Lecture 5: Search 4

Victor R. Lesser

CMPSCI 683

Fall 2010

This Lecture

- Finish off Discussion of A*
- IDA*

Admissibility and Monotonicity*

- Admissible heuristic *h(n)* = never overestimates the actual cost *h*(n)* to reach a goal & ***h(n)>=0*** & ***h*(*goal*)=0**
- Monotone (Consistency) heuristic
  - For every node n and every successor n' reached from n by action a
  - ***h*(*n*) < = cost of (*n, a, n'*) + *h*(*n'*)** & ***h*(*goal*)=0**
    - The deeper you go along a path the better (or as good) the estimate of the distance to the goal state
  - **the *f* value (*which is g+h*) never decreases along any path.**
  - Implies ===> Each state reached has the minimal *g(n)*
- When h is admissible, monotonicity can be maintained when combined with pathmax: $f(n') = \max(f(n), g(n')+h(n'))$
  - Create new h -- $\boldsymbol{h'}(n') = \max(f(n) - g(n'), h(n'))$

    $f(n') = g(n') + \boldsymbol{h'}(n')$,
  - Force f to never decrease along path since $f(n') >= f(n)$

Does monotonicity in f imply admissibility?

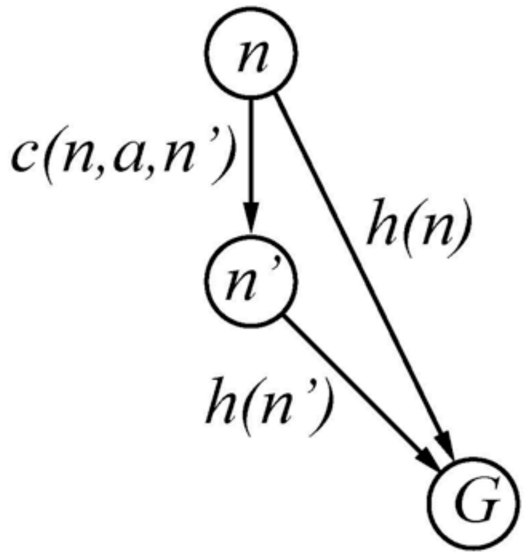
Consistent $h \Rightarrow$ Monotone f

A heuristic is *consistent* if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \;\; [= f(n)\,] \end{aligned}$$



I.e., $f(n)$ is nondecreasing along any path.

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:
  - there is a positive lower bound on the cost of operators.
  - 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.

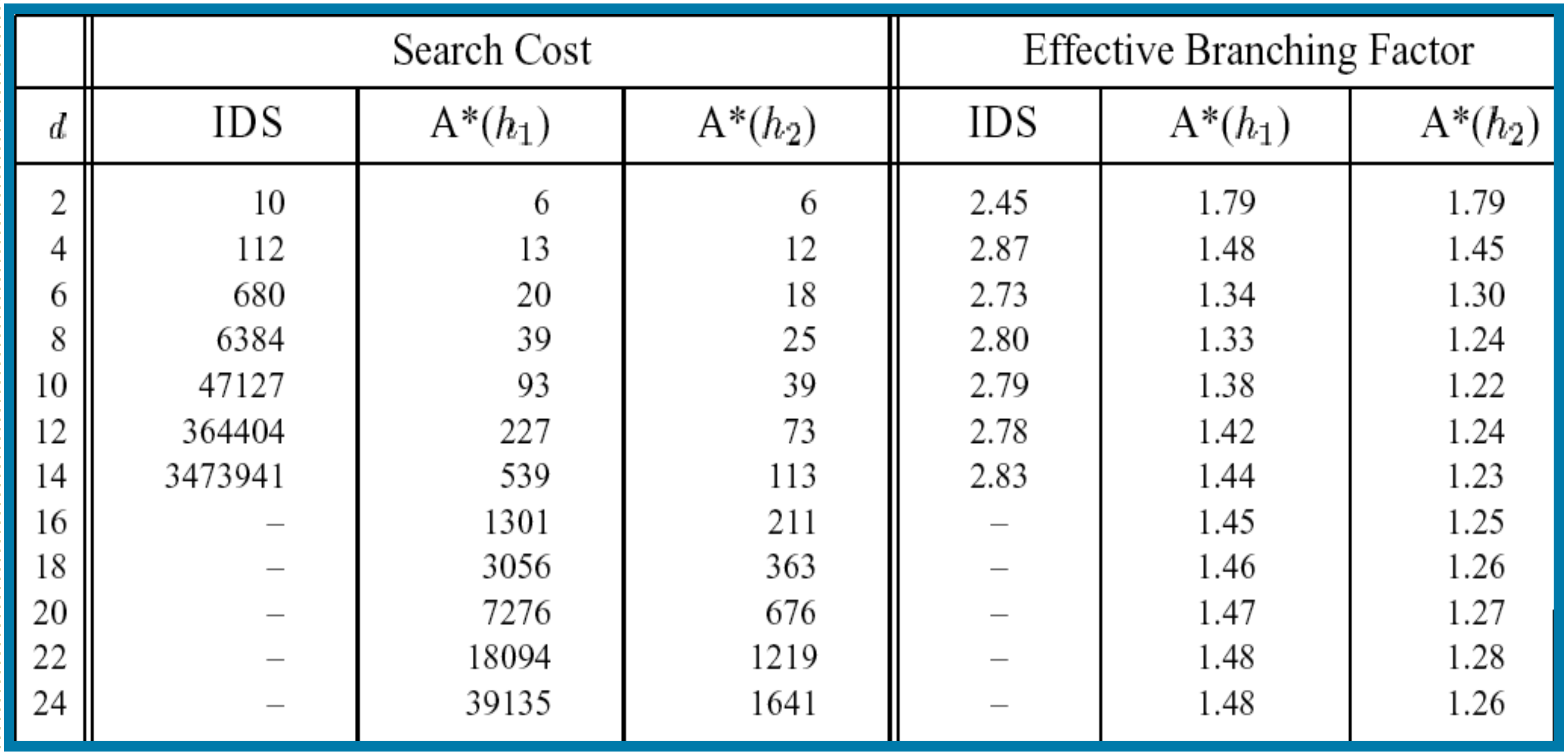
Questions*

- What is the implications of local monotonicity
  - Amount of storage
- What happens if h1<=h2<=h for all states
  - h2 dominates h1
- If h_1 and h_2 are admissible, is $\max\{h_1,h_2\}$ admissible? Is it better than h_1 and h_2?
- What are the implications of overestimating h
  - Suppose you can bound overestimation
- What if you are doing a maximizing search

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
      ($N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$; *Given a d and N what is the b*
    - 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 Iterative Deepening vs A*



	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 the use of monotone f:

  $\hat{f} = \hat{g} + \hat{h} \leq f^*$ (actual cost) and f(n) <= f(next node after n)
  - *create incremental subspaces 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.
  - ***Why if you reach a goal as a result of depth-first search is it optimal?***

Iterative-Deepening-A*

- Algorithm: Iterative-Deepening-A*

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

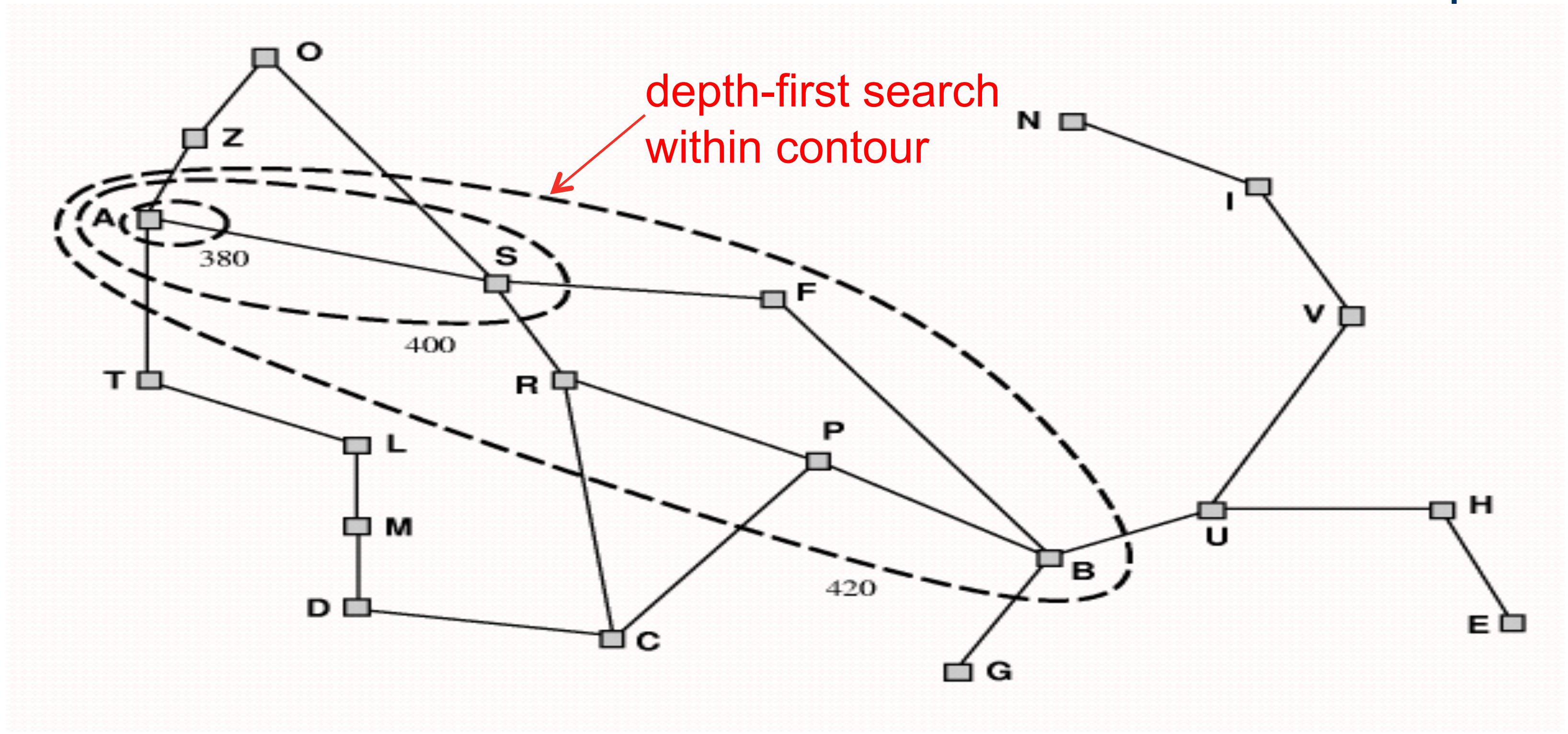
  2) Conduct a depth-first search based on minimal cost from current node, pruning any branch when its total cost function (g + h´) exceeds THRESHOLD. If a solution path is found during the search, return it as the optimal solution.

  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 underestimate and never decreasing.***

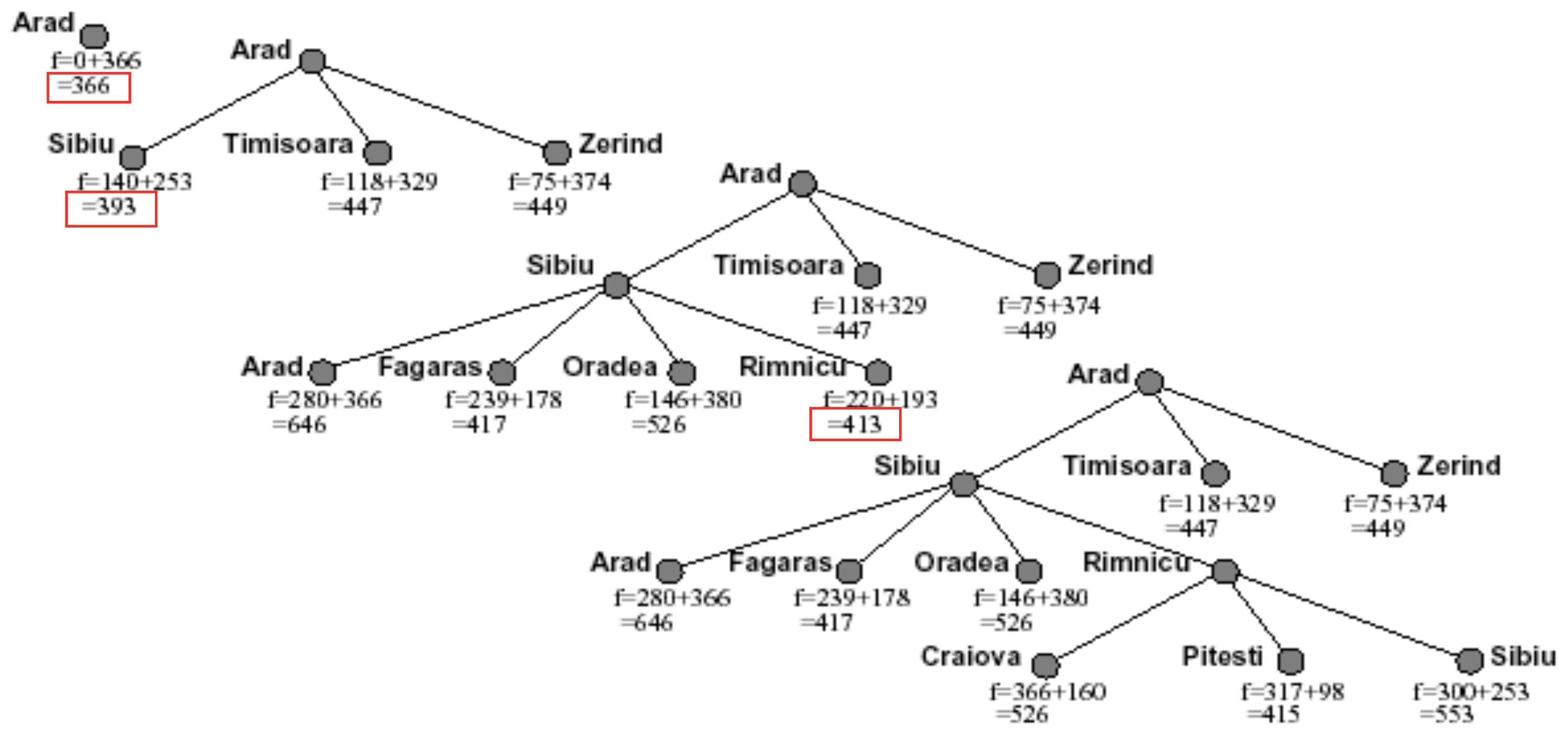
f-Cost Contours

- Monotonic heuristics allow us to view A* in terms of exploring



- 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*
  - Also 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*

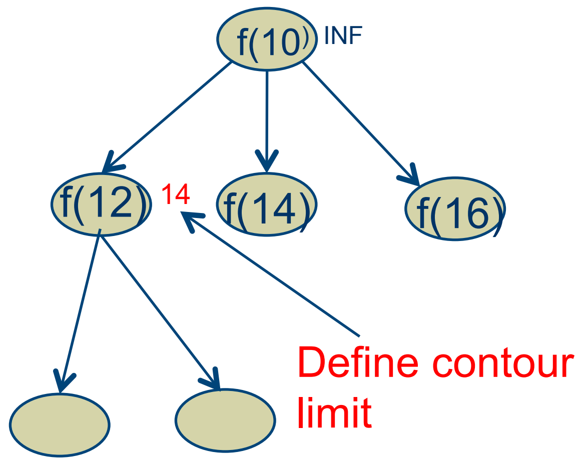
RBFS - Recursive Best-First Search

- Mimics best-first search with linear space
- Similar to recursive depth-first
  - Limits recursion by keeping track of the f-value of the best alternative path from any ancestor node – *one step look-ahead*
  - *If current node exceeds this value, recursion unwinds back to the alternative path – same idea as contour*
  - As recursion unwinds, replaces f-value of node with *best f-value* of children
    - *Allows to remember whether to re-expand path at later time*
- Exploits information gathered from previous searches about minimum f so as to focus further searches

RBFS - Recursive Best-First Search Algorithm

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    RBFS(MAKE-NODE(INITIAL-STATE[problem]), $\infty$)

function RBFS(problem, node, f-limit) returns a solution, or failure and a new f-cost limit
    if GOAL-TEST[problem](state) then return node
    successors ← EXPAND(node, problem)
    if successors is empty, then return failure, $\infty$
    for each s in successors do f[s] ← $\max(g(s) + h(s), f[node])$ ; Pathmax heuristic; guarantee monotonic f
    repeat
        best ← the lowest f-value node in successors
        if f[best] > f-limit then return failure, f[best]
        alternative ← the second-lowest f-value among successors     Defines next highest f-contour
        result, f[best] ← RBFS(problem, best, min(f-limit, alternative))     Recursive search on best successor, remember when to backup
        if result ≠ failure then return result
    end
```



Next lecture

- Other Time and Space Variations of A*
  - Finish off RBFS
  - SMA*
  - Anytime A*
  - RTA* (maybe if have time)