind Search)	Informed Search (Heuristic Search)
Breadth First Search	Best First Search
Uniform Cost Search	Greedy Search
Depth First Search	A* Search
Depth First Limited Search	Beam Search
Iterative deepening depth-first search	
Bi-directional Search	
Constraint Satisfaction Search	

pra-sâmi

33

Informed Search / Heuristic Search

pra-sâmi

34

Heuristics and Algorithms

- A correct algorithm will find you the best solution given good data and enough time
  - It is precisely specified
- A heuristic gives you a workable solution in a reasonable time
  - It gives a guided or directed solution

pra-sâmi

35

Heuristic Search

- A heuristic is a rule or principle used to guide a search
  - It provides a way of giving additional knowledge of the problem to the search algorithm
  - Must provide a reasonably reliable estimate of how far a state is from a goal, or the cost of reaching the goal via that state
- Heuristic search is also known as informed search
- Important aspect: formation of heuristic function h(n)
  - A way of calculating or estimating such distances/cost
- Distance: heuristic function can be straight line distance (SLD)

Locations		SLD (km)
From	to	
Airport	Aga Khan Palace	2.87
Airport	Shreemant Dagdusheth Halwai Ganpati	8.80
Airport	Shri Devi Chattushringi Mandir	9.32
Airport	JW Marriott	9.73
Aga Khan Palace	Shreemant Dagdusheth Halwai Ganpati	6.31
Aga Khan Palace	Shri Devi Chattushringi Mandir	7.78
Aga Khan Palace	JW Marriott	7.95
Shreemant Dagdusheth Halwai Ganpati	Shri Devi Chattushringi Mandir	3.07
Shreemant Dagdusheth Halwai Ganpati	JW Marriott	3.35
Shri Devi Chattushringi Mandir	JW Marriott	0.82

pra-sâmi

36

Evaluation function

- There are an infinite number of possible heuristics
  - Criteria is that it returns an assessment of the point in the search
- If an evaluation function is accurate, it will lead directly to the goal
- More realistically, this usually ends up as “seemingly-best-search”
- Traditionally, the lowest value after evaluation is chosen as we usually want the lowest cost or nearest

pra-sâmi

37

Heuristic evaluation functions

- Estimate of expected utility value from a current position
  - E.g. value for pieces left in chess
  - Way of judging the value of a position
- Humans have to do this as we do not evaluate all possible alternatives
  - These heuristics usually come from years of human experience
- Performance of a game playing program is very dependent on the quality of the function

38

Heuristics for the 8-puzzle

- Number of tiles out of place (h1)
- Manhattan distance (h2)
  - Sum of the distance of each tile from its goal position
  - Tiles can only move up / down or left/right → city blocks

0	1	2
3	4	5
6	7	

39 H1 and H2 Functions

Goal state

0	1	2
3	4	5
6	7	

Current state

0	1	2
3	4	5
6		7

h_1=1
h_2=1

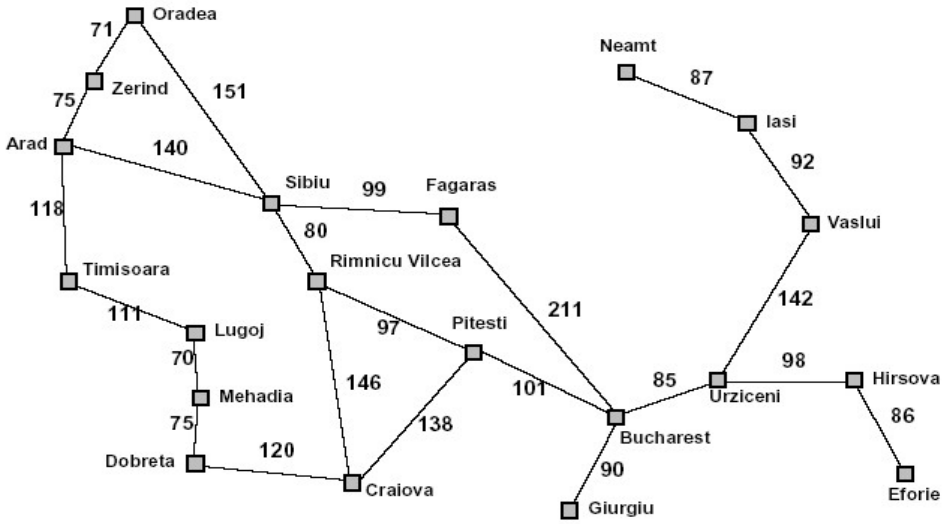
Current state

0	2	5
3	1	7
6		4

h_1=5
h_2=0+1+1+0+2+1+0+2=7

pra-sâmi

40 Heuristic Search : Heuristic Function



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

pra-sâmi

41

Heuristic Search : Best-First Search

pra-sâmi

42

Heuristic Search: Best-First Search

- Idea: use an evaluation function f(n) for each node
  - f(n) provides an estimate for the total cost.
  - Expand the node n with smallest f(n).
- Implementation:
  - Order the nodes in the fringe (frontier) increasing order of cost.
- Special cases:
  - Greedy best-first search
  - A* search

pra-sâmi

43

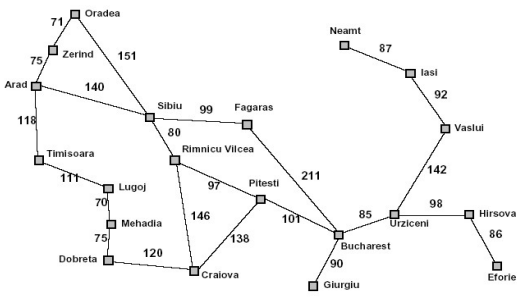
Heuristic Search: Greedy-Best Search

- Tries to expand the node that is closest to the goal
- Evaluates using only heuristic function :
  - f(n) = h(n)
- Possibly lead to the solution very fast
- Problem :
  - Can end up in sub-optimal solutions (doesn't take notice of the distance it travels).
- Complexity and time: $O(b^m)$
- Complete & optimal : No (stuck in infinite loop)

pra-sâmi

44

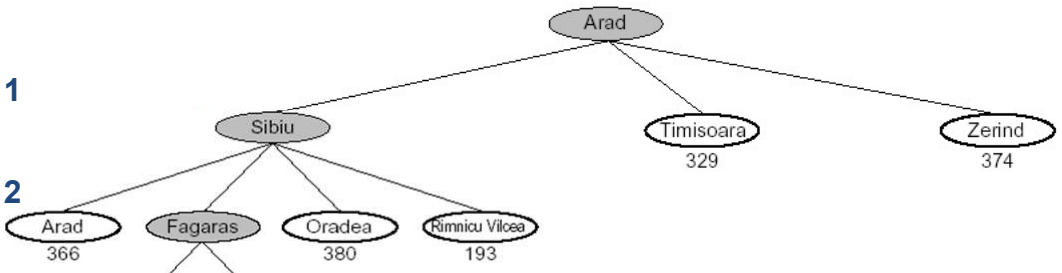
Heuristic Search : Greedy-Best Search

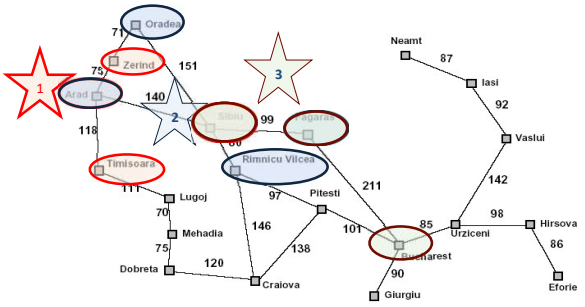


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

45

Heuristic Search : Greedy-Best Search





Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

46

Heuristic Search : A* Algorithm

pra-sâmi

47

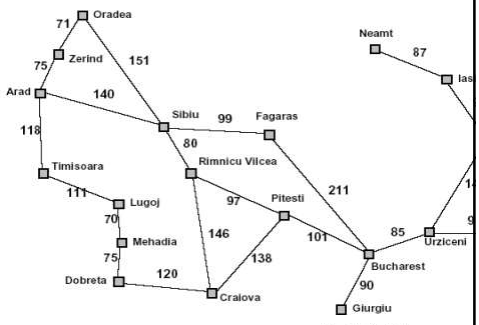
Heuristic Search : A* Algorithm

- Widely known algorithm – (pronounced as “A star” search)
- Evaluates nodes by combining g(n) “cost to reach the node” and h(n) “cost to get to the goal”
- f(n) = g(n) + h(n), f(n) ➔ estimated cost of the cheapest solution
- Complete and optimal : since evaluates all paths
- Time ? : a bit time consuming
- Space ? : lot of it!

pra-sâmi

48

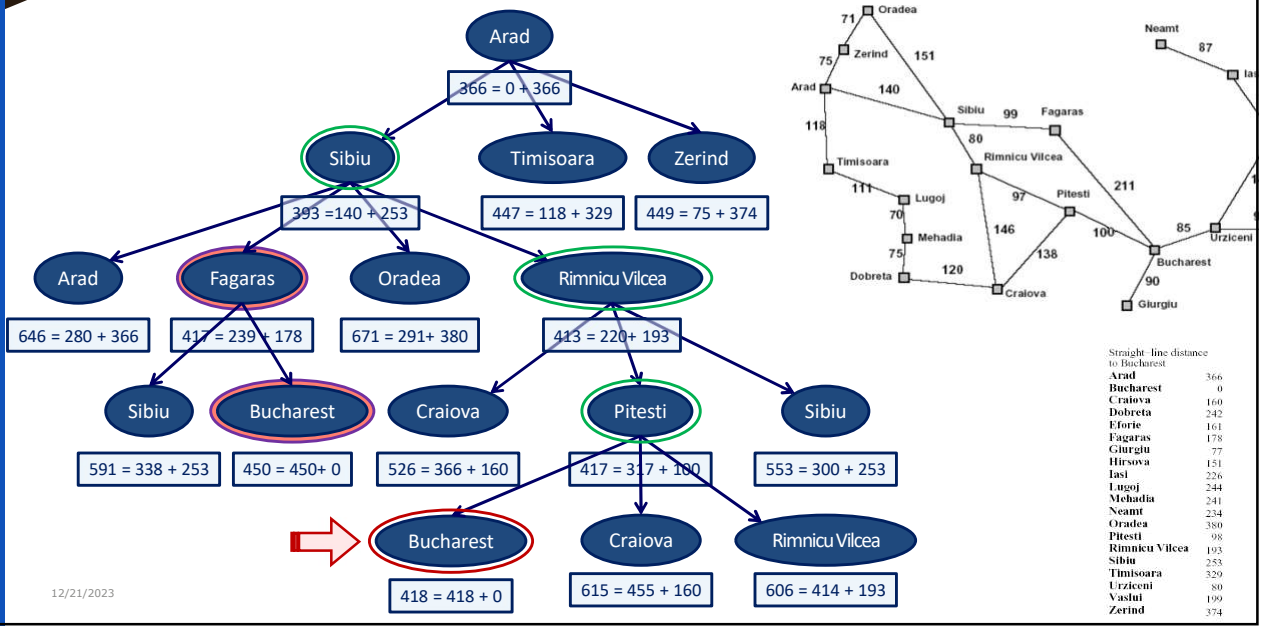
Heuristic Search : A* Algorithm



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

49

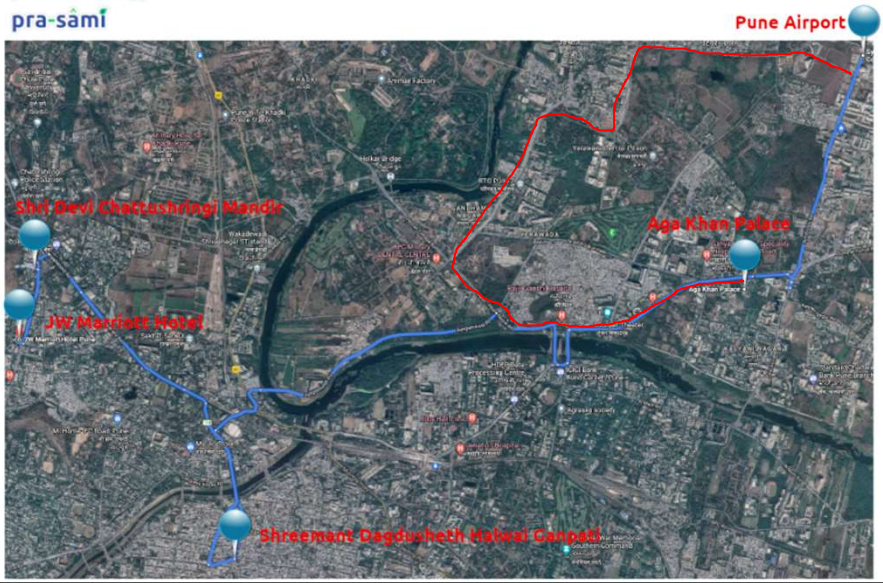
Heuristic Search : A* Algorithm



50

A* Algorithm : Memory Hungry

Pramod Sharma
pramod.sharma@prasami.com
pra-sâmi



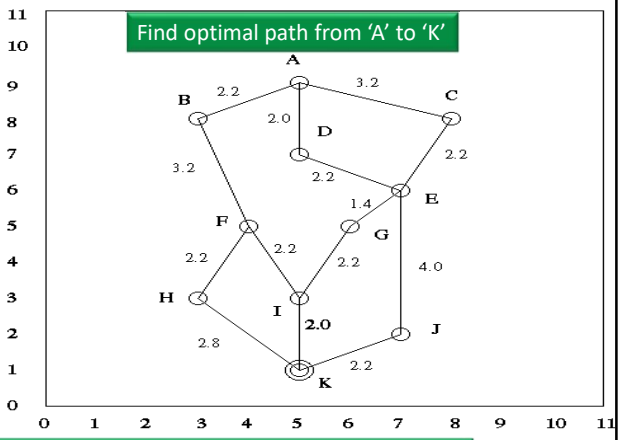
pra-sâmi

51

A* Practice Assignment

Course & Batch : _____
Roll No: _____
Name: _____
Date of submission:_____

Node	Coordinates	SL Distance to K
A	(5,9)	8.0
B	(3,8)	7.3
C	(8,8)	7.6
D	(5,7)	6.0
E	(7,6)	5.4
F	(4,5)	4.1
G	(6,5)	4.1
H	(3,3)	2.8
I	(5,3)	2.0
J	(7,2)	2.2
K	(5,1)	0.0



Answer to include
- Optimal Path
- Open and close list
- g(n) : cost to reach the node
- h(n) : cost to get to the goal
- And f(n) = g(n) + h(n), : estimated cost of the cheapest solution

pra-sâmi

52

pra-sâmi

53

Part II

pra-sâmi

54

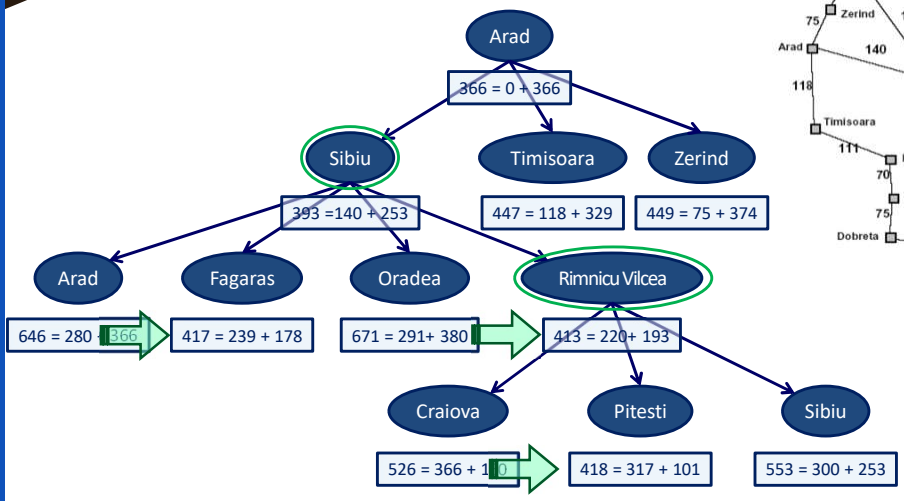
Memory Bound Search - RBFS

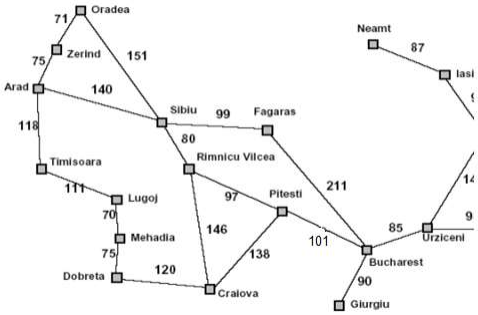
- Recursive Best First Search is similar to A* algorithm
  - Both are recursive
  - Difference
    - A* keeps all nodes in memory
    - RBFS keeps current path and sibling nodes only
  - When to stop search
    - It no longer looks good
  - Forget the sub tree to save space

pra-sâmi

55

Memory Bound : Recursive Best First Search

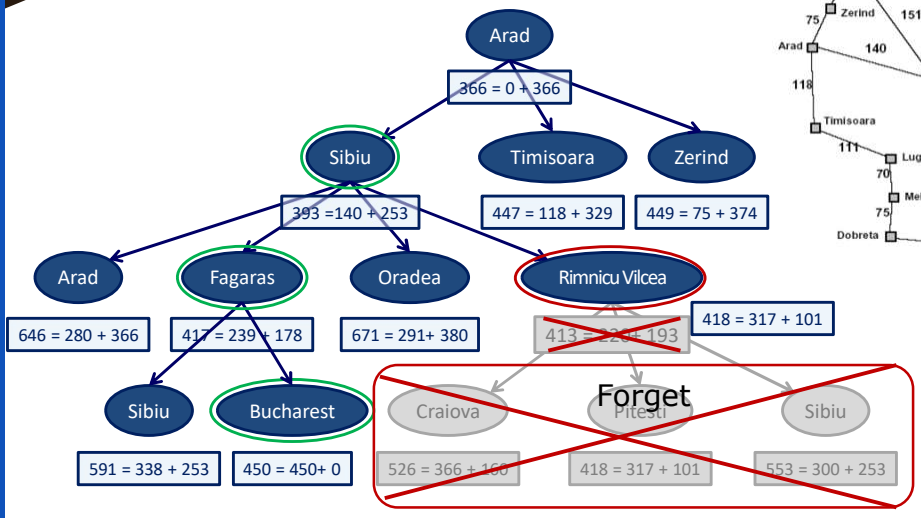


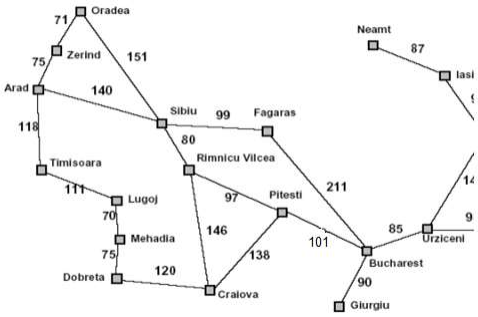


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

56

Memory Bound : Recursive Best First Search

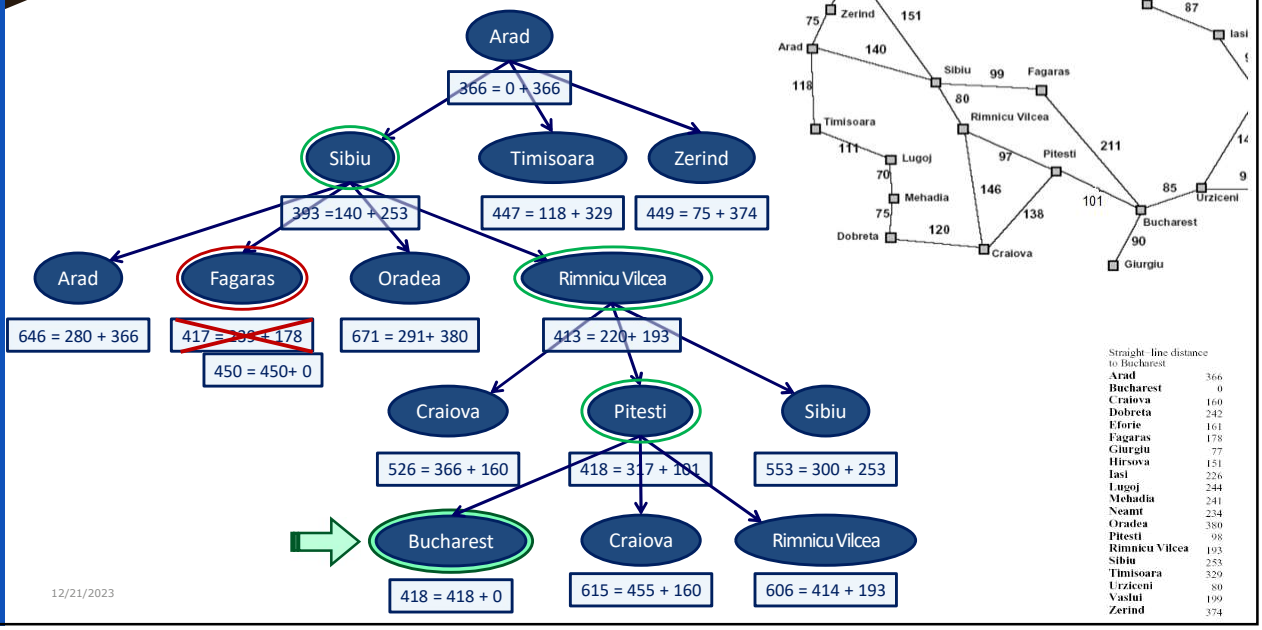




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

57

Memory Bound : Recursive Best First Search



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

58

Uninformed Search

pra-sâmi

59

Informed search

- Guided search
- Search with knowledge
- Quick Solution
- Less Complex
- Best First Search, Greedy Best search, A*, etc
- A cleverer strategy that searches toward the goal, based on the information from the current state so far

pra-sâmi

61

Uninformed Search

No prior knowledge

Time consuming

Search without information

More complex

Two forms namely;
- Breadth first
- Depth first

No information about:
- The number of steps
- The path cost from the current state to the goal

pra-sâmi

62

Uninformed Search Strategies

- Breadth-first search
  - Uniform cost search
- Depth-first search
  - Depth-limited search
  - Iterative deepening search
- Bidirectional search

pra-sâmi

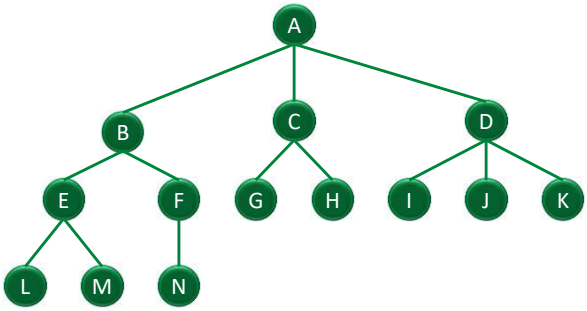
63

Breadth-first Search

- Breadth-first search is the most common search strategy for traversing a tree or graph
  - Searches breadthwise in a tree or graph
- It's a FIFO technique
  - Implements a queue
  - The root node is expanded first; Thereafter children of root node; Then their successors and so on…
- Level search technique
  - Moves level by level
- New nodes are getting added to the queue
- Goal test is applied to each node when it is generated rather than when it is selected for expansion
- For graph search, discards any new path to a state already in the frontier or explored set
  - Breadth-first search always has the shallowest path to every node on the frontier

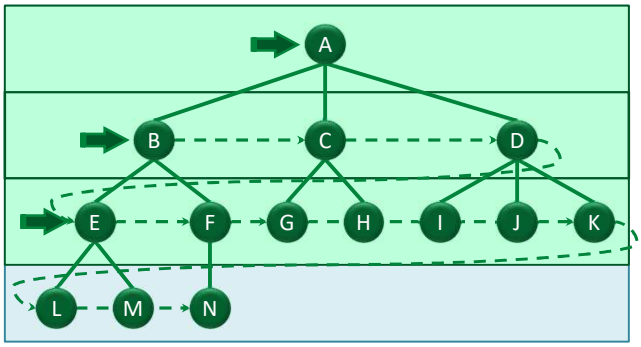
pra-sâmi

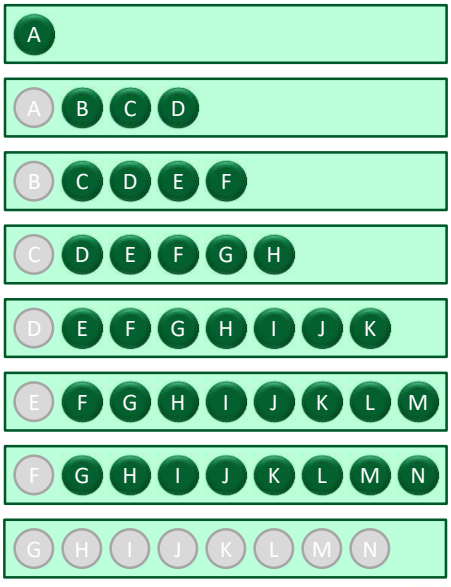
64 Breadth-first Search



pra-sâmi

65 Breadth-first Search





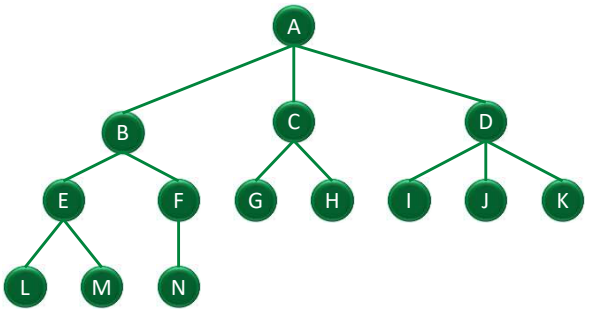
All leaf nodes…

pra-sâmi

66

Breadth-First Search

- Complete
  - Has to search all nodes
  - Finds the solution eventually
- Optimal: yes, if step cost is 1
- Time Complexity : O(b^d)
  - b: branch factor
  - d : depth
- Space Complexity:
  - Memory size of frontier which is O (b^d)



pra-sâmi

67

Breadth-first Search – Pseudo Code

- Create two empty **queues**
- Start from the initial node and add it to the ordered open queue
- Following steps are repeated until the final node or endpoint is reached
  - If the open queue is empty exit the loop and return a False statement which says that the final node cannot be reached
  - Select the top node in the open queue and move it to the closed queue while keeping track of the parent node
  - If the node removed is the endpoint node return a True statement meaning a path has been found and moving the node to the closed queue
  - However if it is not the endpoint node then list down all the neighboring nodes of it and add them to the open queue

pra-sâmi

70

Performance Heavy

- An exponential complexity bound such as O (b^d) is scary
- Table with branching factor b = 10,
  - 1 million nodes can be generated per second
  - A node requires 1000 bytes of storage
- Two major issues
  - The memory requirements are a bigger problem for breadth-first search than is the execution time
  - 13 days for a problem with search depth 12 may still be ok; but One Petabyte of memory
  - Time is still a major factor
  - If a problem has a solution at depth 16, then it will take about 350 years
  - Use current uninformed methods for any small instances

Depth	Nodes	Time	Memory
2	110	0.11 mili sec	107 KB
4	11110	11 mili sec	10.6 MB
6	10^6	1.1 sec	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 hour	10 TB
12	10^{12}	13 days	1 Peta B
14	10^{14}	3.5 years	99 Peta B
16	10^{16}	350 years	10 Exa B

Estimated values on modern laptops

pra-sâmi

71

Advantage - Disadvantages

Advantages	Disadvantage
BFS will provide a solution if any solution exists.	It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.	BFS needs lots of time if the solution is far away from the root node.

pra-sâmi

72

Uniform Cost Search

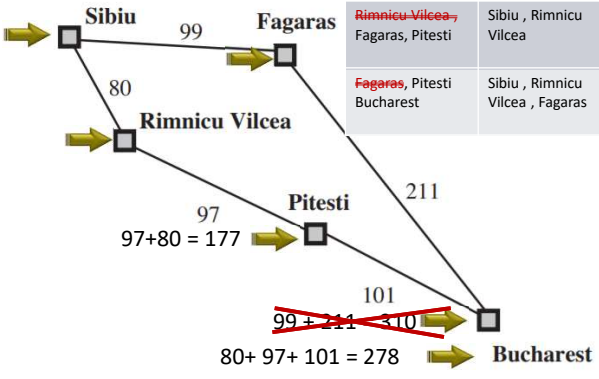
- Breadth-first finds the shallowest goal state
  - But not necessarily be the least-cost solution
  - Work only if all step costs are equal
- Uniform cost search
  - Modifies breadth-first strategy
    - By always expanding the lowest-cost node
  - The lowest-cost node is measured by the path cost g(n)
- The goal test is applied to a node when it is selected for expansion

pra-sâmi

75

Back to Romania

- The successors of Sibiu are Rimnicu Vilcea and Fagaras,
- The least-cost node, Rimnicu Vilcea, is expanded next adding Pitesti with cost 177
- The least-cost node is now Fagaras
  - Adding Bucharest with cost 99 + 211 = 310
  - A goal node has been generated
- Uniform-cost search keeps going
- Choosing Pitesti adds a second path to Bucharest with cost 80+ 97+ 101 = 278.
- New path is better than the old one;



Open	Close
Sibiu	
~~Sibiu~~ , Rimnicu Vilcea , Fagaras,	Sibiu
~~Rimnicu Vilcea,~~ Fagaras, Pitesti	Sibiu , Rimnicu Vilcea
~~Fagaras~~, Pitesti Bucharest	Sibiu , Rimnicu Vilcea , Fagaras

pra-sâmi

76

Uniform Cost Search

- Expand least-cost unexpanded node
  - Uniform-cost search expands nodes in order of their optimal path cost
  - Search does not care about the number of steps a path has, but only about their total cost.
- Implementation:
  - fringe = queue ordered by path cost
- Equivalent to breadth-first if step costs all equal
- Complete? Yes
- Time? # of nodes with g ≤ cost of optimal solution, O($b^{(1+\text{ceiling}(C*/\varepsilon))}$)
  - Where C* be the cost of the optimal solution, and ε is positive constant (every action cost)
- Space? # of nodes with g ≤ cost of optimal solution, O($b^{(1+\text{ceiling}(C*/\varepsilon))}$)
- Optimal? Yes – nodes expanded in increasing order of g(n)

77

Uniform Cost Search

- When all step costs are the same,
  - Uniform-cost search is similar to breadth-first search,
- Except
  - That the breadth-first search stops as soon as it generates a goal
  - Whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost
  - Uniform-cost search does strictly more work by expanding nodes at depth d unnecessarily
- Advantages:
  - Uniform cost search is optimal because at every state the path with the least cost is chosen.
- Disadvantages:
  - It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop
    - E.g, a sequence of NoOp actions

78

Use of Breadth First Search

- Web Crawlers
- Social networking websites - for finding the people in the specified distance (depth)
- Torrenting/peer-to-peer network to look for neighboring computers
- GPS navigation systems can use it to find nearby locations

pra-sâmi

79

Depth First Search

pra-sâmi

80

Depth-first search

- Always expands one of the nodes at the deepest level of the tree
- Only when the search hits a dead end
  - Goes back and expands nodes at shallower levels
  - Dead end → leaf nodes but not the goal
- Backtracking search
  - Only one successor is generated on expansion
  - Rather than all successors
  - Lower memory

81

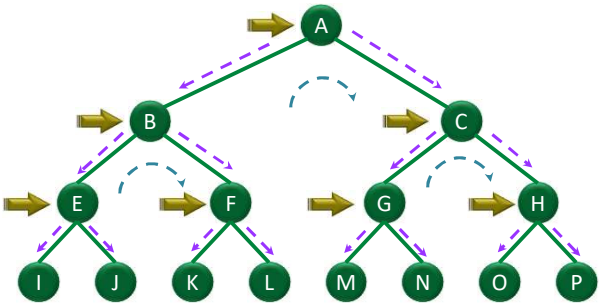
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - fringe = LIFO queue, i.e., put successors at front

82

Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - fringe = LIFO queue, i.e., put successors at front



pra-sâmi

84

Depth-first search

- Not complete
  - Because a path may be infinite or looping
  - Fails in infinite-depth spaces, spaces with loops
  - Needs modification to avoid repeated states along path
    - Complete in finite spaces
- Not optimal
  - It doesn't guarantee the best solution
- It overcomes
  - Time and space complexities
- Time? O(b^m) where m is the maximum depth of any node
  - Terrible if m is much larger than d
  - But if solutions are dense, may be much faster than breadth-first
- Space? O(b x m), i.e., linear space!

pra-sâmi

85

Depth-first search – Pseudo Code

- Create two empty **Stacks**
  - Open and closed
- Start from the initial node and add it to the ordered open stack
- Following steps are repeated until the final node or endpoint is reached
  - If the open stack is empty exit the loop and return a False statement which says that the final node cannot be reached
  - Select the top node in the open stack and move it to the closed stack while keeping track of the parent node
  - If the node removed is the endpoint node return a True statement meaning a path has been found and moving the node to the closed list
  - However if it is not the endpoint node then list down all the neighboring nodes of it and add them to the open stack

pra-sâmi

86

Use of Depth First Search

- Scheduling jobs from the given dependencies among jobs
- Find **a path** between two given points
- Solving puzzles fast when only one solution is needed

pra-sâmi

87

Adversarial Search

pra-sâmi

88

Games

- So far single Agent
- There can be multi agent problems
- Good reasoning problems
  - Formal rules
  - Non-trivial
  - Time tested
- Direct comparison to human performance!

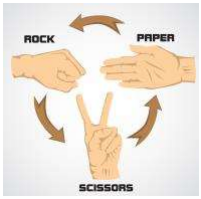
pra-sâmi

89

Type of Games

Sequence of moves

Opponent also moves



Both play simultaneously

- Rewards for each move
  - May be, may be not
- Defined win conditions

pra-sâmi

90

Search versus Games

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and Constraints Satisfaction Problems techniques can find optimal solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

pra-sâmi

91

Search versus Games

- Games – adversary
  - Unpredictable opponent
    - You make a move,
    - Opponent makes a move
  - Solution is strategy
    - Strategy specifies move for every possible opponent's reply
  - Optimality depends on opponent. Why?
  - Time limits
    - Large problems, huge state space, not enough compute power
    - Force an approximate solution
  - Evaluation function: evaluate "goodness" of game position
  - Examples: chess, checkers, Othello, backgammon

92

Adversarial Search

- Examine the problems that arise when we try to plan ahead in a world where other agents are planning against us
- A good example is board games
- Adversarial games, while much studied in AI, are a small part of game theory in economics

93

Games - Typical Assumptions

- Two agents whose actions alternate
  - May be more than 2
- Utility values for each agent are the opposite of the other
  - Creates the adversarial situation
  - You want to maximize your position so does your opponent
  - Both functions are interdependent
  - Simply put, you want to maximize your position and your opponent wants to minimize your position
- Fully observable environments
- In game theory terms: Zero-sum games of perfect information.
- Generalizes to stochastic games, multiple players, non-zero-sum, etc.
- Relaxation in these assumptions make it more complex

94

Types of Games

	Deterministic	Chance Moves
Perfect Information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect Information (Initial Chance Moves)	Skat, Battleship, Kriegspiel	Bridge, Poker, Scrabble, Blackjack

Not Considered: Physical games like tennis, croquet, ice hockey, etc.
(but see “robot soccer” http://www.robocup.org/)

95

Perfect and Imperfect Information Games

- A perfect information or Markov game is fully observed

Chess

Backgammon

Othello

Go

Checkers

- An imperfect information game is partially observed
  - Scrabble
  - Poker

pra-sâmi

96

Environment Types

- Turn-taking: Semi-dynamic
- Deterministic and non-deterministic

pra-sâmi

97

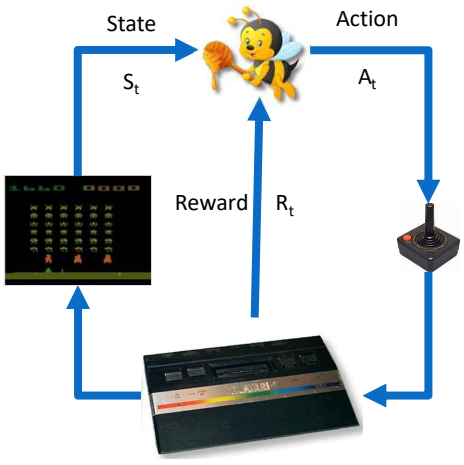
Two Fundamental Branches

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy a.k.a. deliberation, reasoning, introspection, pondering, thought, search

pra-sâmi

98

Atari Example: Reinforcement Learning

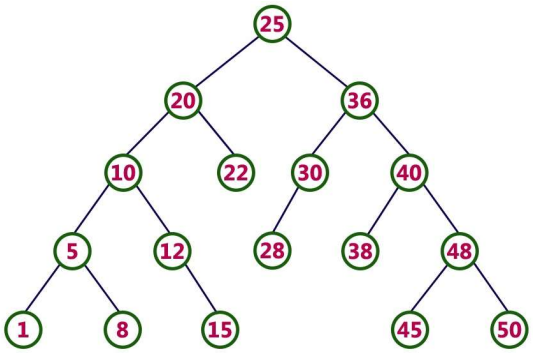


- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

pra-sâmi

99

Atari Example : Planning



- Rules of the game are known
- Can query emulator
  - Perfect model inside agent's brain?
- If I take action 'a' from state 's':
  - What would the next state be?
  - What would the score be?
- Plan ahead to find optimal policy
  - Tree search

pra-sâmi

100

Game Setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  - Winner gets reward, loser gets penalty.
- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move, state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states
- Tic-Tac-Toe : win (+1), lose (-1) and draw (0)
- Chess : win (+1), lose (-1) and draw ($\frac{1}{2}$)

pra-sâmi

101

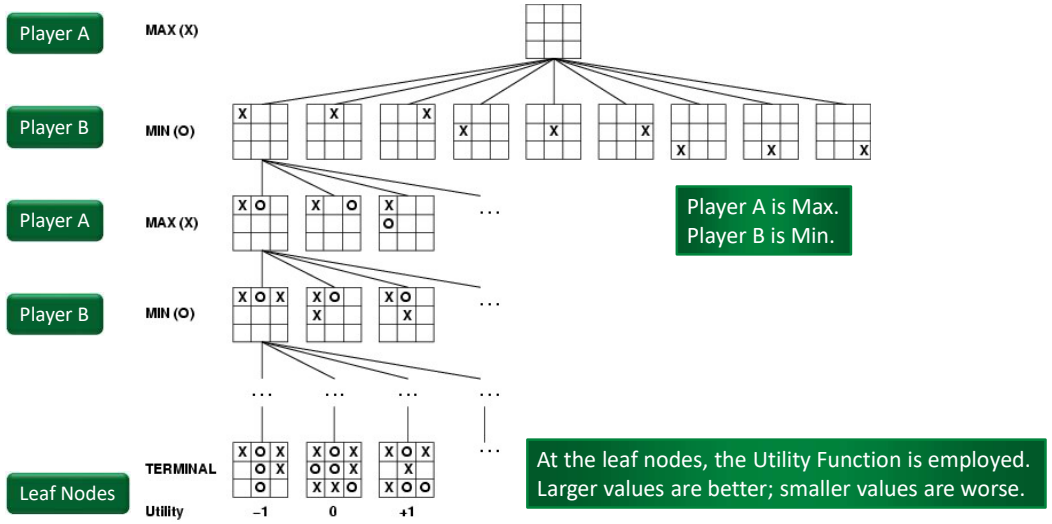
Two-Player Zero-Sum Games

- A two-player game has two (alternating) players
  - Say player white and player black
- A zero sum game has equal and opposite rewards for black and white
  - $R_w + R_b = 0$
- Both methods are applicable in such cases
  - Game tree search (i.e. planning)
  - Self-play reinforcement learning

pra-sâmi

102

Game Tree (2-player, Deterministic, Turns)



How do we search this tree to find the optimal move?

pra-sâmi

103

Minimax Terminology

- Move: a move by both players
- Ply : one players move; a half-move
- Utility Function : function applied to leaf node
- Backed-up values
  - For Max : the values of largest successor
  - For Min : the values of smallest successor
- Minimax Procedure:
  - Search down wards
  - At the bottom apply utility function
  - Back-up values all the way up to root node
  - Root selects the move

- Formal definition as a search problem:
  - Initial state: Set-up specified by the rules, e.g., initial board configuration of chess.
  - Player(s): Defines which player has the move in a state.
  - Actions(s): Returns the set of legal moves in a state.
  - Result(s, a): Transition model defines the result of a move.
  - Successor function: list of (move, state) pairs specifying legal moves
  - Terminal-Test(s): Is the game finished? True if finished, false otherwise.
  - Utility function(s, p): Gives numerical value of terminal state 's' for player p.

104

An Optimal Procedure: The Minimax Method

- Designed to find the optimal strategy for Max and find best move
  - Assuming an infallible MIN opponent
- Assumption: Both players play optimally!
- Definition of optimal play for MAX assumes MIN plays optimally:
  - Maximize worst-case outcome for MAX
- But if MIN does not play optimally, MAX will do even better

105

An Optimal Procedure: The Minimax Method

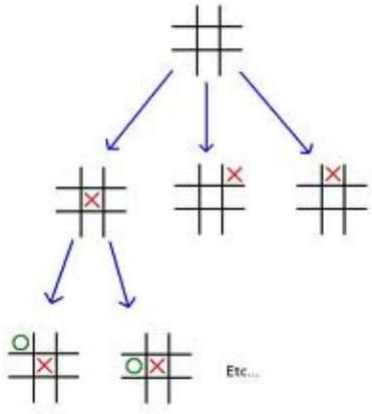
- Step 1. Generate the whole game tree, down to the leaves
- Step 2. Apply utility (payoff) function to each leaf
- Step 3. Back-up values from leaves through branch nodes:
  - a Max node computes the Max of its child values
  - a Min node computes the Min of its child values
- Step 4. At root: choose the move leading to the child of highest value

pra-sâmi

106

Minimax Search

- Minimax values can be found by depth-first game-tree search
- Introduced by Claude Shannon: Programming a Computer for Playing Chess
- Ran on paper!

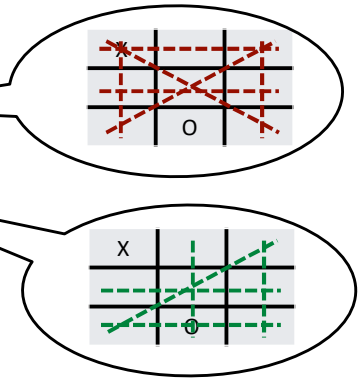


pra-sâmi

107

Applying Minimax to Tic-tac-toe

- The static heuristic evaluation function
- X has 6 possible win paths
- O has 5 possible win paths
- Therefore E(n) = 6 - 5 = 1

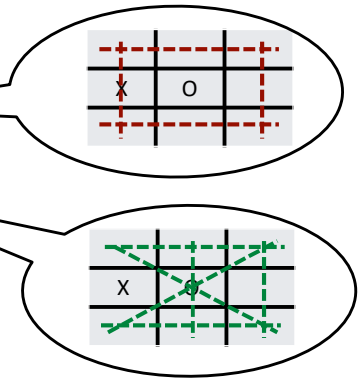


pra-sâmi

108

Applying Minimax to Tic-tac-toe

- The static heuristic evaluation function
- X has 4 possible win paths
- O has 6 possible win paths
- Therefore E(n) = 4 - 6 = -2

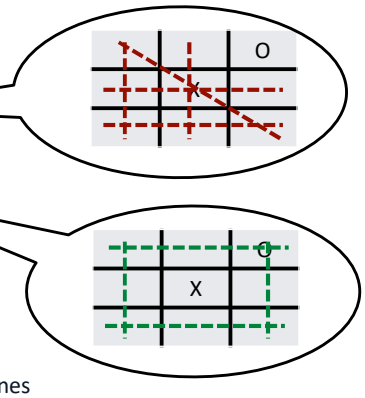


pra-sâmi

109

Applying Minimax to Tic-tac-toe

- The static heuristic evaluation function
- X has 5 possible win paths
- O has 4 possible win paths
- Therefore E(n) = 5 - 4= 1
- Heuristic in E(n) = M (n) – O (n)
  - M (n) : total of Max's possible winning lines
  - O (n) : total of Min's possible winning lines
  - E(n) : total Evaluation of the State

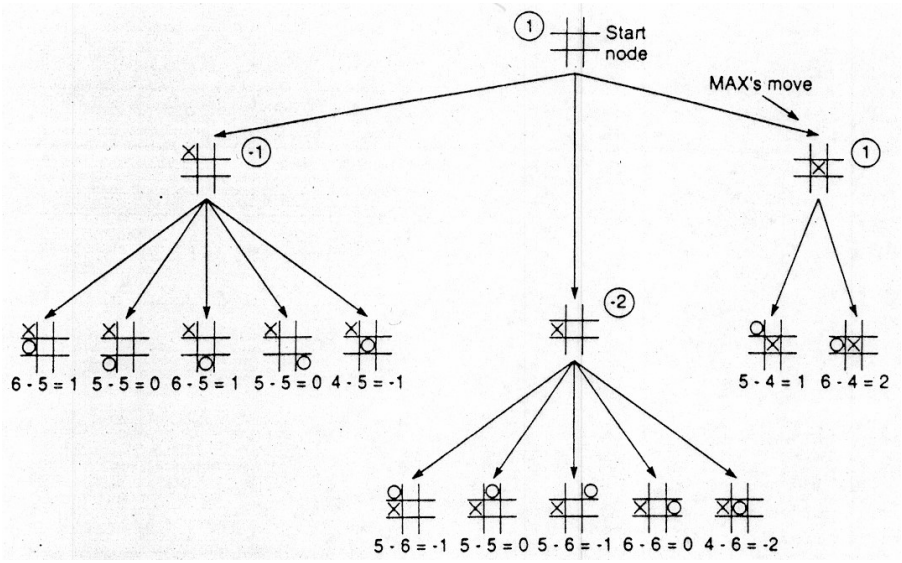


Heuristic measuring conflict applied to state of tic-tac-toe

pra-sâmi

110

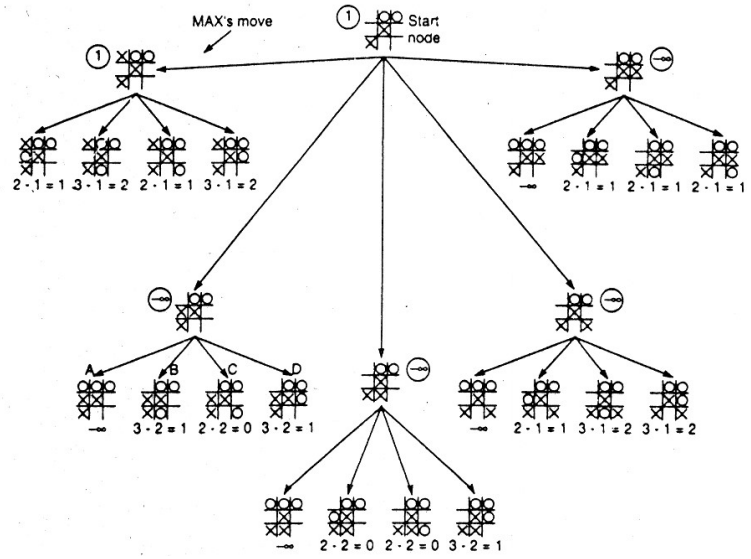
Two-ply Minimax Applied to the Opening Move of Tic-tac-toe



pra-sâmi

112

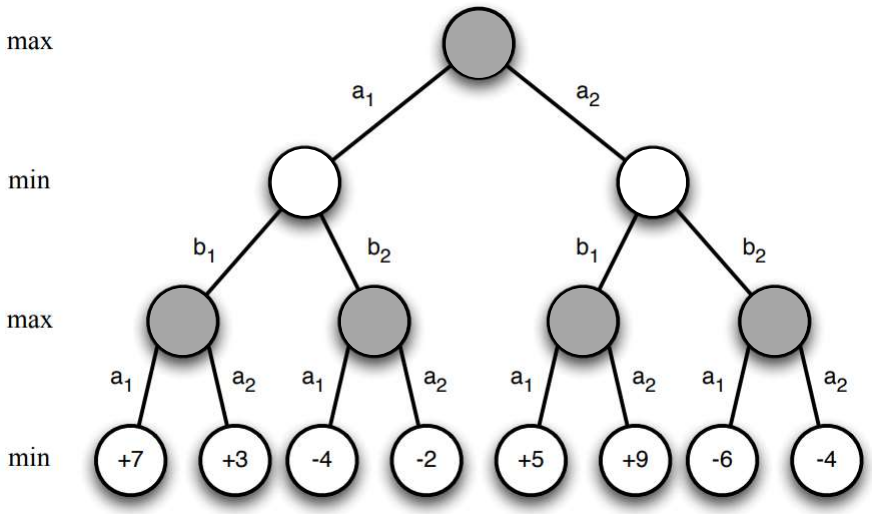
Two-ply Minimax Applied to X's Move Near End Game



pra-sâmi

113

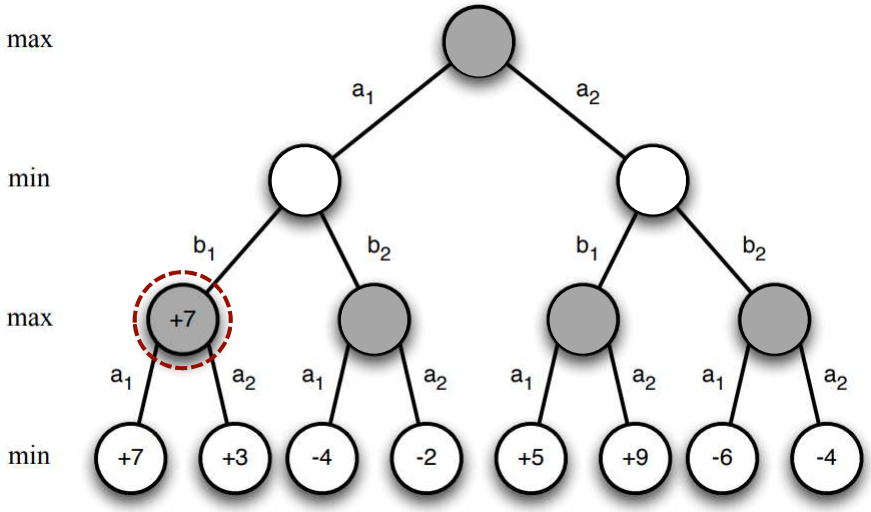
Minimax Search Example



pra-sâmi

114

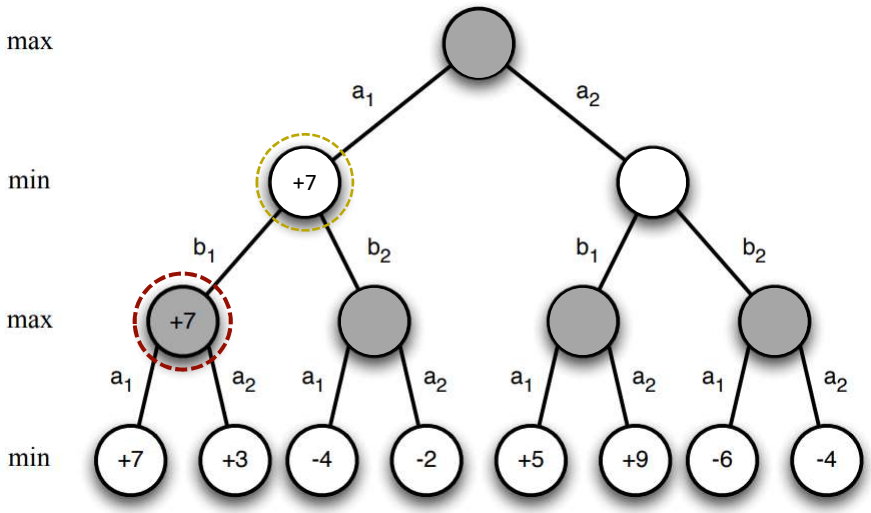
Minimax Search Example



pra-sâmi

115

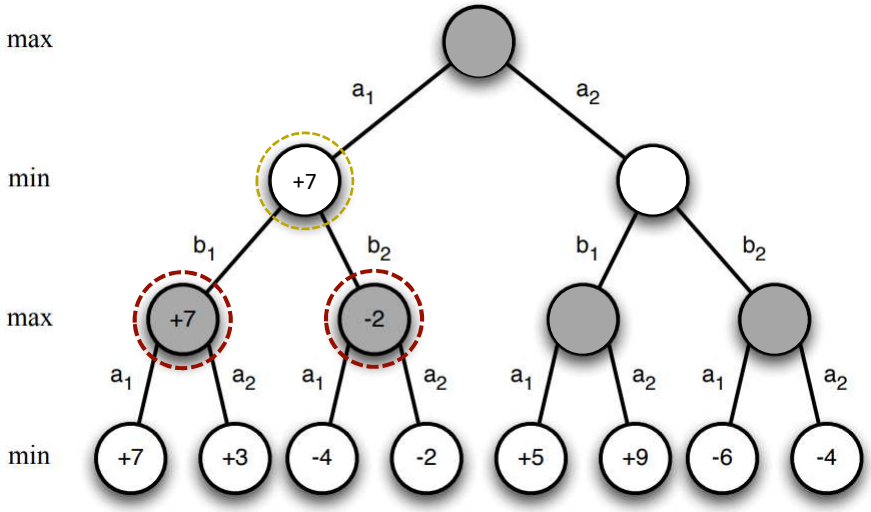
Minimax Search Example



pra-sâmi

116

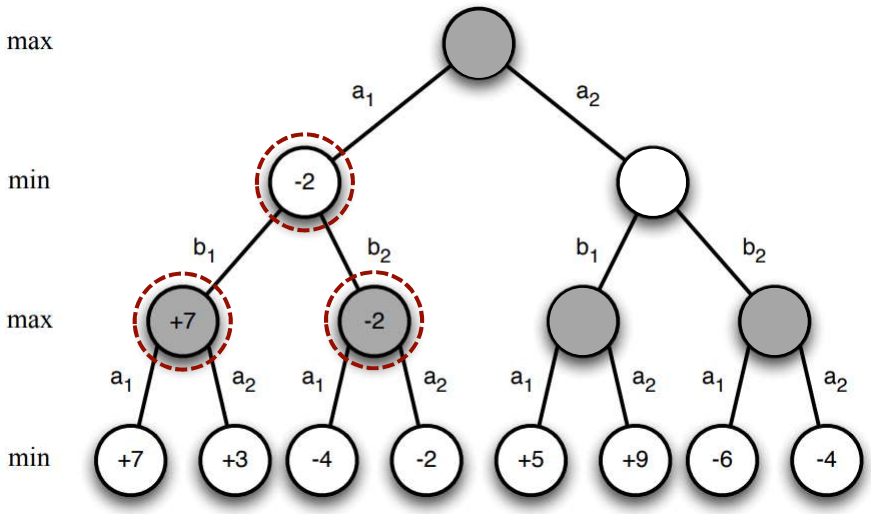
Minimax Search Example



pra-sâmi

117

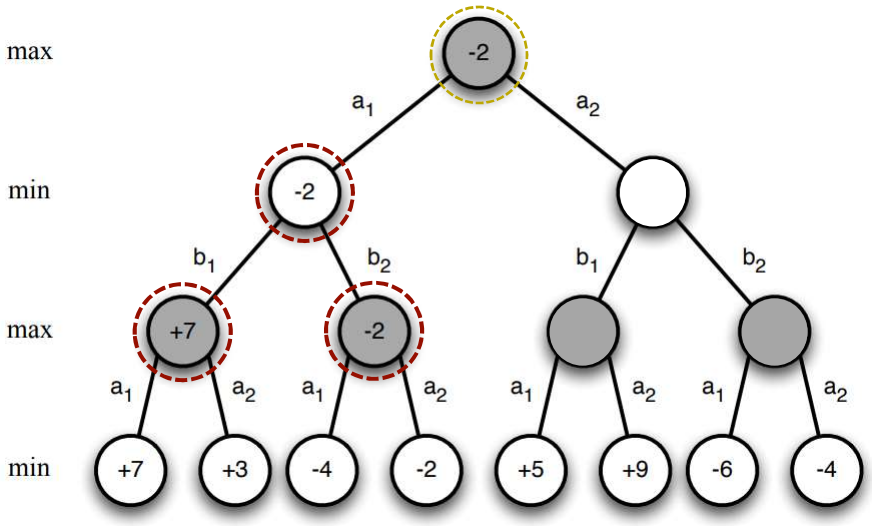
Minimax Search Example



pra-sâmi

118

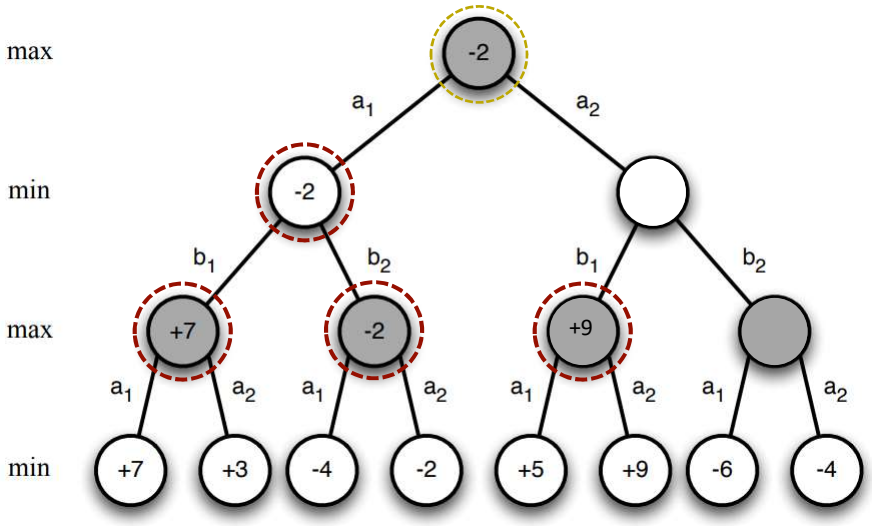
Minimax Search Example



pra-sâmi

119

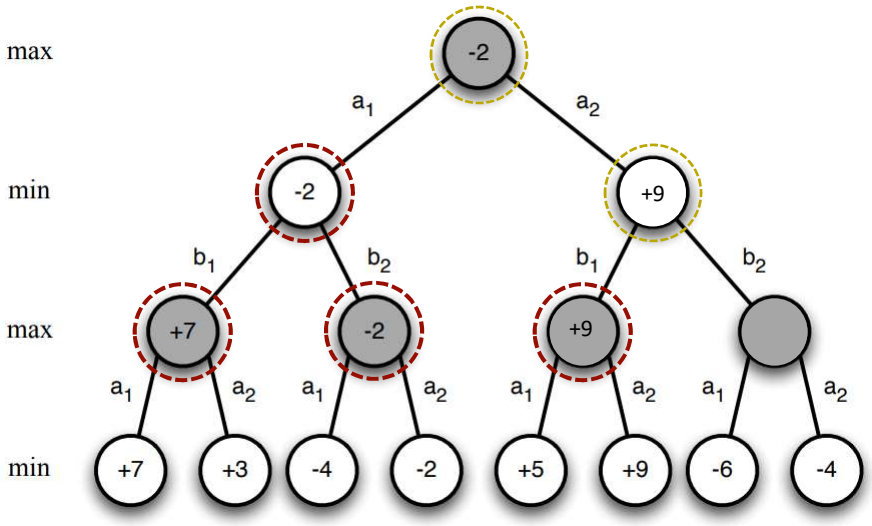
Minimax Search Example



pra-sâmi

120

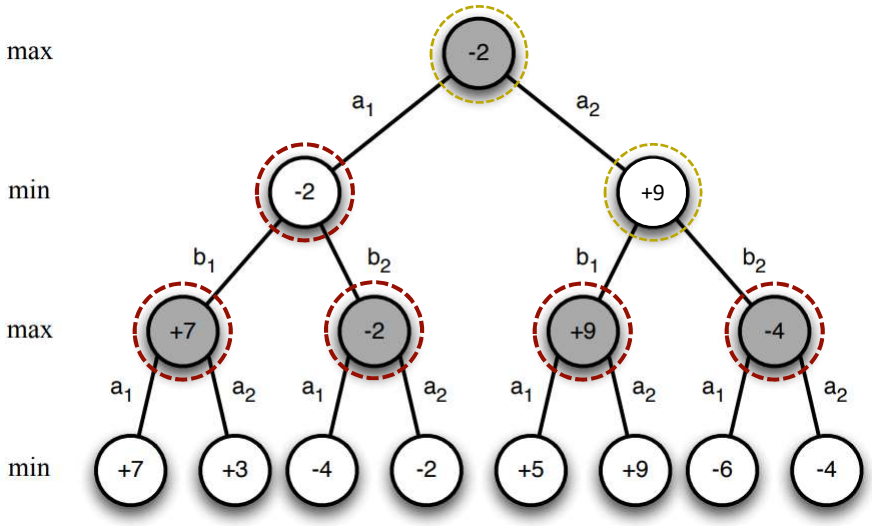
Minimax Search Example



pra-sâmi

121

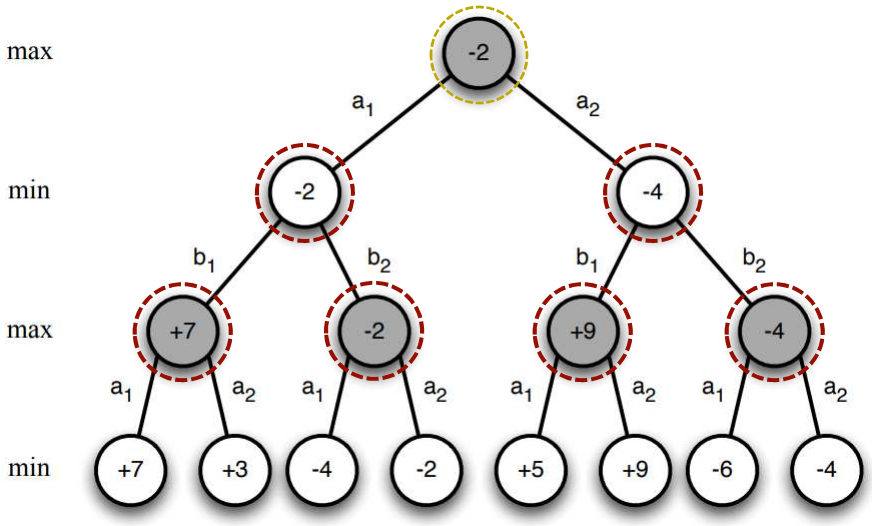
Minimax Search Example



pra-sâmi

122

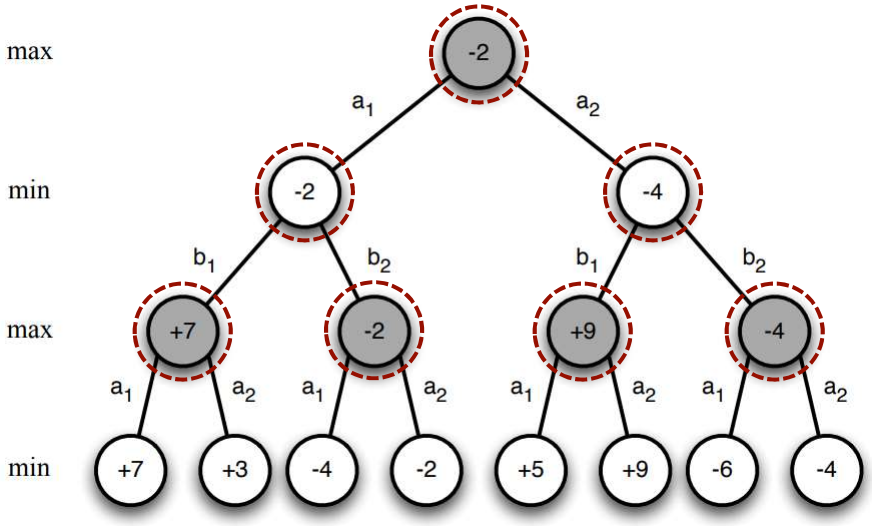
Minimax Search Example



pra-sâmi

123

Minimax Search Example



pra-sâmi

124

Limitation

- Time complexity of the search is O(b^d)
  - b : number of branches
  - d : depth of the tree
- Hence for the example with d = 3
  - Time complexity will be 2^3 = 8
- In regular game of chess ,
  - Average choices are 35,
  - Average game lasts for about 50 moves by a player ➔ Total 100 moves
- Hence complexity will be 35^{100} = 2.5516E+154

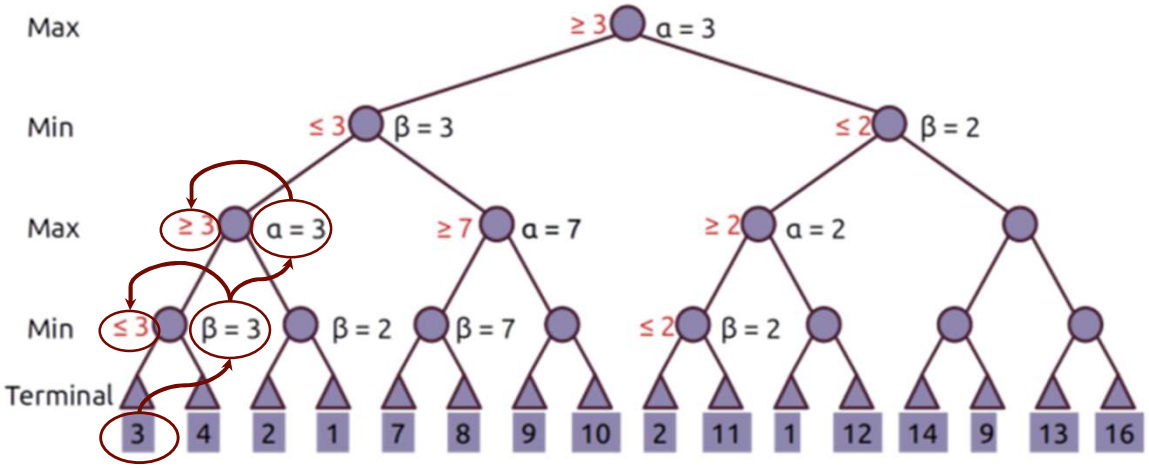
Minimax search is suitable for games like tic-tac-toe and not suitable for game of chess.

125

Solution to the Complexity Problem

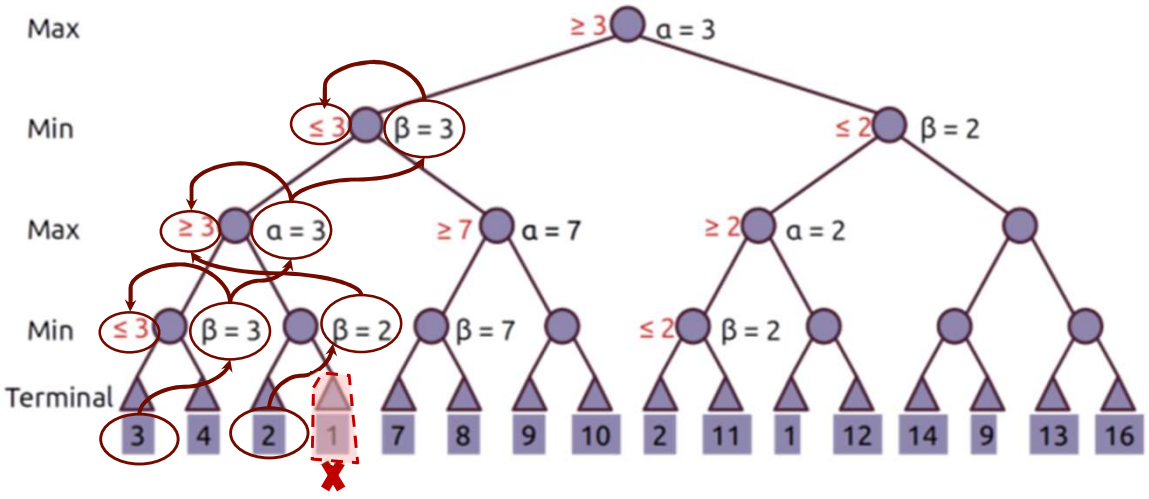
- Two solutions
- Dynamic pruning of redundant branches
  - Identify a provably suboptimal branch of the search tree before it is fully explored
  - Eliminate the suboptimal branch
  - It is called Alpha – Beta Pruning
  - Some branches will never be played by rational players since they include sub-optimal decisions for either player
- Early cutoff the search tree
- Use imperfect minimax value estimate for non-terminal states (positions)
  - α = highest-value choice that we can guarantee for MAX so far in the current subtree.
  - β= lowest-value choice that we can guarantee for MIN so far in the current subtree.
  - Update values of α and β during search and prunes remaining branches as soon as the value is known to be worse than the current α or β value for MAX or MIN respectively.

126 Alpha Beta Pruning



pra-sâmi

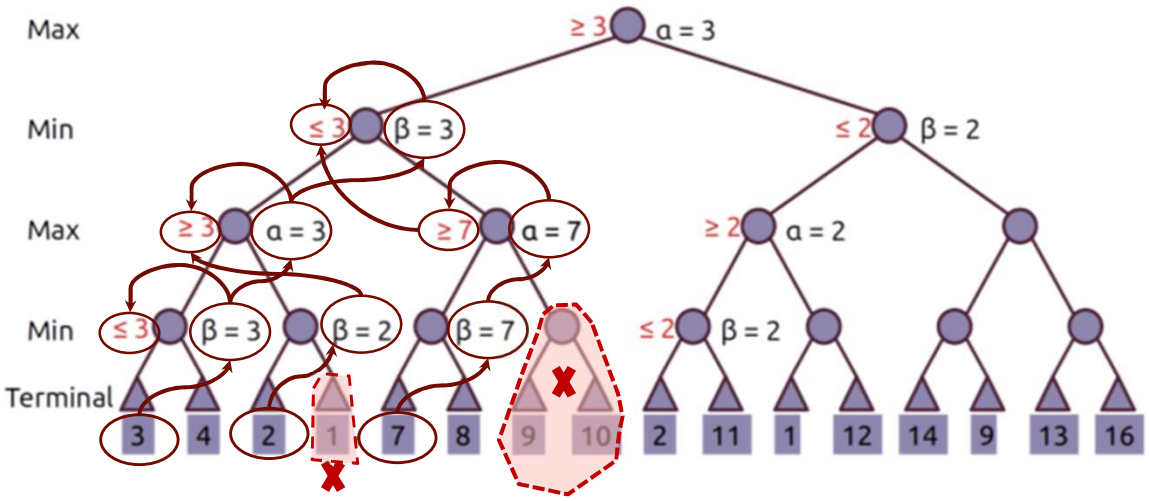
127 Alpha Beta Pruning



pra-sâmi

128

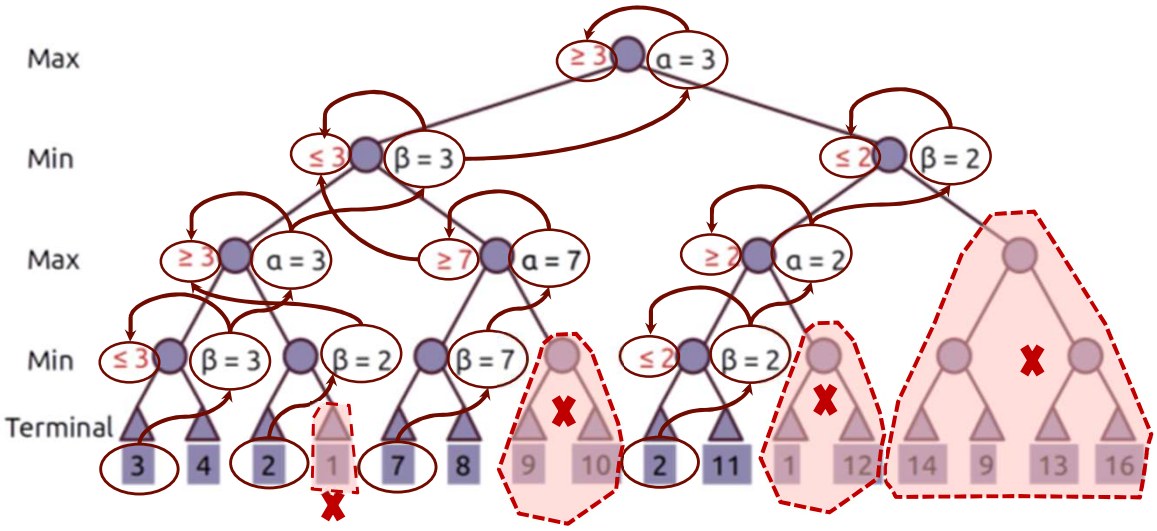
Alpha Beta Pruning



pra-sâmi

129

Alpha Beta Pruning



pra-sâmi

130

Effectiveness of Alpha-Beta Search

- Worst-Case
  - Branches are ordered so that no pruning takes place
  - Gives no improvement over exhaustive search
- Best-Case
  - Each player's best move is the left-most child (i.e., evaluated first)
  - In practice, performance is closer to best rather than worst-case
  - E.g., sort moves by the remembered move values found last time
  - E.g., expand captures first, then threats, then forward moves, etc.
  - E.g., run Iterative Deepening search, sort by value last iteration
- In practice often get O($b^{(\frac{d}{2})}$) rather than O(b^d)
  - This is the same as having a branching factor of sqrt(b),
    - $\sqrt{b}^d = b^{(\frac{d}{2})}$,i.e., we effectively go from b to square root of b
  - e.g., in chess go from b ~ 35 to b ~ 6
    - This permits much deeper search in the same amount of time

pra-sâmi

131

Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned
- Good move ordering improves effectiveness of pruning
- Repeated states are again possible
  - Store them in memory = transposition table

pra-sâmi

132

Depth - Limited Strategy

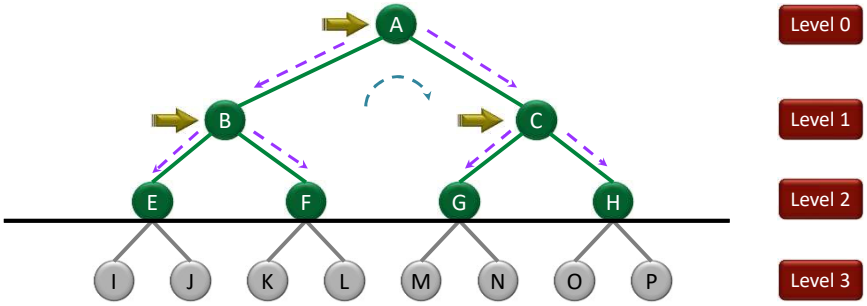
- It is depth-first search
  - With a predefined maximum depth
  - However, it is usually not easy to define the suitable maximum depth
  - Too small → no solution can be found
  - Too large → the same problems as Depth - First
- Anyway the search is
  - complete
  - but still not optimal
- Depth-first with depth cutoff k
  - maximal depth below which nodes are not expanded
  - Treat them as leaf node
- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

pra-sâmi

133

Depth - Limited Strategy

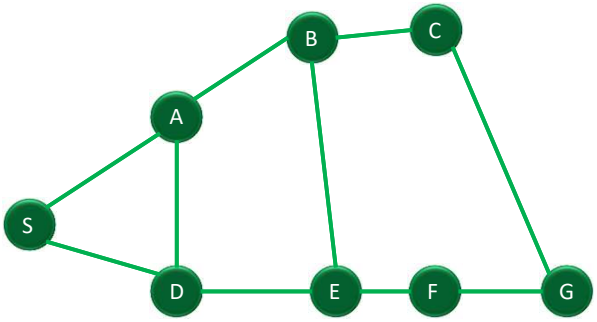
- Level 2



pra-sâmi

134

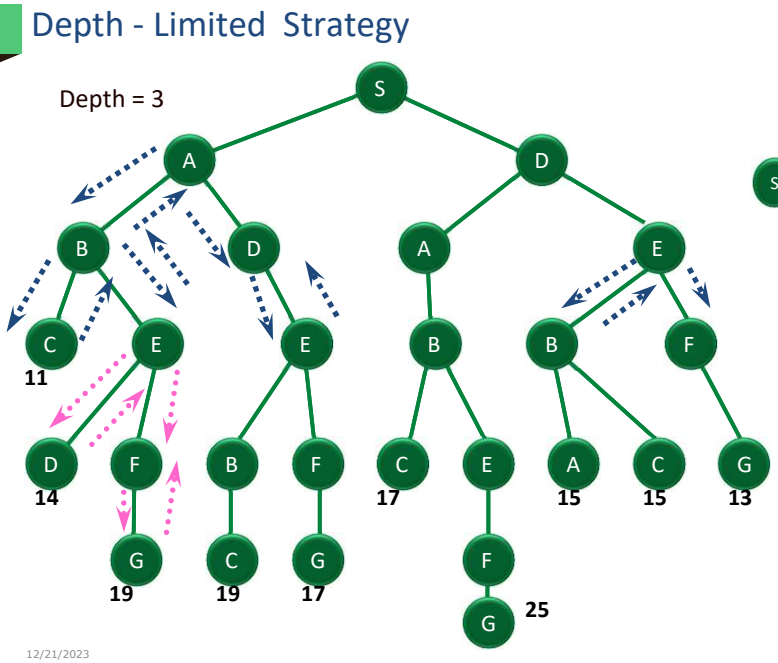
Depth - Limited Strategy

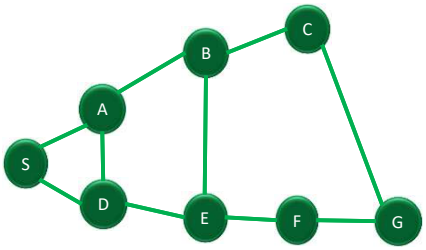


pra-sâmi

135

Depth - Limited Strategy





3

6

It also introduces an additional source of incompleteness if we choose <d, that is, the shallowest goal is beyond the depth limit

pra-sâmi

136

Depth - Limited Strategy

- Advantages:
  - Depth-limited search is Memory efficient.
- Disadvantages:
  - Depth-limited search also has a disadvantage of incompleteness.
  - It may not be optimal if the problem has more than one solution.
- Completeness:
  - DLS search algorithm is complete if the solution is above the depth-limit.
- Time Complexity:
  - Time complexity of DLS algorithm is O(b$^{\ell}$).
- Space Complexity: Space complexity of DLS algorithm is O(b×ℓ).
- Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

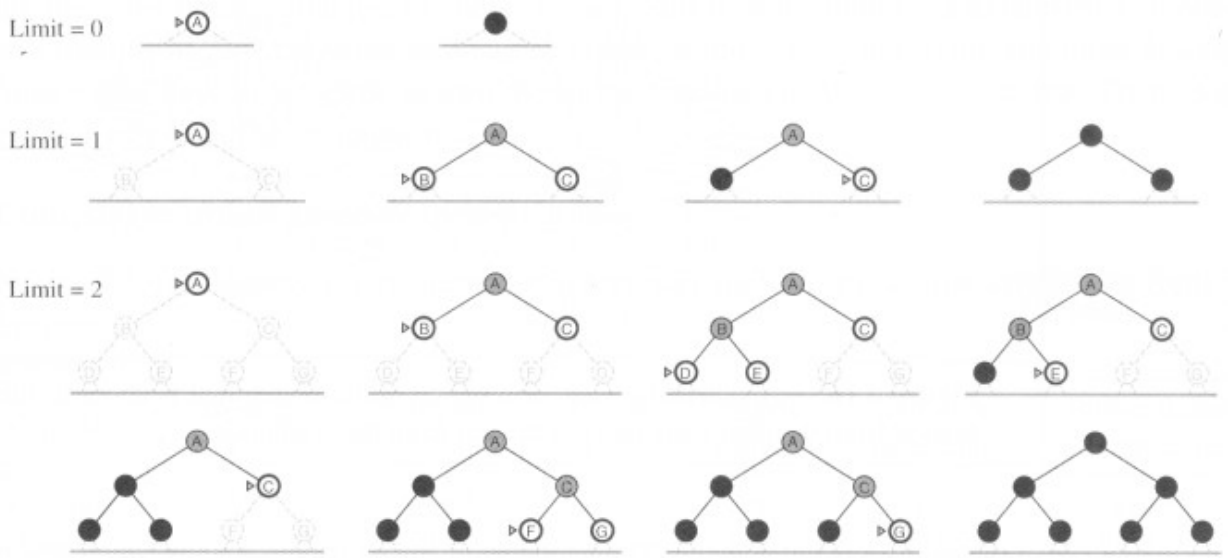
137

Iterative Deepening Search

- No choosing of the best depth limit
- It tries all possible depth limits:
  - First 0, then 1, 2, and so on
  - Combines the benefits of depth-first and breadth-first search

138

Iterative Deepening Search



pra-sâmi

139

Iterative Deepening Search

- Optimal : Yes, if step cost = 1
  - Otherwise use Iterative Lengthening search
- Complete : Yes
- Time and space complexities
  - Reasonable
- Time : $O(b^d)$
- Space: $O(b^d)$
- Suitable for the problem
  - Having a large search space
  - Depth of the solution is not known

pra-sâmi

140

Iterative Lengthening Search

- Iterative Deepening Search is using depth as limit
- Iterative Lengthening Searchis using path cost as limit
  - An iterative version for uniform cost search
  - Has the advantages of uniform cost search
    - while avoiding its memory requirements
  - but ILS incurs substantial overhead
    - compared to uniform cost search

pra-sâmi

141

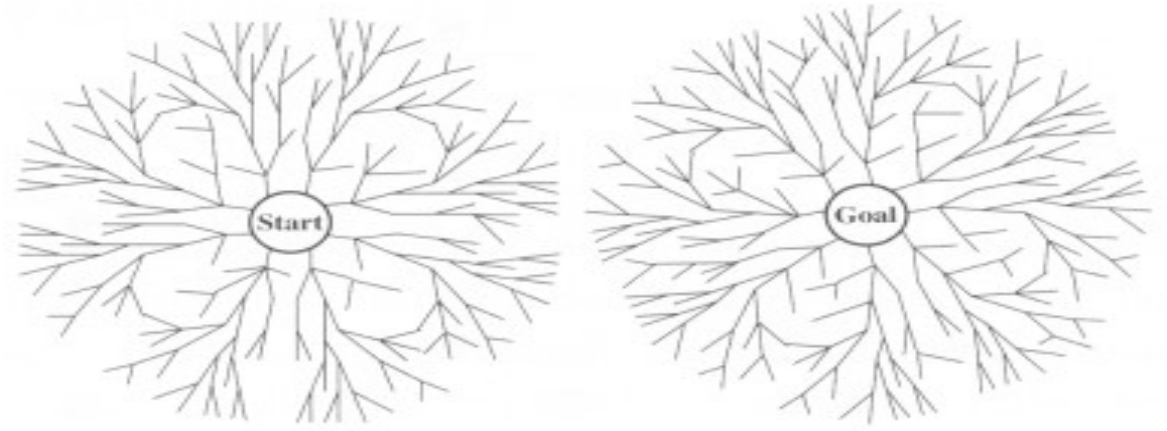
Bidirectional Search

- Run two simultaneous searches
  - One forward from the initial state another backward from the goal
  - Stop when the two searches meet
- However, computing backward is difficult
  - A huge amount of goal states
  - At the goal state, which actions are used to compute it?
  - Can the actions be reversible to computer its predecessors?

pra-sâmi

142

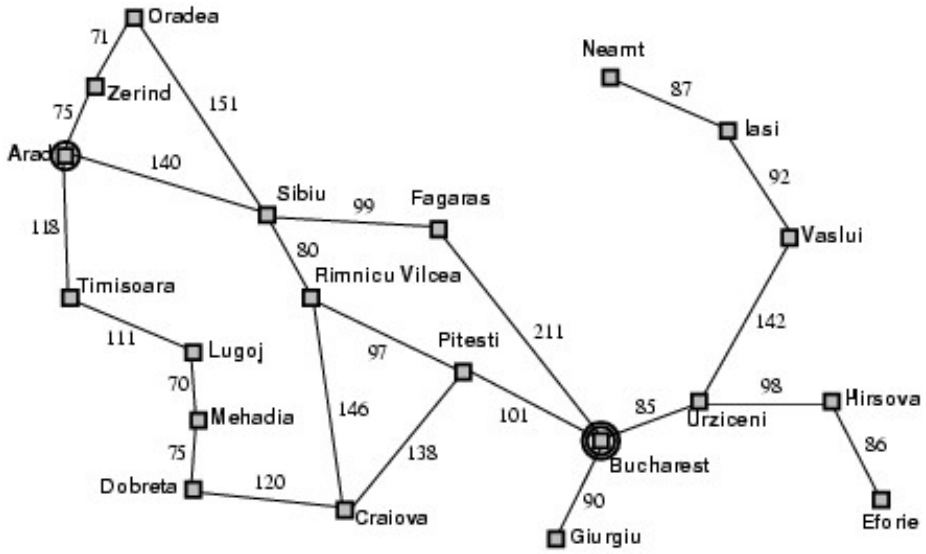
Bidirectional Search



pra-sâmi

143

Bidirectional Search

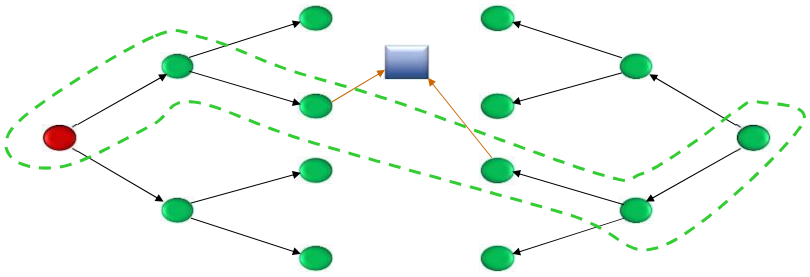


pra-sâmi

144

Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2

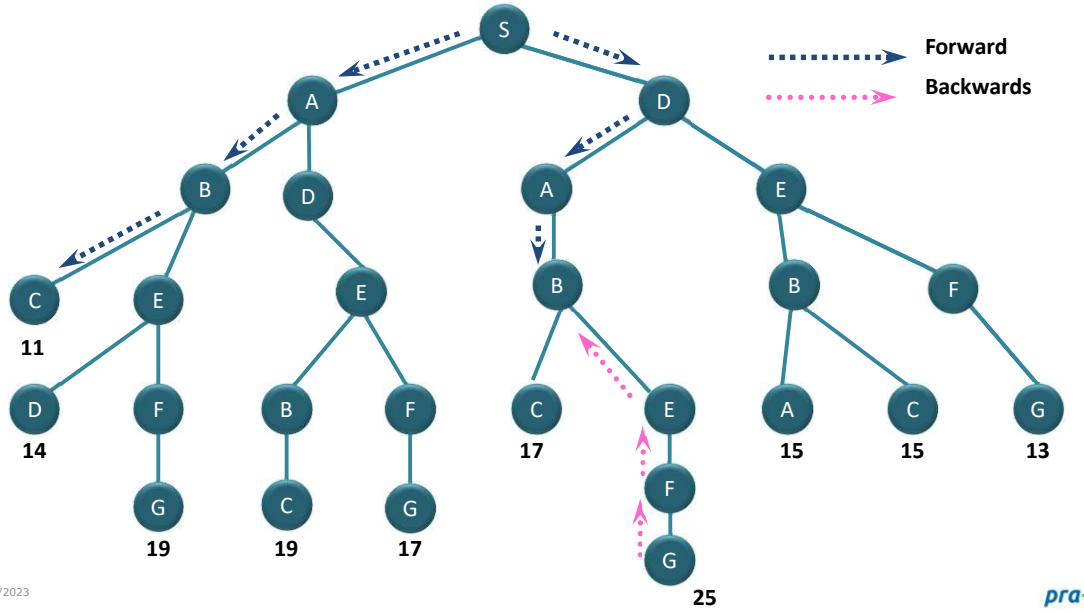


Time and space complexity = $O(b^{d/2}) << O(b^d)$

pra-sâmi

145

Bidirectional Search



pra-sâmi

146

Bidirectional Search is Challenging

- The reduction in time complexity makes bidirectional search attractive,
  - But how do we search backward? This is not as easy as it sounds.
- Let the predecessors of a state x be all those states that have x as a successor
  - Bidirectional search requires a method for computing predecessors
  - When all the actions in the state space are reversible, the predecessors of x are just its successors.
- What we mean by "the goal" in searching "backward from the goal."
  - For the 8-puzzle and for finding a route in Romania, there is just one goal state, so the backward search is very much like the forward search
- If there are several explicitly listed goal states
  - for example, the two dirt-free goal states in vacuum world then we can construct a new dummy goal state whose immediate predecessors are all the actual goal states
  - But if the goal is an abstract description, such as the goal that "no queen attacks another queen" in the n-queens problem, then bidirectional search is difficult to use

pra-sâmi

147

Comparing search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

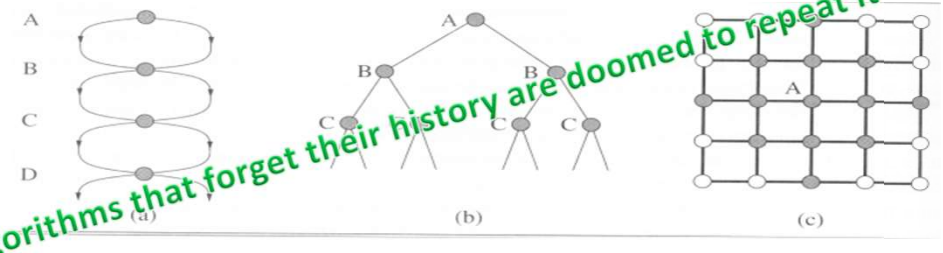
Figure 3.17 Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

pra-sâmi

148

Avoiding repeated states

- For all search strategies
  - There is possibility of expanding states that have already been encountered and expanded before, on some other path
  - May cause the path to be infinite → loop forever



Algorithms that forget their history are doomed to repeat it

pra-sâmi

149

Avoiding repeated states

- Try to remember all previously generated states
  - Space requirement goes up exponentially
  - Do not go back to parent in bidirectional graphs
  - May be do not go to ancestor
- Do not return to the state it just came from
  - Refuse generation of any successor same as its parent state
- Do not create paths with cycles
  - Refuse generation of any successor same as its ancestor states
- Do not generate any generated state
  - Not only its ancestor states, but also all other expanded states have to be checked against

pra-sâmi

150 Avoiding repeated states

- We then define a data structure
  - closed list:
  - a set storing every expanded node so far
  - If the current node matches a node on the closed list, discard it.

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```


pra-sâmi

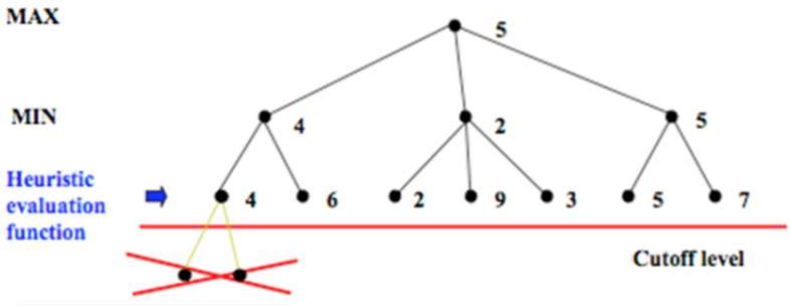
151 Goal Not Found

pra-sâmi

152

Early cutoff the search tree

- Using Minimax value estimates
  - Cutoff the search tree before the terminal state is reached
  - Use imperfect estimate of the minimax value at the leaves
    - Evaluation function



pra-sâmi

153

Practical Implementation

- How do we make these ideas practical in real game trees?

- Standard approach:
  - Cutoff test: (where do we stop descending the tree)
    - Depth limit
    - Better: iterative deepening
    - Cutoff only when no big changes are expected to occur next (quiescence search).
  - Evaluation function
    - When the search is cut off, we evaluate the current state by estimating its utility using an evaluation function.

pra-sâmi

154

Static (Heuristic) Evaluation Functions

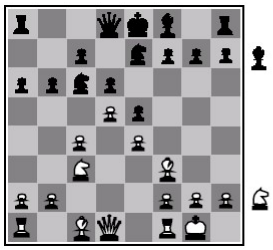
- An Evaluation Function:
  - Estimates how good the current board configuration is for a player.
  - Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponents score from the players
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces
- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board evaluation is X for a player, it's -X for the opponent.
- Many clever ideas about how to use the evaluation function.
  - e.g. null move heuristic: let opponent move twice.
- Example:
  - Evaluating chess boards,
  - Checkers
  - Tic-tac-toe

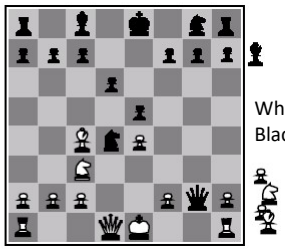
pra-sâmi

155

Evaluation Function

Black to move
White in better position





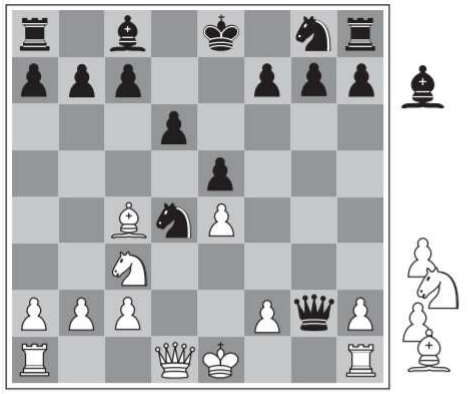
White to move
Black Winning

- For Chess, use linear weighted sum of features
  - E(s) = w_1 x $f_1(s)$ + w_2 x $f_2(s)$ + … + w_n x $f_n(s)$
  - Where w_n is weight assigned to any coin (say w_n = 9 for queen and w_n = 1 for Pawn)
  - $f_n(s)$ = number of white pieces – number of black pieces
    - For 2 white knight and 1 black knight; $f_{knight}(s)$ = 1

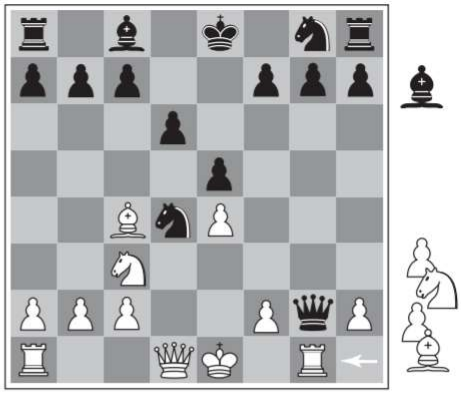
pra-sâmi

156

Evaluation Function



(a) White to move



(b) White to move

Two chess positions that differ only in the position of the rook at lower right., In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game, In (b), White will capture the queen, giving it an advantage that should be strong enough to win. Adding up the values of features apparently reasonable, but involves a strong assumption; The contribution of each feature is independent of the values of the other features. For example, assigning the value 3 to a bishop ignores the fact that bishops are more powerful in the endgame, when they have a lot of space to maneuver

pra-sâmi

157

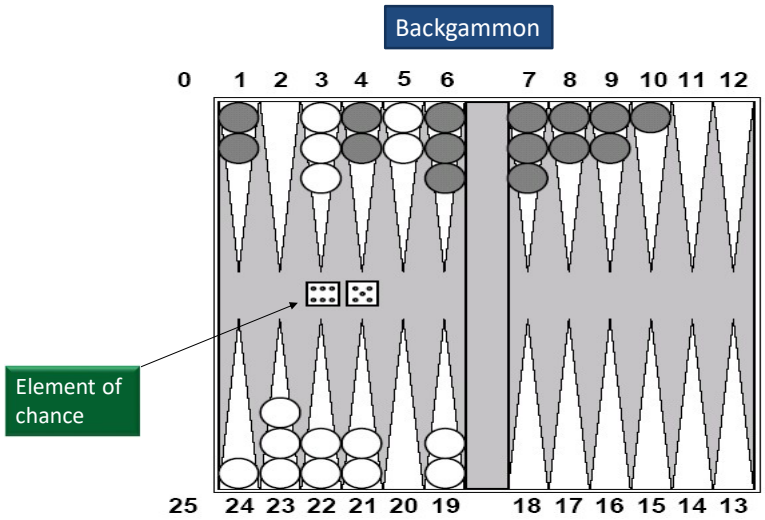
Iterative (Progressive) Deepening

- In real games, there is usually a time limit T on making a move
- How do we take this into account?
  - Using alpha-beta we cannot use "partial" results with any confidence unless the full breadth of the tree has been searched
  - So, we could be conservative and set a conservative depth-limit which guarantees that we will find a move in time < T
    - Disadvantage is that we may finish early, could have done more search
- In practice, iterative deepening search (IDS) is used
  - IDS runs depth-first search with an increasing depth-limit
  - When the clock runs out we use the solution found at the previous depth limit

pra-sâmi

158

Chance Games.



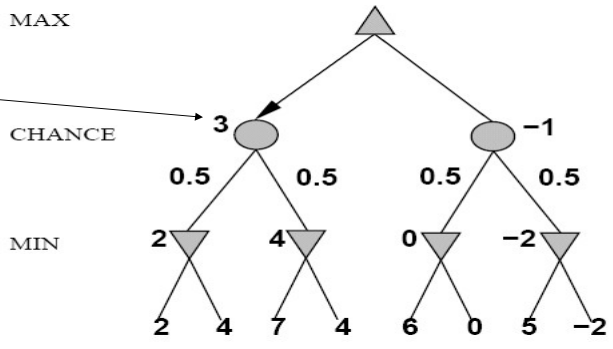
pra-sâmi

159

Expected Minimax

$$v = \sum_{chance\ nodes} P(n) \times Minimax(n)$$

$$3 = 0.5 \times 4 + 0.5 \times 2$$



Interleave chance nodes with min/max nodes

Again, the tree is constructed bottom-up

pra-sâmi

160

Game Tree for Backgammon

MAX

DICE

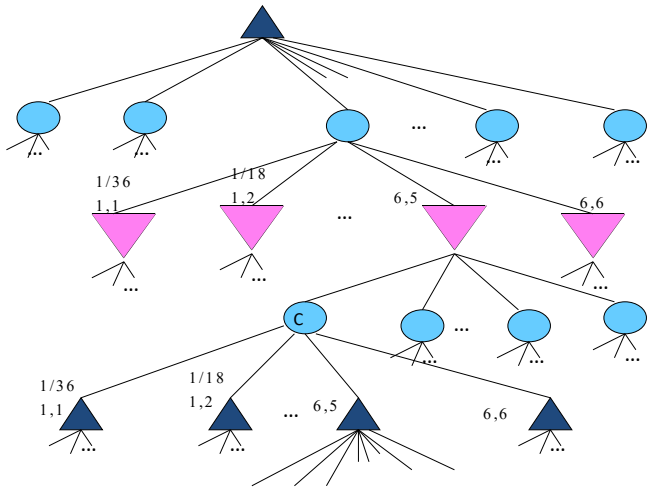
MIN

DICE

...

MAX

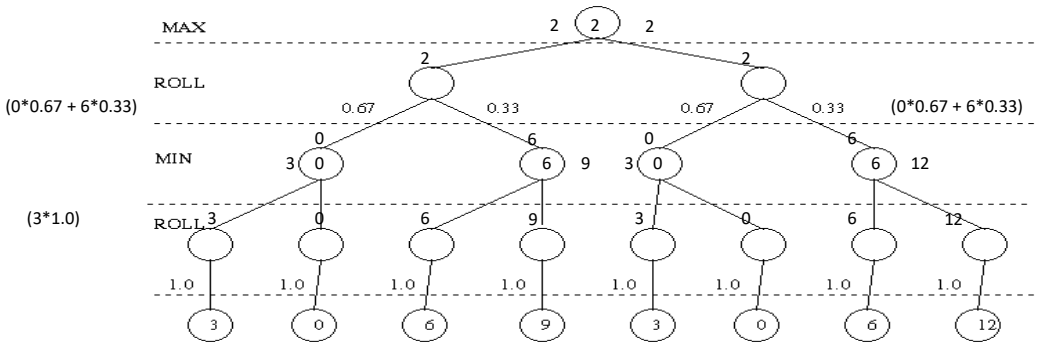
TERMINAL



pra-sâmi

161

Expectiminimax Example

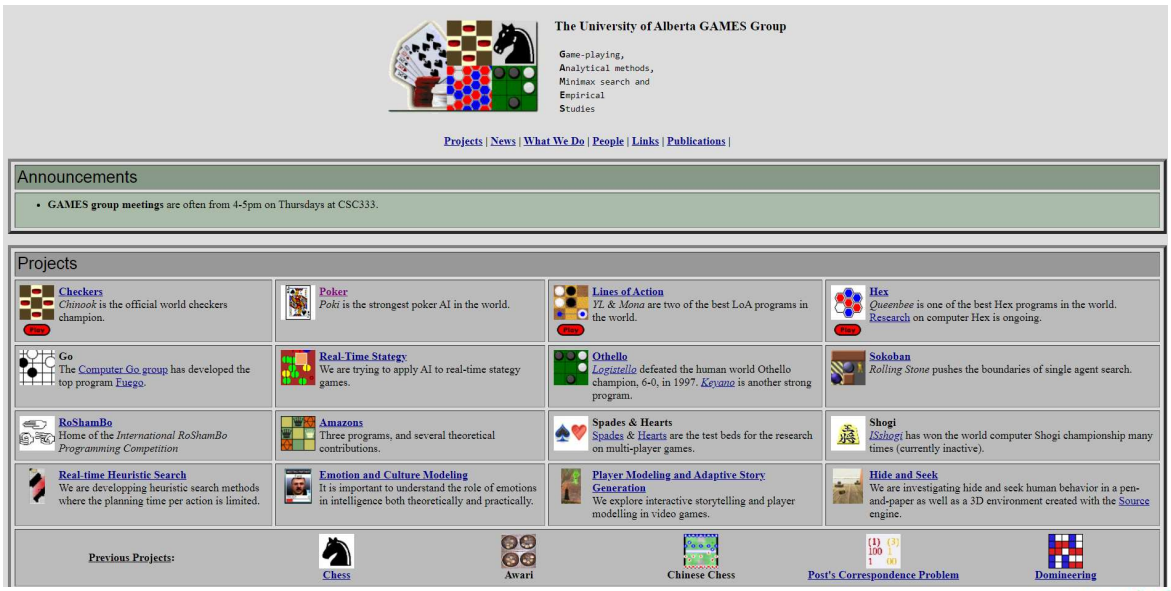


pra-sâmi

162 The State of Play

- Checkers:
  - Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- Chess:
  - Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997.
- Othello:
  - Human champions refuse to compete against computers: they are too good.
- Go:
  - Human champions refuse to compete against computers: they are too bad b > 300 (!)

163 The University of Alberta; CA



164

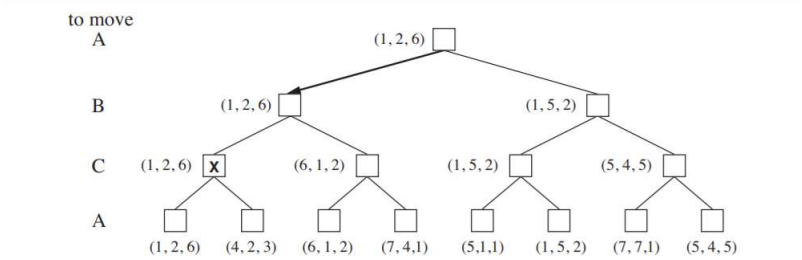
Deep Blue

- 1957: Herbert Simon
  - "within 10 years a computer will beat the world chess champion"
- 1997: Deep Blue beats Kasparov
- Parallel machine with 30 processors for "software" and 480 VLSI processors for "hardware search"
- Searched 126 million nodes per second on average
  - Generated up to 30 billion positions per move
  - Reached depth 14 routinely
- Uses iterative-deepening alpha-beta search with transpositioning
  - Can explore beyond depth-limit for interesting moves

pra-sâmi

165

Optimal Decisions in Multiplayer Games

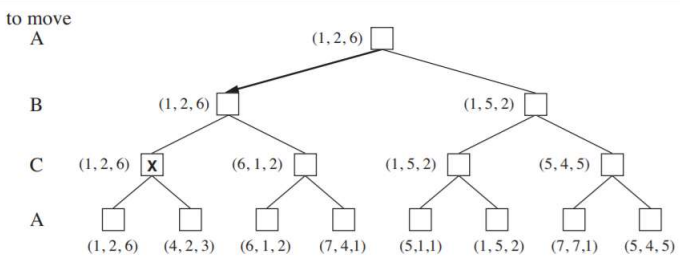


- A vector of values instead of a single value for each node
- Multiplayer games usually involve alliances, among the players
  - May be formal or informal
- Alliances are made and broken as the game proceeds
- Suppose A and B are in weak positions and C is in a stronger position
- Then it is often optimal for both A and B to attack C rather than each other
  - Collaboration emerges from purely selfish behavior

pra-sâmi

166

Optimal Decisions in Multiplayer Games



- Of course, as soon as C weakens under the joint onslaught, the alliance loses its value
- Either A or B could violate the agreement
- Explicit alliances, if any, merely make concrete what would have happened anyway
- In some cases, a social stigma attaches to breaking an alliance
  - Immediate advantage of breaking an alliance vs the long-term disadvantage of being perceived as untrustworthy

pra-sâmi

167

Nonzero Sum Game Trees

- The idea of “look ahead, reason backward” works for any game tree with perfect information.
  - I.e., also in cooperative games
- In AI, this is called retrograde analysis.
- In game theory, it is called backward induction or subgame perfect equilibrium.
- Can be extended to many games with imperfect information (sequential equilibrium).

pra-sâmi

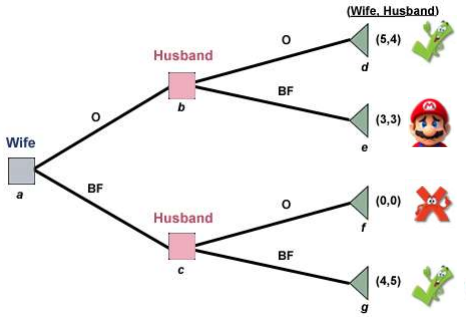
168

A Bullfight or the Opera

- Background
  - A husband and wife are deciding whether to spend the evening at a Bullfight or the Opera.
  - The husband prefers the bullfight to the opera, but the wife prefers the opera to the bullfight.
  - Both would prefer to be together at either event rather than spending the evening apart.
  - The worst outcome would be for the husband to spend the evening alone at the opera and the wife to spend the evening alone at the bullfight

- This game draws attention to the problem of co-ordination
  - Players have divergent preferences
  - A common interest in coordinating their strategies.

		Husband	
		O	BF
Wife	O	5, 4	3, 3
	BF	0, 0	4, 5



169

Compare to "Prisoner's Dilemma"

- "Deterministic, NON-turn-taking, NON-zero-sum game of Imperfect information"

A prisoner's dilemma is a situation where individual decision-makers always have an incentive to choose in a way that creates a less than optimal outcome for the individuals as a group.

There are methods of overcoming prisoner's dilemmas to choose better collective results despite apparently unfavorable individual incentives.

- Dave and Henry - Two members of a gang of bank robbers; arrested ; being interrogated in separate rooms; No other witnesses; need confession
- Be loyal and remain silent; the authorities will only be able to convict them on a lesser charge
  - One year in jail for each (1 year for Dave + 1 year for Henry = 2 years total jail time)
- If one testifies and the other does not
  - The one who testifies will go free and the other will get five years (0 years for the one who defects + 5 for the one convicted = 5 years total)
- If both testify against the other
  - Each will get two years in jail for being partly responsible for the robbery (2 years for Dave + 2 years for Henry = 4 years total jail time)

pra-sâmi

170

Win as much as you can!

- Four Teams, choose either X or Y
- Pay off Schedule
  - 4 X's:
    - Lose $ 1.00 each
  - 3 X's, 1 Y :
    - Xs: Win $1.00 each; Ys : Lose $3.00
  - 2 X's, 2 Y's :
    - Xs: Win $ 2.00 each; Ys : Lose $ 2.00 each
  - 1 X , 3 Y's:
    - Xs : Win $ 3.00; Ys: Lose $ 1.00 each
  - 4 Y's:
    - Win $ 1.00 each

	Round	Your Choice	Group's pattern of Choice	Payoff	Balance
	1	X Y	_X _Y		
	2	X Y	_X _Y		
	3	X Y	_X _Y		
	4	X Y	_X _Y		
Bonus Round (Payoff X 3)	5	X Y	_X _Y		
	6	X Y	_X _Y		
Leader's Conference	7	X Y	_X _Y		
Bonus Round (Payoff X 5)	8	X Y	_X _Y		
	9	X Y	_X _Y		
Bonus Round (Payoff X 10)	10	X Y	_X _Y		

pra-sâmi

171

Summary

- Game playing can be effectively modeled as a search problem
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for the Max player.
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.

pra-sâmi

172

Summary

- Game systems rely heavily on
  - Search techniques
  - Heuristic functions
  - Bounding and pruning techniques
  - Knowledge database on game
- For AI, the abstract nature of games makes them ct for study:
  - State of the game is easy to represent;
  - Agents are usually restricted to a small hose outcomes are defined by precise rules
- Game playing was one of the dertaken in AI as soon as computers became programmable (e.g., Turing, Sh , Wiener tackled chess).
- Game playing research has spawned a number of interesting research ideas on search, data structures, databases, heuristics, evaluations functions and many areas of computer science.

Games are fun:
Teach your computer how to play a game!