Lecture 5: Search - 4

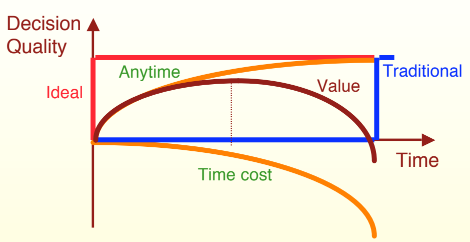
Jiaying Shen

CMPSCI 683
Fall 2004

Today's lecture

- Anytime A*
- Hierarchical A*
- Other Examples of Hierarchical Problem Solving
- Reviews of A* and its varations

Anytime algorithms



- **Ideal** (maximal quality in no time)
- **Traditional** (quality maximizing)
- **Anytime** (utility maximizing)

Anytime A*

- A* is best first search with $f(n) = g(n) + h(n)$
- Three changes make it an anytime algorithm:

  (1) Use a non-admissible evaluation function $f'(n)$ to select node to expand next so that sub-optimal solutions are found quickly.

  (2) Continue the search after the first solution is found, using an auxiliary, admissible evaluation function $f(n)$ to prune the open list.

  (3) When the open list is empty, the last solution generated is optimal.

- How to choose a non-admissible evaluation function $f'(n)$?

Weighted evaluation functions

- **Use** $f'(n) = (1 - w)*g(n) + w*h(n)$
- **Higher weight on** $h(n)$ **tends to search deeper.**
- **Admissible if** $h(n)$ **is admissible and** $w \leq 0.5$
- **Otherwise, the search may not be optimal, but it normally finds solutions much faster.**
- **An appropriate w makes possible a tradeoff between the solution quality and the computation time**

Adjusting W Dynamically

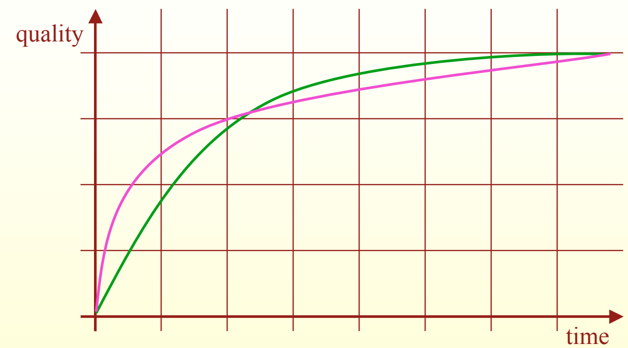
Suppose you had the following situations, how would you adjust w.

- the open list has gotten so large that you are running out of memory?
- you are running out of time and you have not yet reached an answer?
- there are a number of nodes on the open list whose h value is very small?

Pruning States in Anytime A*

- **For each node, store real f(n) = g(n)+h(n)**
  - f(n) is the lower bound on the cost of the best solution path through n
- **When find solution node n1**
  - f(n1) is an upper bound of the cost of the optimal solution
  - Prune all nodes n on the open list that have **real** f(n) >= f(n1)

Performance profile of w=.6



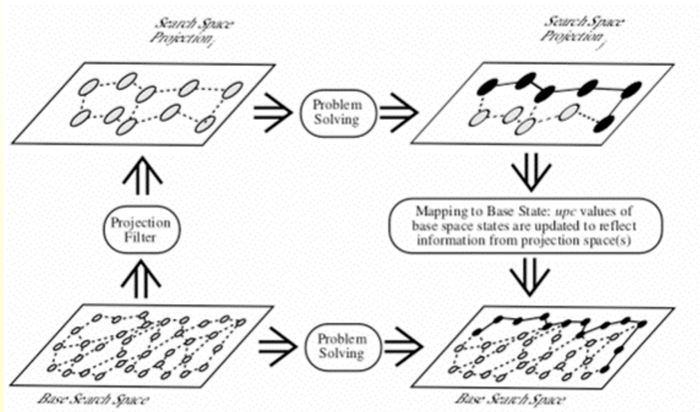
What would w=.75 look like?

Heavier weights tend to create more of an anytime effect

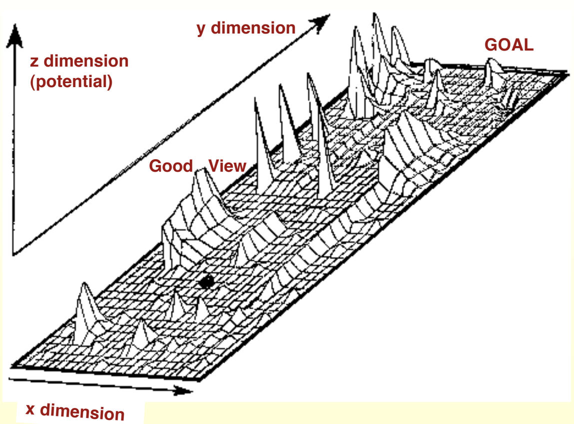
Hierarchical Search

- Idea: Find a high-level structure for a solution, and then use to find detail solution
- Benefit: Potentially Reduce "significantly" the size of the detail search space that needs to be searched

Hierarchical Search Perspective



Climbing the Hill for a Better View



Types of abstractions

- Ignoring features of the world
- Ignoring constraints
- Limiting the horizon
- Limiting the number of goals

Naive Hierarchical A*

- **Operates like A* except that h(s) is computed by searching at the next higher level of abstraction.**
  - $h(s) = d(\Phi(s), \Phi(goal))$
- **The result is combined with other estimates (e.g. cheapest operator cost) to produce the final h(s).**
  - h(s) >= to cheapest operator cost
- **Heuristic values are being cached to improve performance.**

State-space Abstraction for A*

- **A mapping Φ of states from state space <S,d> into <S',d'> is an abstraction transformation iff:**

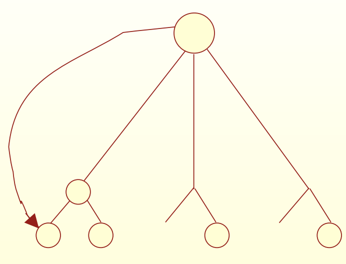
$$d'(\Phi(s1), \Phi(s2)) \leq d(s1, s2)$$

- **Abstraction can be used in order to automatically create admissible heuristic functions**
  - $h(s1) = d'(\Phi(s1), \Phi(g))$
  - Searching in Abstraction Space to Compute h using blind search

Types of Abstractions

- **Embedding =** adding **edges to the original graph (corresponds to macro or relaxed operators).**
- **Homomorphism = grouping together sets of states to create a single abstract state (corresponds to dropping a feature/variable from the state representation).**

Embedding



Eliminate conditions
Make possible new operator

Not always a useful idea...

The primary risk in using a heuristic created by abstraction is that the total cost of computing h(-) over the course of the search can exceed the savings.

Generalized Valtorta's Theorem

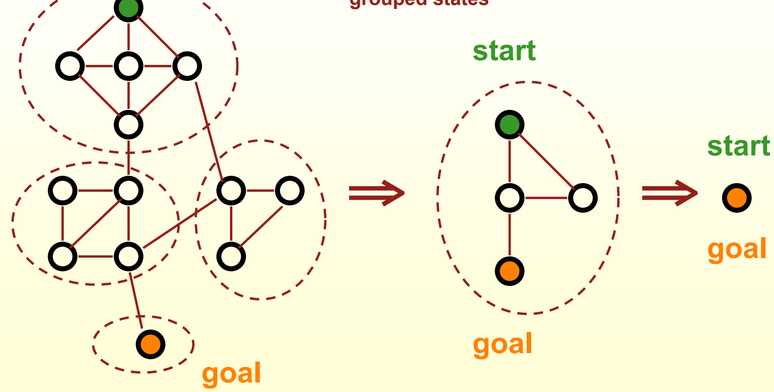
- **If state s must be expanded by blind search, then either s or $\Phi(s)$ must be expanded by A* using $h_\Phi(-)$.**
  - A state is necessarily expanded by blind search if its distance from the start state is strictly less than the distance from the start state to the goal state
- **As a result**
  - no speed-up when Φ is an embedding since $\Phi(s)$ not equal to $\Phi(s')$
  - possible speed-up when Φ is a homomorphism
- **Q. What speed-up can be achieved?**

"Max-degree" Star Abstraction

- **The state with the highest degree is grouped together with its neighbors within a certain distance (the abstraction radius) to form a single abstract state.**

Star abstraction with radius = 2

State with the largest degree within a certain distance is grouped together with neighbors, repeat for non-grouped states



Naive Hierarchical A*

TABLE 1. Naive Hierarchical A*. (abstraction radius = 2)

Search Space	Size (# states) All Levels	Size (# states) Base Level	Nodes Expanded: Blind Search	Nodes Expanded: Hierarchical A* All Levels	Nodes Expanded: Hierarchical A* Base Level
Blocks-5	1166	866	389	2766	118
5-puzzle	961	720	348	3119	224
Fool's Disk	4709	4096	1635	12680	629
Hanoi-7	2894	2187	1069	18829	701
KL2000	3107	2736	1236	7059	641
MC 60-40-7	2023	1878	934	2412	702
Permute-6	731	720	286	806	77
Words	5330	4493	1923	19386	604

- A single base level search can spawn a large number of searches at the abstract level

Reducing Search in Abstract Spaces

- Observation: all searches related to the same base level problem have the same goal.
- This allows additional types of caching of values.
- It leads to breaking Valtorta's barrier in 5 out of 8 search spaces.

Exploit Information for Blind Search in Abstract Space

- Naïve Hierarchical A*
  - Cache h in abstract space
- V1 - h*caching
  - Cache exact h's (h*) along optimal solution in abstract space
    - Exploit in further searches in abstract space and cache for use in base level search
  - Does not preserve monotone properties (h* not comparable with h), but don't need to reopen nodes
- V2
  - Cache optimal path in abstract space (optimal-path caching)
- V3
  - Remember optimal path length in abstract search space (P-g caching)
    - P being optimal path length from start to goal in abstract space

Hierarchical A*

TABLE 2. Hierarchical A*. (abstraction radius = 2)

Search Space	Nodes Expanded: Blind Search	Hierarchical A* Naive	V1	V2	V3	# problems V3 < BS (out of 200)
Blocks-5	389	2766	1235	478	402	96
5-puzzle	348	3119	1616	854	560	14
Fool's Disk	1635	12680	8612	3950	**1525**	132
Hanoi-7	1069	18829	10667	5357	3174	0
KL2000	1236	7059	3490	1596	**1028**	171
MC 60-40-7	934	2412	1531	1154	**863**	128
Permute-6	286	806	482	**279**	**242**	113
Words	1923	19386	7591	2849	**1410**	124

The Granularity of Abstraction

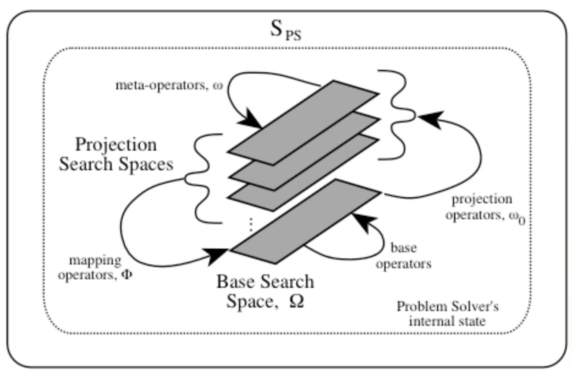
- **Increasing the radius of abstraction has two contradictory effects:**
  - **+ abstract spaces contain fewer states and each abstract search produces values for more states, but**
  - **- the heuristic is less discriminating**
- **The best case breaks the Valtorta's barrier in every search space.**

A* with best abstraction radius

TABLE 3. Hierarchical A*. (best abstraction radius)

Search Space	Radius	Nodes Expanded: Blind Search	Nodes Expanded: Hierarchical A* Naive	Nodes Expanded: Hierarchical A* V3	# problems V3 < BS (out of 200)	CPU seconds: Blind Search	CPU seconds: V3
Blocks-5	5	389	611	**309**	123	69	86
5-puzzle	12	348	354	**340**	131	36	40
Fool's Disk	4	1635	**1318**	**1172**	194	872	902
Hanoi-7	20	1069	1097	**1055**	117	102	108
KL2000	5	1236	1306	**1072**	178	398	384
MC 60-40-7	4	934	**822**	**803**	144	266	253
Permute-6	5	286	**201**	**194**	192	82	67
Words	3	1923	9184	**1356**	128	1169	1273

Hierarchical Problem Solving



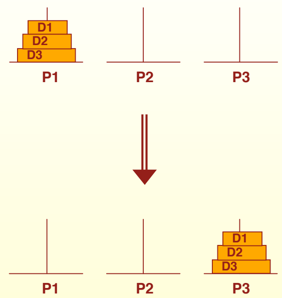
Traditional vs. Hierarchical Problem Solver

- **Traditional problem solver:**
  - Problem space
    - Set of operators
    - Set of states
  - Problem
    - Initial state
    - Goal state
- **Hierarchical problem solver:**
  - Generate abstraction space(s)
    - Set of operators and states
  - Produce solution in highest abstraction space
    - Map from operators and states in ground space
  - Refine down to ground level
    - Map to operators and states in ground space

3-Disk Towers of Hanoi Example Operator

(MOVE_DISK3_FROM_PEG2_TO_PEG3)
(preconds ((on disk3 peg2)
(not (on disk2 peg2))
(not (on disk1 peg2))
(not (on disk2 peg3))
(not (on disk1 peg3))))
(effects ((not (on disk3 peg2))
(on disk3 peg3))))

(MOVE_DISK2_FROM_PEG2_TO_PEG3)
(preconds ((on disk1 peg2)
(not (on disk1 peg2))
(not (on disk1 peg3))
(effects ((not (on disk2 peg2))
(on disk2 peg3))))



A smaller disk can be always moved without interfering with a large disk!!

Producing Abstraction Spaces

- **Idea:** Abstraction spaces are formed by removing sets of literals from the operators and states of the domain
- **Premise:** Literals in a domain only interact with some of the other literals
  - literals in D3 moves do not interact with literals in D2 moves
- **Method:** Partition literals into classes based on their interactions, and order the classes
- **This forms a monotonic hierarchy of abstraction spaces, that is, any plan to achieve a literal cannot add or delete a literal higher in the hierarchy.**

Producing Abstraction Spaces: Algorithm

Input: The set of operators for a domain.
Output: A hierarchy of monotonic abstraction spaces.

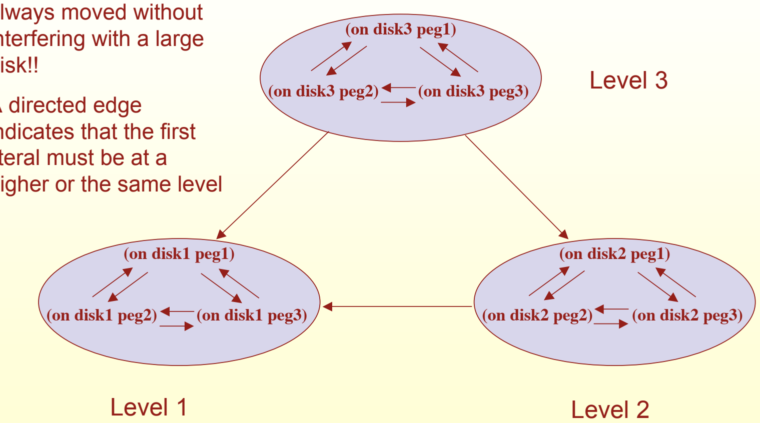
```
Create_Abstraction_Hierarchy(OPERATORS)
1.   ForEach OP in OPERATORS
        ForEach LIT1 in Effects(OP)
        i.   ForEach LIT2 in Effects(OP)
                Add_Directed_Edge(LIT1,LIT2,GRAPH)
        ii.  ForEach LIT2 in Preconditions(OP)
                Add_Directed_Edge(LIT1,LIT2,GRAPH)
2.   Combine_Strongly_Connected_Components(GRAPH)
3.   Topological_Sort(GRAPH)
```

- Complexity: $O(o\ n^2)$
  - o is number of operators
  - n is number of different instantiated literals

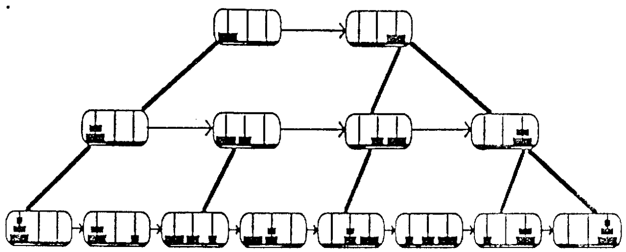
3-Disk Towers of Hanoi Constraints on the Literals

A smaller disk can be always moved without interfering with a large disk!!

A directed edge indicates that the first literal must be at a higher or the same level



3-Disk Towers of Hanoi Abstraction Hierarchy



Analysis

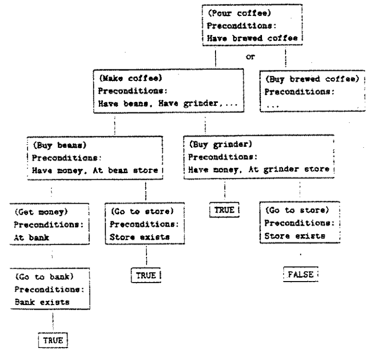
- Solution to an n-disk problem will require 2^n steps
- Backtracking across abstraction levels or subproblems within an abstraction level is never required
- Search space reduction is from exponential, $O(b^L)$, to linear, $O(L)$, in length of the solution, where b is the branching factor.
  - Never factored in construction of abstraction space; assumption used over and over for many problems

ABSTRIPS - Sacerdoti

- Plans using a *hierarchy of abstraction spaces.*
- Tries to avoid backtracking by working on "more important" goals first.
- *Criticality* assigned to preconditions by user and adjusted by system based on ability of operators to achieve them.
- At each level, planner would assume less critical preconditions to be true.

STRIPS Example

Operator	*Precondition*	*Effect*	*Starting state*	*Goal state*
Pour coffee	Have brewed coffee	Problem solved	Not have brewed coffee In kitchen Have grinder Have money Have boiling water	Have brewed coffee In kitchen Have grinder Have money Have boiling water
Make coffee	Have beans Have grinder Have boiling water Be in the kitchen	Have brewed coffee		
Buy something	Be at store Have money	Have something		
Go someplace	Place exists	Be at place Not at any other place		
Get money	Be at bank	Have money		
Boil water	Be in the kitchen	Have boiling water		



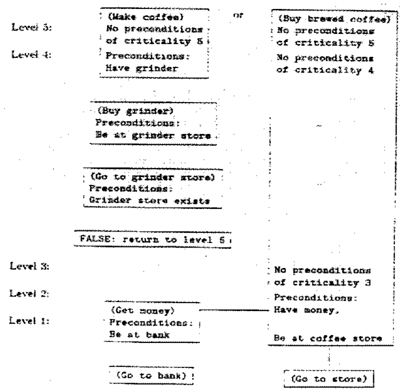
Planning I 10

ABSTRIPS cont'd

Assign precondition criticalities:

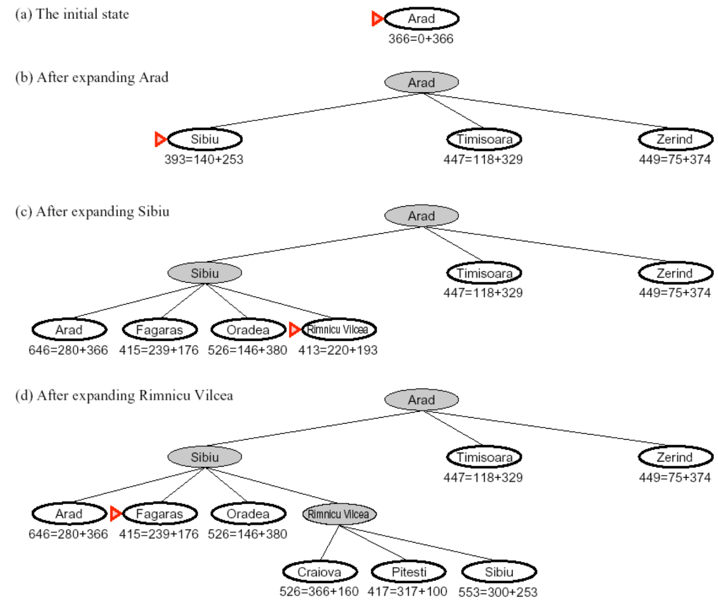
Precondition	*Criticality*
Bean store exists	5
Brewed-coffee store exists	5
Bank exists, Gr.	5
Have grinder	4
Have beans, boiling water, money	2
Be at brewed-coffee store, bean store, bank	1

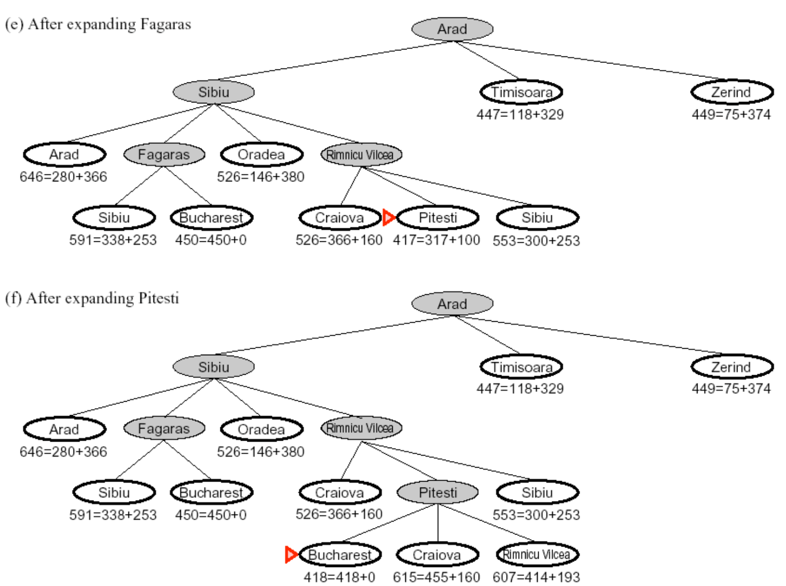
The coffee example revisited:



Planning I

A*

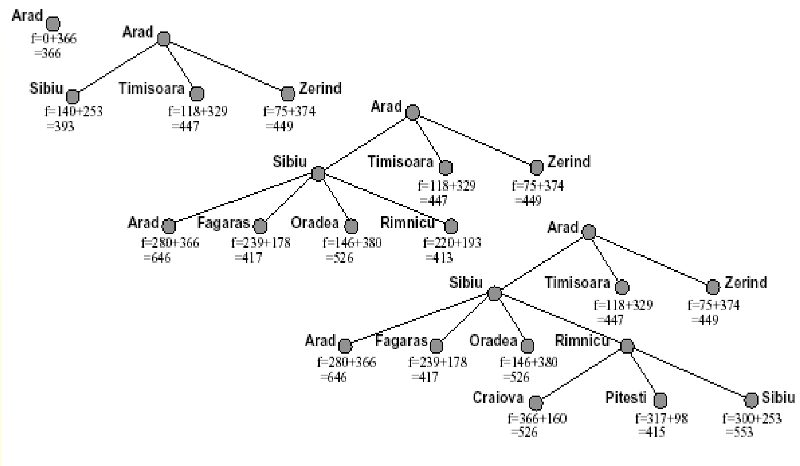




Questions

- **What are open list and closed list? Why do we need them?**
- **Why is A* optimal?**
- **Why does A* suffer from high memory requirement?**

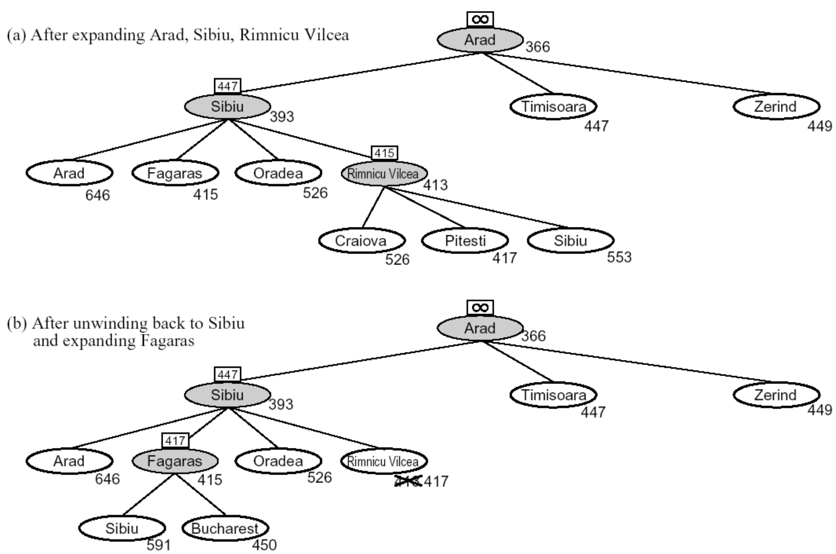
IDA*

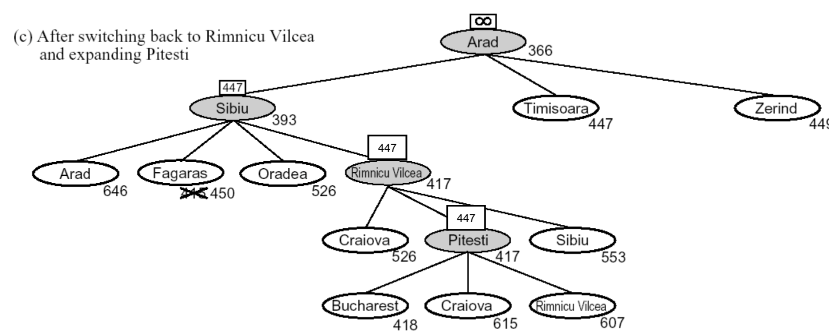


Questions

- How is IDA* different from standard iterative deepening?
- What is the f-bound of each iteration?
- Why does IDA* use less memory than A*?
- What problems does IDA* suffer from?

RBFS

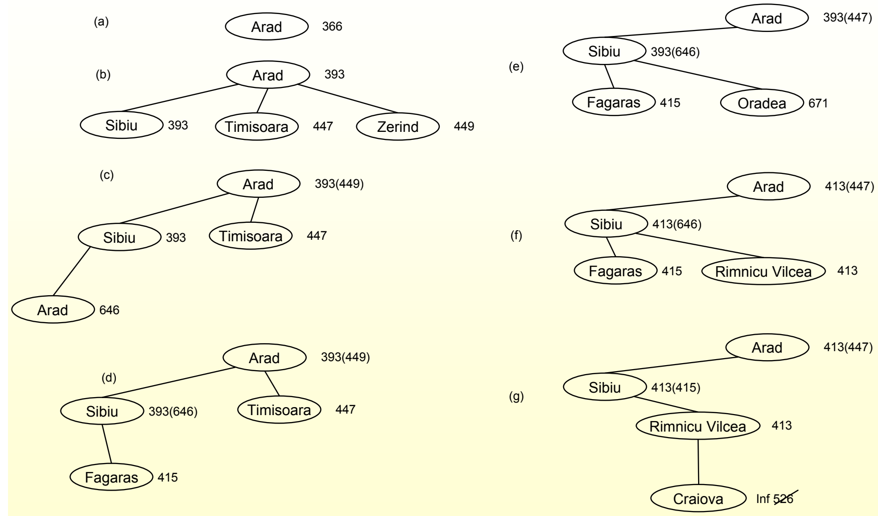


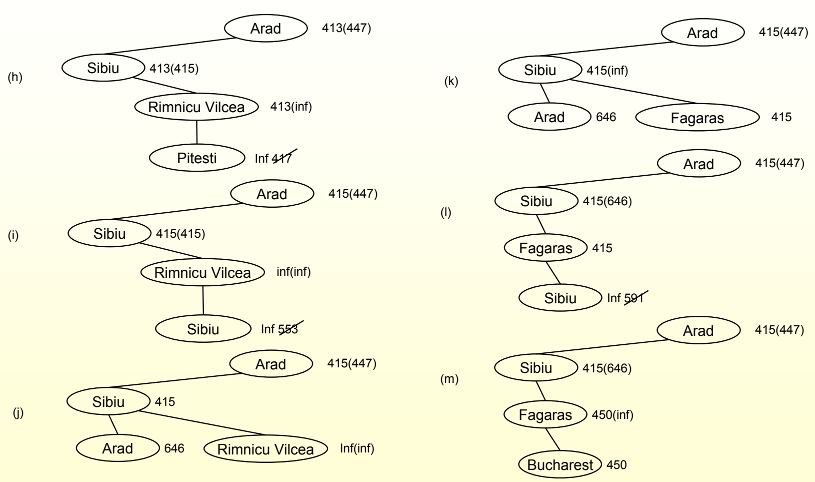


Questions

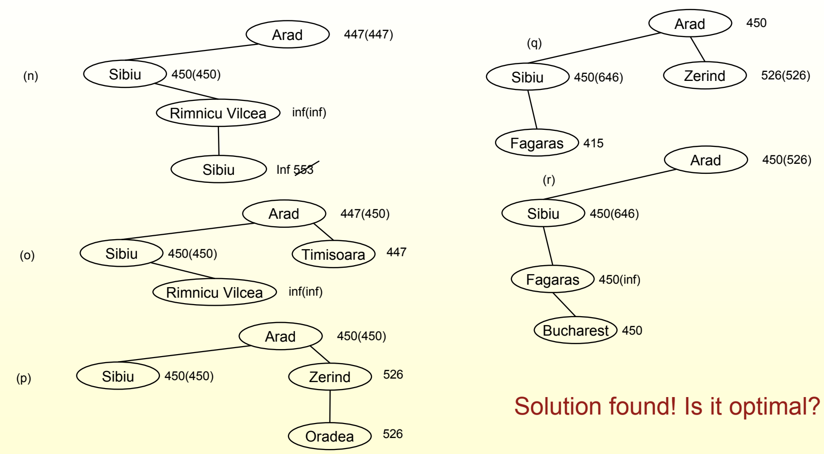
- Why do we need to memorize the best alternative path?
- Why do we need to memorize the best descendent of a forgotten node?
- Why does RBFS need less space than A*?
- Why is RBFS optimal?
- What problem does RBFS suffer from?

SMA* with memory of 4 nodes





Do we end here?



Solution found! Is it optimal?

Questions

- **Why do we need to back up the best f-value of all the successors of a node?**
- **Why do we need to back up the f-value of a node's best forgotten child?**
- **Is SMA* optimal? Why?**
- **Why is SMA* guaranteed not to get stuck in a loop?**

Next lecture

- **Iterative Improvement**
  - Simulated Annealing (Hill Climbing)
  - Solution Repair/Debugging
  - GSAT
- **Heuristics for CSP**
  - texture measures