Lecture 3: Search - 2

Victor R. Lesser

CMPSCI 683

Fall 2004

Today's lecture

- **Search and Agents**
  - Material at the end of last lecture
- **Continuation of Simple Search**
  - The use of background knowledge to accelerate search
  - Understand how to devise heuristics
  - Understand the A* and IDA* algorithms
  - Reading: Sections 4.1-4.2.
- **Characteristics of More Complex Search**
  - Subproblem interaction
  - More complex view of operator/control costs
  - Uncertainty in search
  - Non-monotonic domains
  - Search redundancy

Problem Solving by Search

There are four phases to problem solving :

1. Goal formulation
- based on current world state, determine an appropriate **goal**;
- describes desirable states of the world;
- goal formulation may involve general goals or specific goals;

2. Problem formulation
- formalize the problem in terms of states and actions;
- **state space representation**;

3. Problem solution via search
- find sequence(s) of actions that lead to goal state(s);
- possibly select "best" of the sequences;

4. Execution phase
- carry out actions in selected sequence.

Agent vs. Conventional AI View

- **A completely autonomous agent would have to carry out all four phases.**
- **Often, goal and problem formulation are carried out prior to agent design, and the "agent" is given specific goal *instances* (agents perform only search and execution).**
  - general goal formulation, problem formulation, specific goal formulation, etc.
- **For "non-agent" problem solving:**
  - a solution may be simply a specific goal that is achievable (reachable);
  - there may be no execution phase.
- **The execution phase for a real-world agent can be complex since the agent must deal with uncertainty and errors.**

Goals vs.Performance Measures (PM)

- **Adopting goals and using them to direct problem solving can simplify agent design.**
- **Intelligent/rational agent means selecting best actions relative to a PM, but PMs may be complex (multiple attributes with trade-offs).**
- **Goals simplify reasoning by limiting agent objectives (but still organize/direct behavior).**
- Optimal **vs.** satisficing **behavior: best performance vs. goal achieved.**
- **May use both: goals to identify acceptable states plus PM to differentiate among goals and their possible solutions.**

Problem-Solving Performance

- **Complete search-based problem solving involves both the search process and the execution of the selected action sequence.**
  - **Total cost** of search-based problem solving is the sum of the search costs and the path costs (operator sequence cost).
- **Dealing with total cost may require:**
  - Combining “apples and oranges” (e.g., travel miles and CPU time)
  - Having to make a trade-off between search time and solution cost optimality (resource allocation).
  - These issues must be handled in the performance measure.

Knowledge and Problem Types

- **Problems can vary in a number of ways that can affect the details of how problem-solving (search) agents are built.**
- **One categorization is presented in AIMA: (related to accessibility and determinism)**
  - Single-state problems
    - Agent knows initial state and exact effect of each action;
    - Search over single states;
  - Multiple-state problems
    - Agent cannot know its exact initial state and/or the exact effect of its actions;
    - Must search over state sets;
    - May or may not be able to find a guaranteed solution;

Knowledge and Problem Types *(cont'd)*

- **Contingency problems**
  - Exact prediction is impossible, but states may be determined during execution (via sensing);
  - Must calculate tree of actions, for every contingency;
  - **Interleaving** search and execution may be better
    - Respond to state of world after execution of action with uncertain outcome (RTA*);
- **Exploration problems**
  - Agent may have no information about the effects of its actions and must experiment and learn
  - Search in real world vs. model.

Uninformed/Blind Search Strategies

- Uninformed strategies do not use any information about how *close* a node might be to a goal (additional cost to reach goal).
- They differ in the order that nodes are expanded (and operator cost assumptions).

Informed/Heuristic Search

- While uninformed search methods can in principle find solutions to any state space problem, they are typically too inefficient to do so in practice.
- Informed search methods use *problem-specific knowledge* to improve *average* search performance.

What are heuristics?

- Heuristic: problem-specific knowledge that reduces expected search effort.
- Informed search uses a heuristic evaluation function that denotes the relative desirability of expanding a node/state.
    - often include some estimate of the *cost to reach the nearest goal state* from the current state.
- In blind search techniques, such knowledge can be encoded only via state space and operator representation.

Examples of heuristics

- Travel planning
    - Euclidean distance
- 8-puzzle
    - Manhattan distance
    - Number of misplaced tiles
- Traveling salesman problem
    - Minimum spanning tree

Where do heuristics come from?

Heuristics from relaxed models

- Heuristics can be generated via simplified models of the problem
- Simplification can be modeled as deleting constraints on operators
- Key property: Heuristic can be calculated efficiently

Best-first search

- Idea: use an evaluation function for each node, which estimates its "desirability"
- Expand most desirable unexpanded node
- Implementation: open list is sorted in decreasing order of desirability

Best-First Search

1) Start with *OPEN* containing just the initial state.
2) Until a goal is found or there are no nodes left on *OPEN* do:
   (a) Pick the best node on *OPEN*.
   (b) Generate its successors.
   (c) For each successor do:
      i. If it has not been generated before, evaluate it, add it to *OPEN*, and record its parent.
      ii. If it has been generated before, **change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.**

Avoiding repeated states in search

- Do not re-generate the state you just came from
- Do not create paths with cycles
- Do not generate any state that was generated before (using a hash table to store all generated nodes)

Greedy search

- Simple form of best-first search
- Heuristic evaluation function $h(n)$ estimates the cost from n to the closest goal
- Example: straight-line distance from n to Bucharest
- Greedy search expands the node that appears to be closets to the goal
- Properties of greedy search?

Problems with best-first search

- Uses a lot of space?
- The resulting algorithm is complete (in finite trees) but not optimal?

Informed Search Strategies

- **Search strategies:**
  - Best-first search (a.k.a. ordered search):
    - greedy (a.k.a. best-first)
    - A*
    - ordered depth-first (a.k.a. hill-climbing)
  - Memory-bounded search:
    - Iterative deepening A* (IDA*)
    - Simplified memory-bounded A* (SMA*)
  - Time-bounded search:
    - Anytime A*
    - RTA* (searching and acting)
  - Iterative improvement algorithms (generate-and-test approaches):
    - Steepest ascent hill-climbing
    - Random-restart hill-climbing
    - Simulated annealing
  - Multi-Level/Multi-Dimensional Search

State Space Characteristics

There are a variety of factors that can affect the choice of search strategy and direction of search:

- Branching factor;
- Expected depth/length of solution;
- Time vs. space limitations;
- Multiple goal states (and/or initial states);
- Implicit or Explicit Goal State specification
- Uniform vs. non-uniform operator costs;
- Any solution vs. optimal solution;
- Solution path vs. state;
- Number of acceptable solution states;
- Different forward vs. backward branching factors;
- Can the same state be reached with different operator sequences

Minimizing total path cost: A*

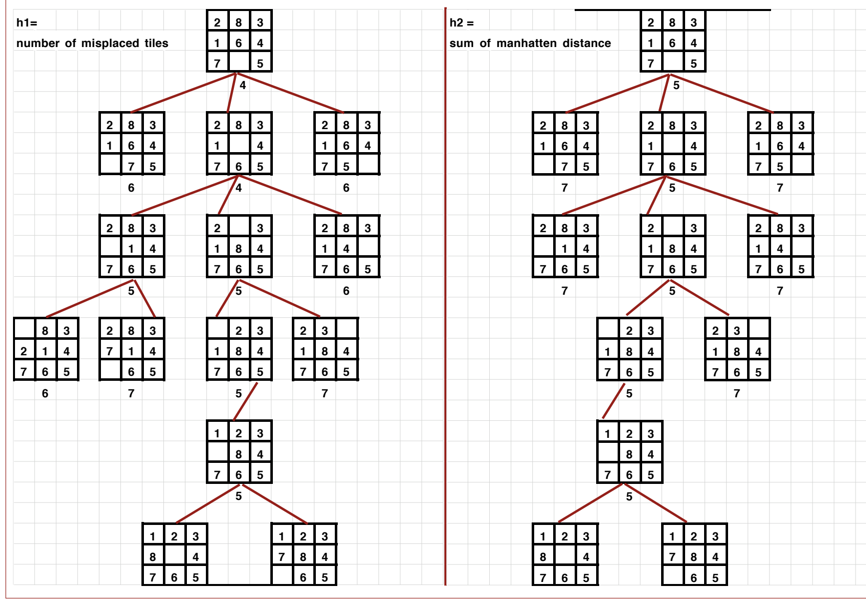
- Similar to best-first search except that the evaluation is based on total path (solution) cost:

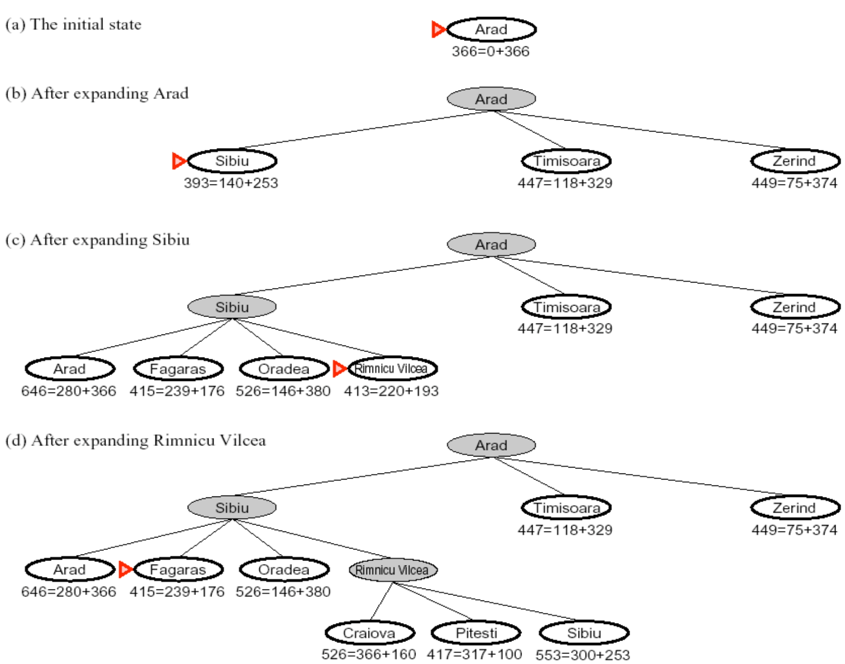
  $f(n) = g(n) + h(n)$ where:

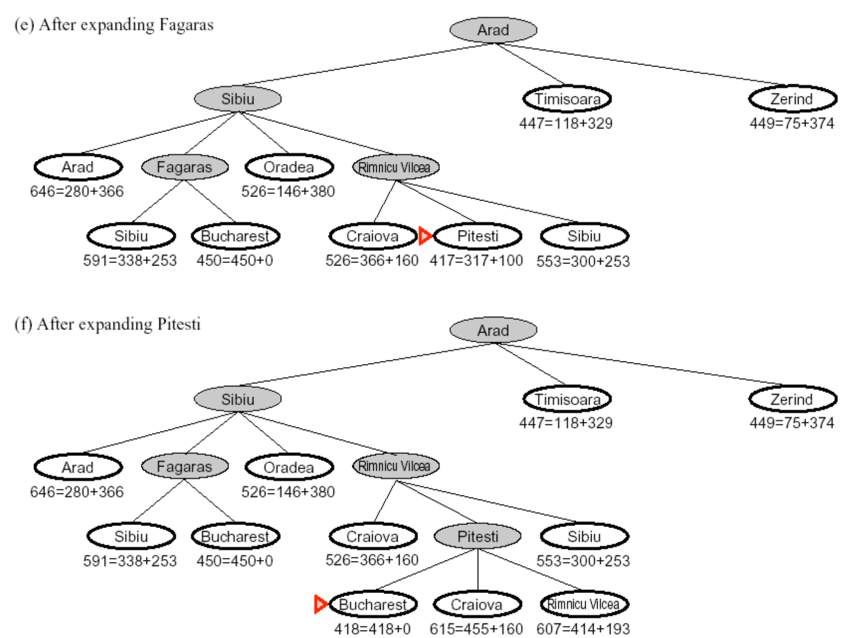
  $g(n)$ = cost of path from the initial state to n

  $h(n)$ = estimate of the remaining distance

Example: tracing A* with two different heuristics







Admissibility and Monotonicity

- **Admissible heuristic** = never overestimates the actual cost to reach a goal.
- **Monotone heuristic** = the f value never decreases along any path.
- When h is admissible, monotonicity can be maintained when combined with pathmax: $f(n') = \max(f(n), g(n')+h(n'))$

  Does monotonicity in f imply admissibility?

Optimality of A*

Intuitive explanation for monotone h:

- If h is a lower-bound, then f is a lower-bound on shortest-path through that node.
- Therefore, f never decreases.
- It is obvious that the first solution found is optimal (as long as a solution is accepted when $f(solution) \leq f(node)$ for every other $node$).

Proof of optimality of A*

Let O be an optimal solution with path cost f^*.
Let SO be a suboptimal goal state, that is $g(SO) > f^*$

Suppose that A* terminates the search with SO.

Let n be a leaf node on the optimal path to O

$f^* \geq f(n)$ admissibility of h

$f(n) \geq f(SO)$ n was not chosen for expansion

$f^* \geq f(n) \geq f(SO)$

$f(SO) = g(SO)$ SO is a goal, $h(SO) = 0$

$f^* \geq g(SO)$ contradiction!

Completeness of A*

A* is complete unless there are infinitely many nodes with $f(n) < f^*$

A* is complete when:

(1) there is a positive lower bound on the cost of operators.

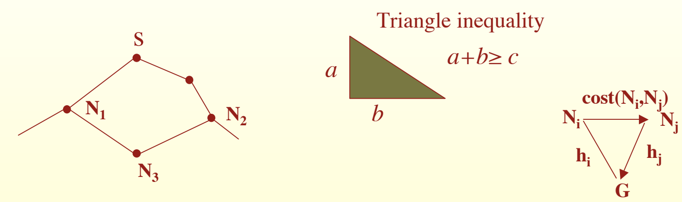
(2) the branching factor is finite.

A* is maximally efficient

- For a given heuristic function, no optimal algorithm is guaranteed to do less work in terms of nodes expanded.
- Aside from ties in f, A* expands every node necessary for the proof that we've found the shortest path, and no other nodes.

Local Monotonicity in A*

"locally" admissible if h(Ni) - h(Nj) ≤ cost(Ni, Nj) & h(goal)=0
Each state reached has the minimal g(N)

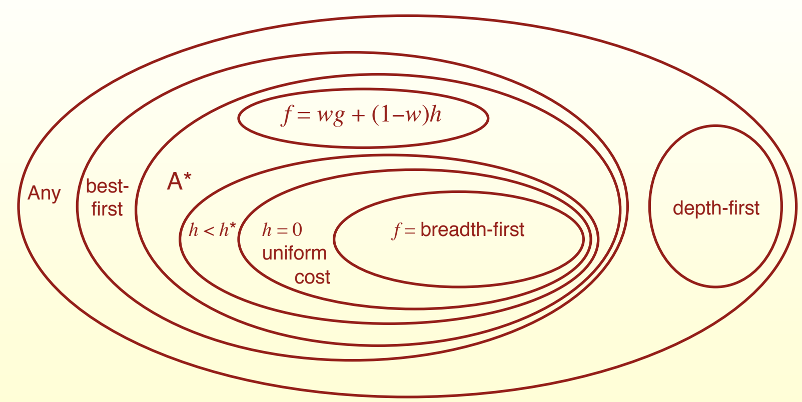


$f(N_1) < f(N_2) \Rightarrow f(N_3 \text{ via } N_1) \leq f(N_3 \text{ via } N_2)$
N_1 always expanded before N_2
Not necessary to expand $N_2 \rightarrow N_3$ if expanded $N_1 \rightarrow N_3$ also $f(N_1) \leq f(N_3)$

Questions

- What is the implications of local monotonicity
  - Amount of storage
- What happens if h1<=h2<=h for all states
  - h2 dominates h1
- What are the implications of overestimating h
  - Suppose you can bound overestimation

Relationships among search algs.



Heuristic Function Performance

- **While informed search can produce dramatic real (average-case) improvements in complexity, it typically does not eliminate the potential for exponential (worst-case) performance.**
- **The performance of heuristic functions can be compared using several metrics:**
  - *Average number of nodes expanded (N)*
  - *Penetrance (P = d/N)*
  - **Effective branching factor** ***(b*)***
    - If solution depth is d then b* is the branching factor that a uniform search tree would have to have to generate *N* nodes ($N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$;
    - EBF tends to be relatively independent of the solution depth.
- ***Note that these definitions completely ignore the*** *cost* ***of applying the heuristic function.***

Measuring the heuristic payoff

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Meta-Level Reasoning

- **Search cost involves both the cost to expand nodes and the cost to apply heuristic function.**
- **Typically, there is a** ***trade-off*** **between the cost and performance of a heuristic function.**
  - E.g., we can always get a "perfect" heuristic function by having the function do a search to find the solution and then use that solution to compute *h(node)*.

Meta-Level Reasoning (*cont'd*)

This trade-off is often referred to as the **meta-level** vs. **base-level** trade-off:

- Base-level refers to the operator level, at which the problem will actually be solved;
- Meta-level refers to the control level, at which we decide *how* to solve the problem.

We must evaluate the cost to execute the heuristic function relative to the cost of expanding nodes and the reduction in nodes expanded.

IDA* - Iterative deepening A* (Space/time trade-off)

- **A* requires open (& close) list for remembering nodes**
  – Can lead to very large storage requirements
- **Exploit the idea that:**
  $\hat{f} = \hat{g} + \hat{h} \le f^*$ (actual cost)
  – create incremental subspaces that can be searched depth-first; much less storage
- **Key issue is how much extra computation**
  – How bad an underestimate f, how many steps does it take to get $\hat{f} = f^*$
  – Worse case N computation for A*, versus N^2 for IDA*

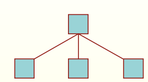
IDA* - Iterative deepening A*

- **Beginning with an f-bound equal to the f-value of the initial state, perform a depth-first search bounded by the f-bound instead of a depth bound.**
- **Unless the goal is found, increase the f-bound to the lowest f-value found in the previous search that exceeds the previous f-bound, and restart the depth first search.**

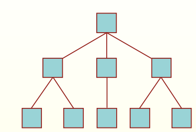
Advantages of IDA*

- **Use depth-first search with f-cost limit instead of depth limit.**
- **IDA* is complete and optimal but it uses less memory [O(bf*/c)] and more time than A*.**

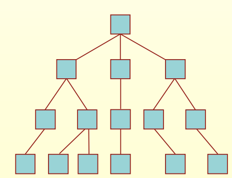
Iterative Deepening



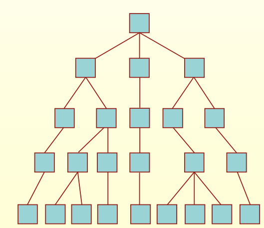
Iteration 1.



Iteration 2.



Iteration 3.



Iteration 4.

Iterative-Deepening-A*

- **Algorithm: Iterative-Deepening-A***

  1) Set THRESHOLD = the heuristic evaluation of the start state.

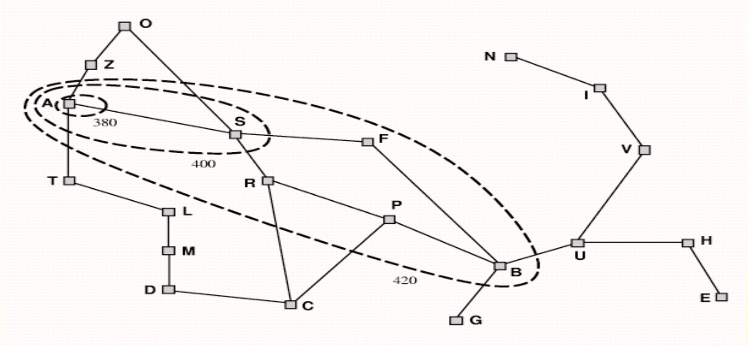
  2) Conduct a depth-first search based on minimal cost from current node, pruning any branch when its total cost function (g + h´) exceeds THRESHOLD. If a solution path is found during the search, return it.

  3) Otherwise, increment THRESHOLD by the minimum amount it was exceeded during the previous step, and then go to Step 2.

  - Start state always on path, so initial estimate is always overestimate and never decreasing.

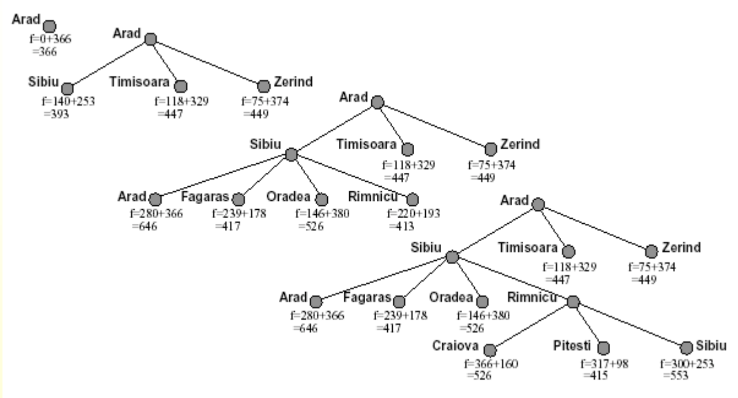
f-Cost Contours

- **Monotonic heuristics allow us to view A* in terms of exploring increasing f-cost** contours:



- **The more informed a heuristic, the more the contours will be “stretched” toward the goal (they will be more focused around the optimal path).**

Stages in an IDA* Search for Bucharest



Nodes are labeled with *f = g +h*. The *h* values are the straight-line distances to Bucharest...

What is the next Contour??

Experimental Results on IDA*

- IDA* is asymptotically same time as A* but only O(d) in space - versus $O(b^d)$ for A*
  - **Avoids overhead of sorted queue of nodes**
- IDA* is simpler to implement - no closed lists (limited open list).
- In Korf’s 15-puzzle experiments IDA*: solved all problems, ran faster even though it generated more nodes than A*.
  - **A*: solved no problems due to insufficient space; ran slower than IDA***

Next lecture

- **Continuation of Discussion of IDA***

- **Other Time and Space Variations of A***
  - RBFS
  - SMA*
  - RTA*