Artificial Intelligence

Topic 4

Informed search algorithms

- ◇ Best-first search
- ◇ Greedy search
- ◇ A* search
- ◇ Admissible heuristics
- ◇ Memory-bounded search
- ◇ IDA*
- ◇ SMA*

Reading: Russell and Norvig, Chapter 4, Sections 1–3

1. Informed (or best-first) search

Recall uninformed search:

- select nodes for expansion on basis of distance from start
- uses only information contained in the graph
- no indication of distance to go!

Informed search:

- select nodes on basis of some estimate of *distance to goal*!
- requires additional information — *evaluation function*, or *heuristic rules*
- choose "best" (most promising) alternative $\Rightarrow$ *best-first search*.

Implementation:

QUEUEINGFN = insert successors in decreasing order of desirability

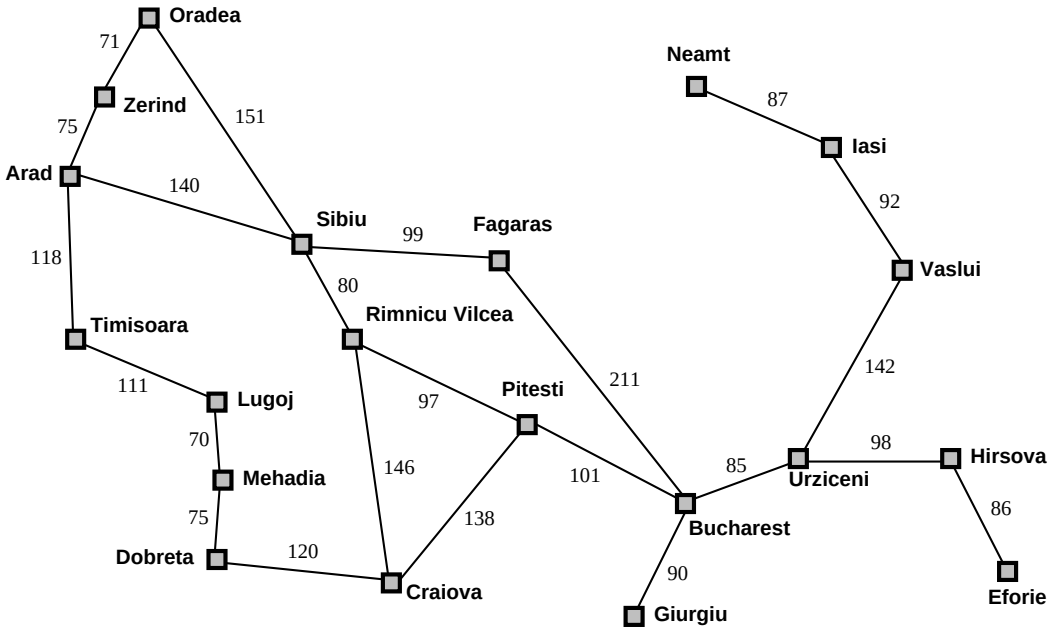
Examples:

- greedy search
- A* search

2. Greedy search

Assume we have an estimate of the distance to the goal.

For example, in our travelling to Bucharest problem, we may know straight-line distances to Bucharest...



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

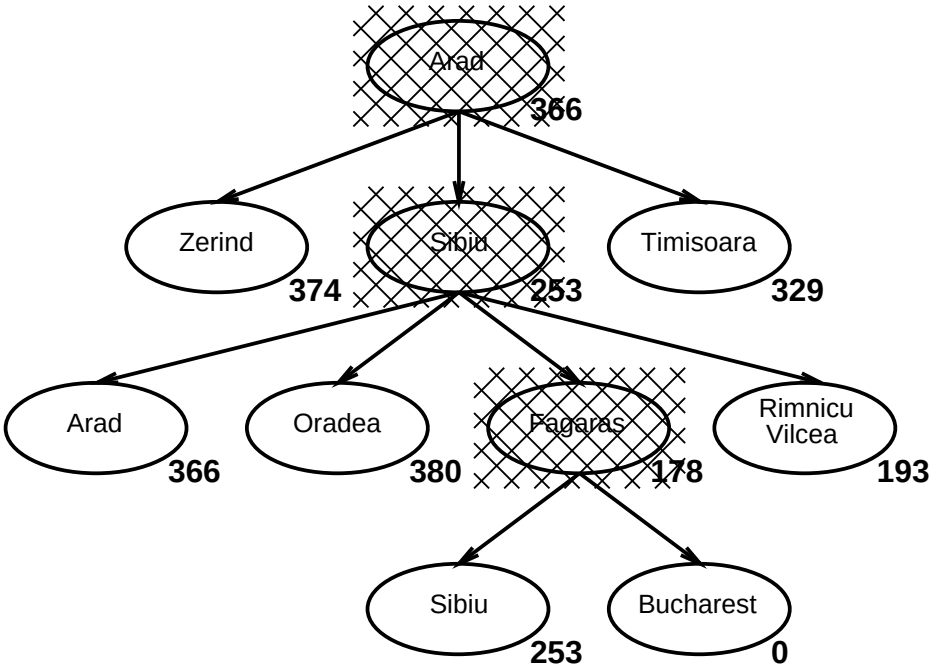
Greedy search always chooses to visit the candidate node with the smallest estimate

$\Rightarrow$ that which *appears* to be closest to goal

Evaluation function $h(n)$ (heuristic)
$=$ estimate of cost from n to $goal$

E.g., $h_{\mathrm{SLD}}(n) =$ straight-line distance from n to Bucharest

2. Greedy search



2. Greedy search

Complete? *No, in general.* e.g. can get stuck in loops.

Example: Iasi to Fagaras. . .

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time? $O(b^m)$, but a good heuristic can give dramatic improvement

Space? $O(b^m)$—keeps all nodes in memory

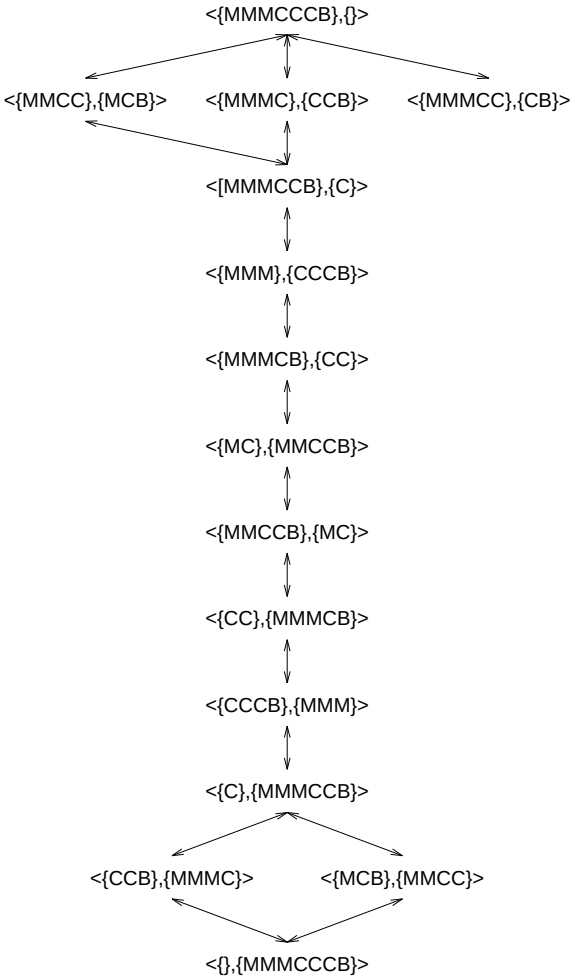
Optimal? *No*

2.1 Means-ends Analysis

Example of greedy search. (Used in SOAR problem solver.)

Heuristic: Pick operations that reduce as much as possible the "difference" between the intermediate state and goal state.

eg. Missionaries and cannibals



Indicates best choice in all states except for $\langle\{\text{MC}\},\{\text{MMCCB}\}\rangle$ and $\langle\{\text{MMCCB}\},\{\text{MC}\}\rangle$

3. A* search

Greedy search minimises estimated cost to goal, and thereby (hopefully) reduces search cost, but is neither optimal nor complete.

Uniform-cost search minimises path cost so far and is optimal and complete, but is costly.

Can we get the best of both worlds. . . ?

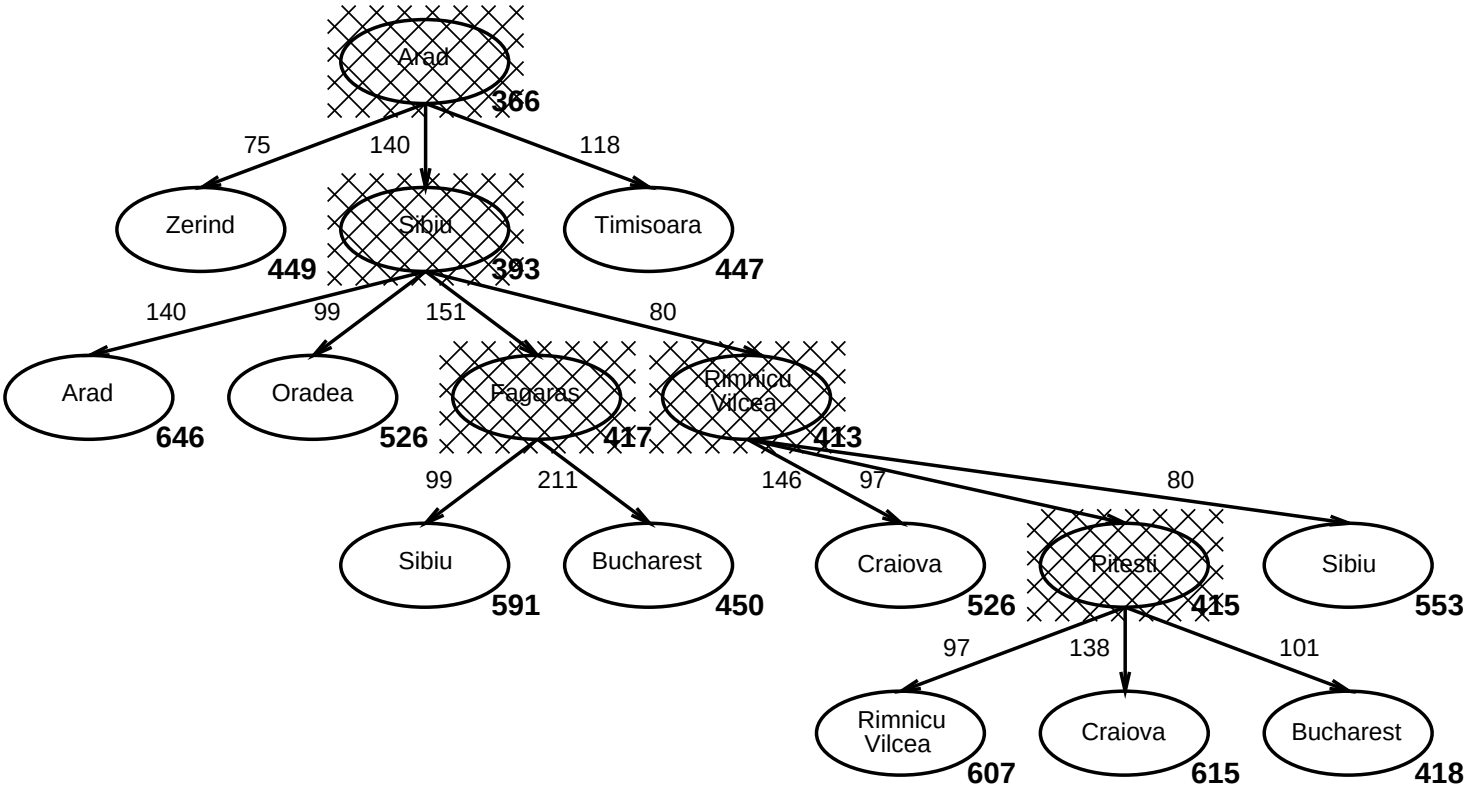
Yes! Just add the two together to get *estimate of total path length* of solution as our evaluation function. . .

Evaluation function

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

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

3. A* search


Arad
366
75
140
118
Zerind
449
Sibiu
393
Timisoara
447
140
99
151
80
Arad
646
Oradea
526
Fagaras
417
Rimnicu Vilcea
413
99
211
146
97
80
Sibiu
591
Bucharest
450
Craiova
526
Pitesti
415
Sibiu
553
97
138
101
Rimnicu Vilcea
607
Craiova
615
Bucharest
418

3. A* search

A heuristic h is *admissible* iff

$$h(n) \leq h^*(n) \quad \text{for all } n$$

where $h^*(n)$ is the *true* cost from n.

i.e. $h(n)$ *never overestimates*

e.g., $h_{\text{SLD}}(n)$ never overestimates the actual road distance

Can prove:

if $h(n)$ is admissible, $f(n)$ provides a complete and optimal search!

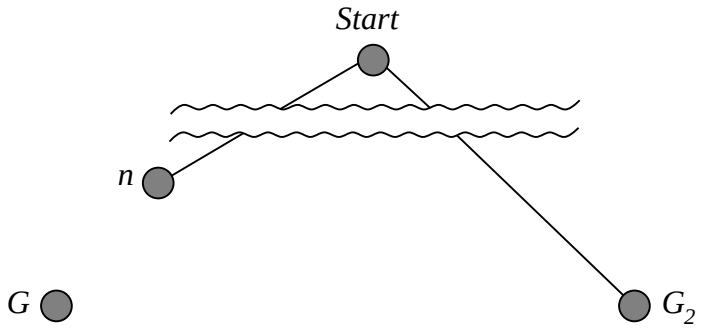
$\Rightarrow$ called A* search.

3.1 Optimality of A*

Theorem: A* search is optimal

Proof

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G_1.



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq g(n) + h(n) && \text{since } h \text{ is admissible} \\
&= f(n)
\end{aligned}
$$

Since $f(G_2) > f(n)$, A* will not select G_2 for expansion

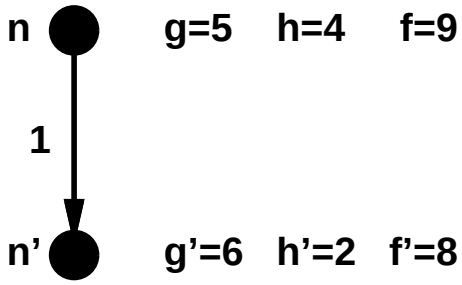
3.2 Monotonicity and the pathmax equation

To get a more intuitive view, we consider the f-values along any path.

For many admissible heuristics, f-values increase *monotonically* (see Romania problem).

For some admissible heuristics, f may be nonmonotonic — ie it may *decrease* at some points.

e.g., suppose n' is a successor of n



But $f' = 8$ is redundant!

$f(n) = 9 \quad \Rightarrow \quad$ true cost of a path through n is ≥ 9

$\Rightarrow \quad$ true cost of a path through n' is ≥ 9

Pathmax modification to A*:

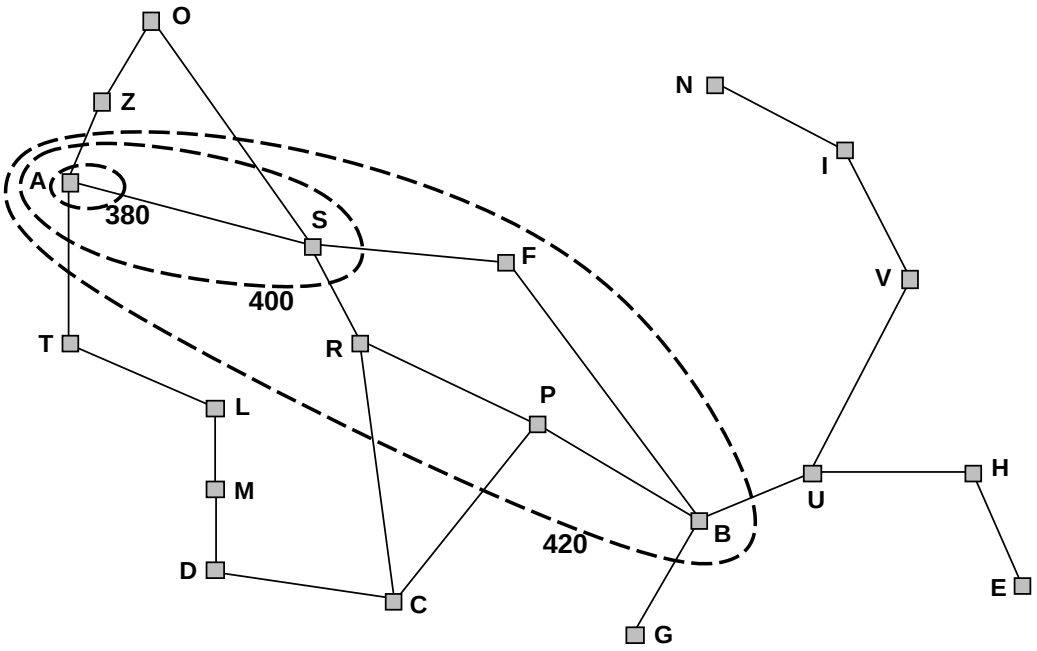
$$f(n') = max(g(n') + h(n'), f(n))$$

With pathmax, f is always increases monotonically $\cdots \rightsquigarrow$

3.3 Contours

Lemma: A* (with pathmax) expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes (cf. breadth-first/uniform-cost adds layers or "circles" — A* "stretches" towards goal)



If f^* is cost of optimal solution path:

- A* expands all nodes with $f(n) < f^*$
- A* expands some nodes with $f(n) = f^*$

Can see intuitively that A* is complete and optimal.

3.4 Properties of A*

Complete Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time Exponential in [relative error in h $\times$ length of soln.]

Space Keeps all nodes in memory (see below)

Optimal Yes—cannot expand f_{i+1} until f_i is finished

Among optimal algorithms of this type A* is *optimally efficient*!

ie. no other algorithm is guaranteed to expand fewer nodes.

"Proof"

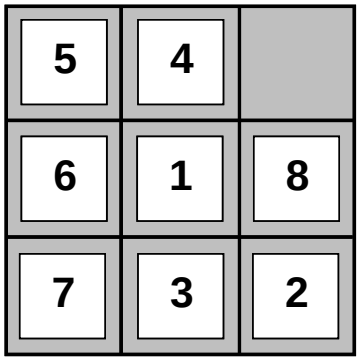
Any algorithm that does not expand all nodes in each contour may miss an optimal solution.

4. Admissible heuristics

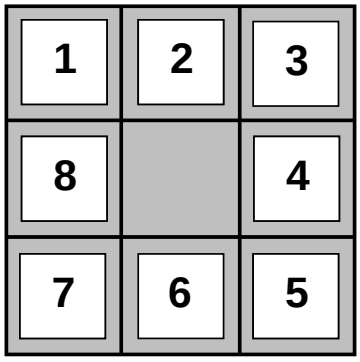
Straight line distance is an obvious heuristic for distance planning.
What about other problems?

This section $\Rightarrow$ examine heuristics in more detail.

E.g., two heuristics for the 8-puzzle:



Start State



Goal State

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)

$h_1(S) =$??

$h_2(S) =$??

Are both admissible?

4.1 Measuring performance

Quality of heuristic can be characterised by *effective branching factor* $b*$.

Assume:

- A* expands N nodes
- solution depth d

b^* is branching factor of uniform tree, depth d with N nodes:

$$N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

- tends to remain fairly constant over problem instances
- can be determined empirically

  $\Rightarrow$ fairly good guide to heuristic performance

a good heuristic would have b^* close to 1

Example

Effective branching factors for iterative deepening search and A* with h_1 and h_2 (averaged over 100 randomly generated instances of 8-puzzle for each solution length):

	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

- informed better than uninformed
- h_2 better than h_1

Is h_2 always better than h_1?

Yes!

We say h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ for all n (both admissible).

dominance $\Rightarrow$ *better efficiency*

— h_2 will expand fewer nodes on average than h_1

"Proof"

A* will expand all nodes n with $f(n) < f^*$.

$\Rightarrow$ A* will expand all nodes with $h(n) < f^* - g(n)$

But $h_2(n) \geq h_1(n)$ so all nodes expanded with h_2 will also be expanded with h_1 (h_1 may expand others as well).

always better to use an (admissible) heuristic function with higher values

4.3 Inventing heuristics — relaxed problems

- How can we come up with a heuristic?
- Can the computer do it automatically?

A problem is *relaxed* by reducing restrictions on operators

cost of exact solution of a relaxed problem is often a good heuristic for original problem

Example

- if the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution
- if the rules are relaxed so that a tile can move to *any adjacent square*, then $h_2(n)$ gives the shortest solution

Note: Must also ensure heuristic itself is not too expensive to calculate.

Extreme case: perfect heuristic can be found by carrying out search on original problem.

4.4 Automatic generation of heuristics

If problem is defined in suitable formal language $\Rightarrow$ may be possible to construct relaxed problems automatically.

eg. 8-puzzle operator description

$$\textit{A is adjacent to B} \ \& \ \textit{B is blank} \rightarrow \textit{can move from A to B}$$

Relaxed rules (eliminate preconditions)

$$\textit{A is adjacent to B} \rightarrow \textit{can move from A to B}$$

$$\textit{B is blank} \rightarrow \textit{can move from A to B}$$

$$\textit{can move from A to B}$$

ABSOLVER (Prieditis 1993)

- new heuristic for 8-puzzle better than any existing one
- first useful heuristic for Rubik's cube!

5. Memory-bounded Search

Good heuristics improve search, but many problems are still too hard.

Usually memory restrictions that impose a hard limit.

(eg. recall estimates for breadth-first search

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

)

This section — algorithms designed to save memory.

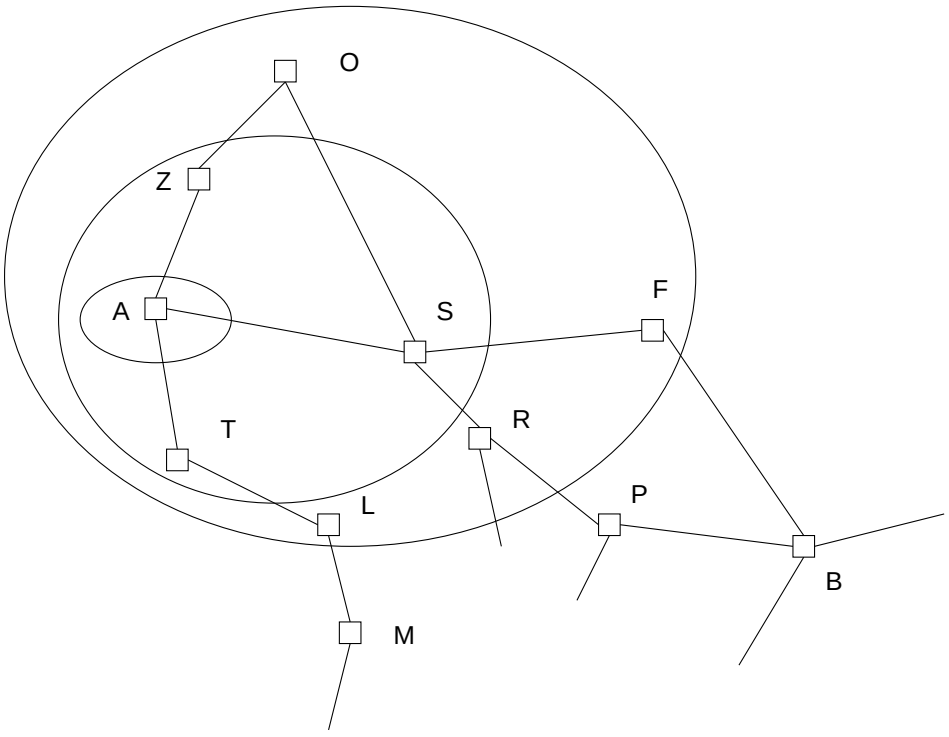
- IDA*
- SMA*

5.1 Iterative Deepening A* (IDA*)

Recall uninformed search

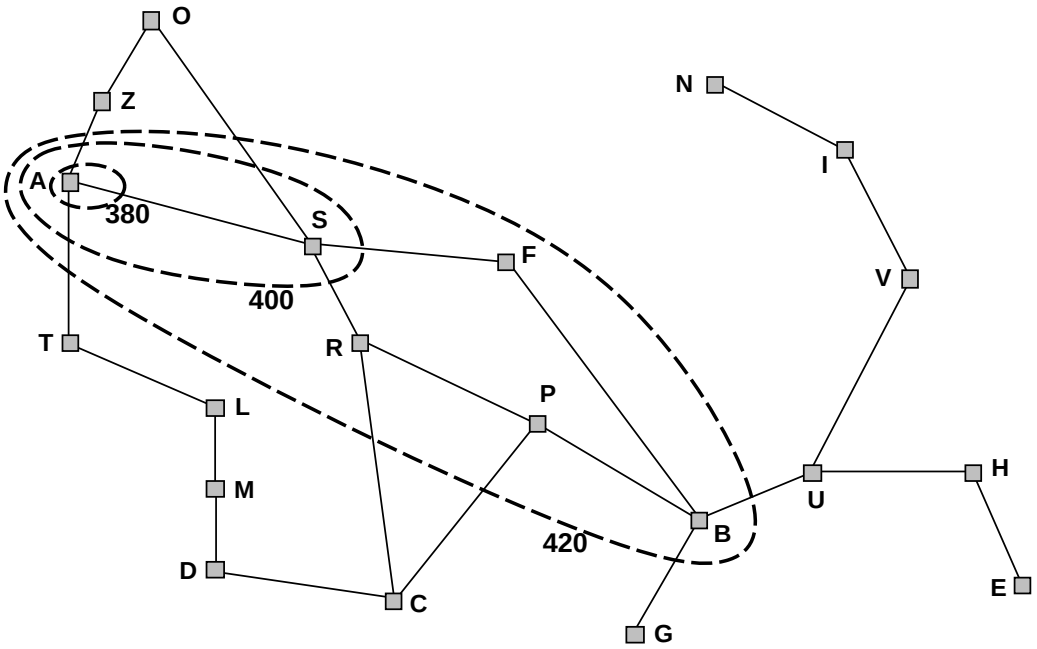
- uniform-cost/breadth-first search
  - + completeness, optimality
  - – exponential space usage
- depth-first
  - + linear space usage
  - – incomplete, suboptimal

Solution: iterative-deepening ⇒ explores "uniform-cost trees", or "contours", using linear space.



5.1 Iterative Deepening A* (IDA*)

Can we do the same with A*?



Contours more directed, but same technique applies!

Modify depth-limited search to use f-cost limit, rather than depth limit $\Rightarrow$ *IDA**

Complete? *Yes* (with admissible heuristic)
Optimal? *Yes* (with admissible heuristic)
Space? *Linear* in path length
Time? *?*

Time complexity of IDA*

Depends on number of different values f can take on.

- small number of values, few iterations
  eg. 8-puzzle
- many values, many iterations
  eg. Romania example, each state has different heuristic $\Rightarrow$ only one extra town in each contour

Worst case: A* expands N nodes, IDA* goes through N iterations

$$1 + 2 + \cdots + N \Rightarrow \quad O(N^2)$$

(Recall N in turn is exponential in $d \times$ relative error in h.)

How does this compare with ID?

5.1 Iterative Deepening A* (IDA*)

A solution: increase f-cost limit by fixed amount ϵ in each iteration

$\Rightarrow$ returns solutions at worst ϵ worse than optimal

Called ϵ-*admissible*.

IDA* was first memory-bounded optimal heuristic algorithm and solved many practical problems.

5.2 Simplified memory-bounded A* (SMA*)

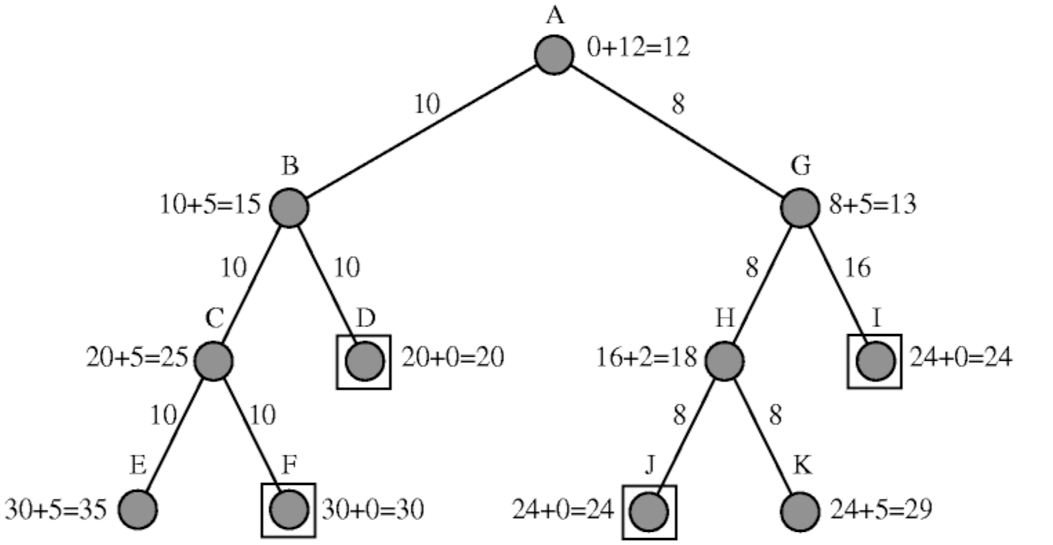
Uses all available memory.

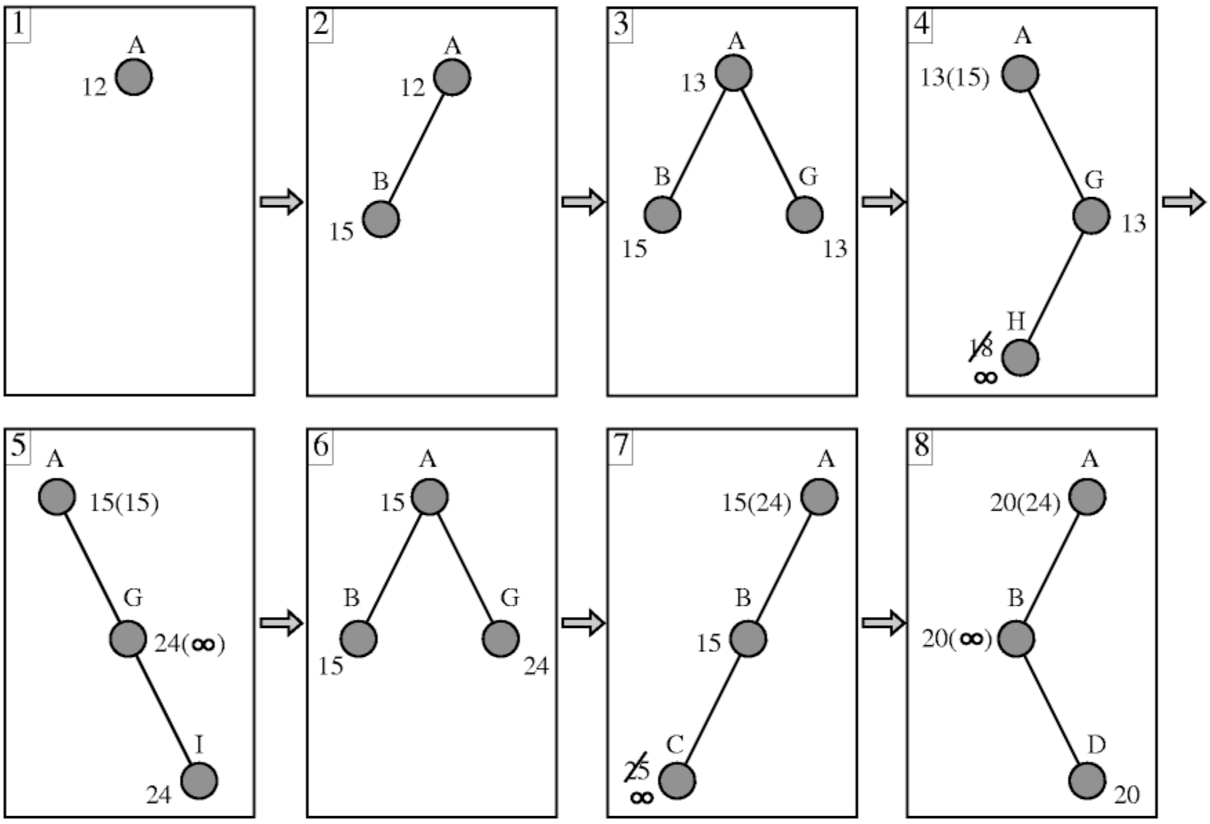
How does it work?

- Need to generate successor nodes but no memory left $\Rightarrow$ drop, or *"forget"*, least promising nodes.
- Keep record of best f-cost of forgotten nodes in ancestor.
- Only regenerate nodes if all more promising options are exhausted.

Example (values of forgotten nodes in parentheses) $\cdots \Rightarrow$

5.2 Simplified memory-bounded A* (SMA*)





5.2 Simplified memory-bounded A* (SMA*)

Performance

- Complete if available memory is sufficient to store the *shallowest* solution path.
- Optimal if available memory is sufficient to store the *shallowest* solution path. Otherwise best "solution" given available memory.

Summary

- Solves significantly more difficult problems than A*.
- Performs well on highly-connected state spaces and real-valued heuristics on which A* has difficulty.
- Susceptible to continual "switching" between candidate solution paths.
  ie. *limit in memory can lead to intractible computation time*

The End