CSE 473: Artificial Intelligence
Autumn 2018

Heuristics & Pattern Databases for Search

Steve Tanimoto

Presented by Emilia Gan

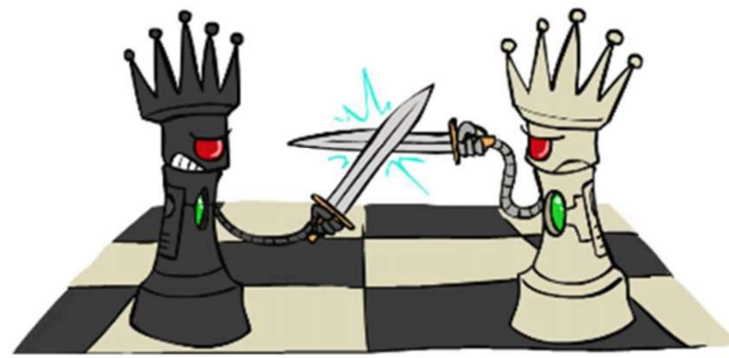
With thanks to Dan Weld, Dan Klein, Richard Korf, Stuart Russell, Andrew Moore, and Luke Zettlemoyer

*With modifications from various sources (as noted on individual slides) E. Gan Fall '18

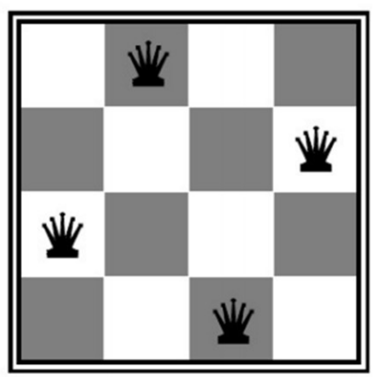
Recap: Search Problem

- States
  - configurations of the world
- Successor function:
  - function from states to lists of (state, action, cost) triples
- Start state
- Goal test

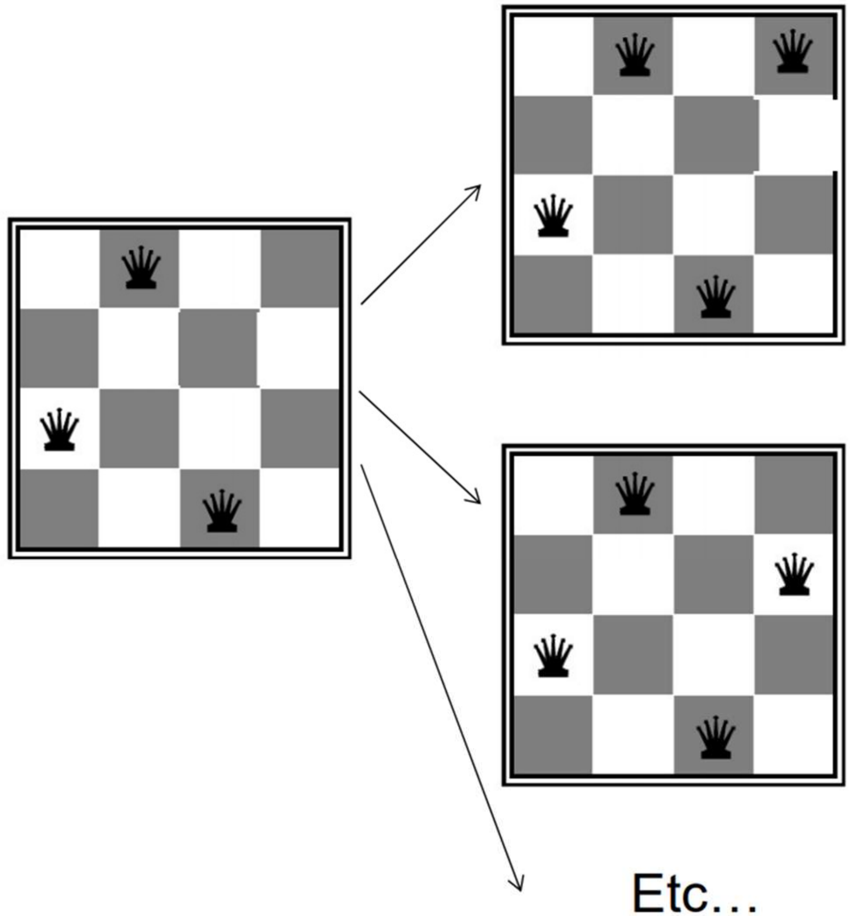
N-Queens as Search?



- Given N x N chess board
- Can you place N queens so they don't fight?



States are Board Positions

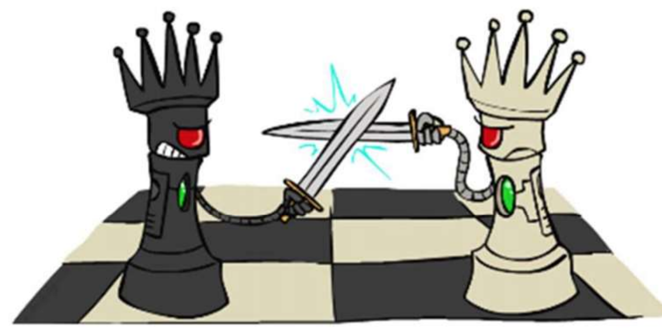


Search Methods

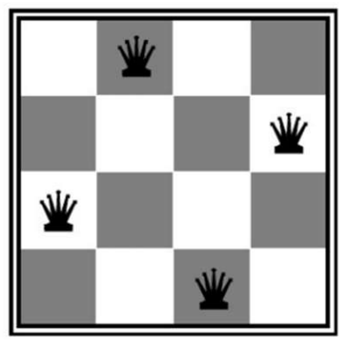
- Depth first search (DFS)
- Breadth first search (BFS)
- Iterative deepening depth-first search (IDS)
- Best first search
- Uniform cost search (UCS)
- Greedy search
- A*
- Iterative Deepening A* (IDA*)
- Beam search, hill climbing
- **Stochastic Search**
- **Constraint Satisfaction**

Heuristic search

IDA* for N-Queens?



- Given N x N chess board
- Can you place N queens so they don't fight?



Cool picture from Dan Klein & Pieter Abeel ai.berkeley.edu

Best-First Search

- Generalization of breadth-first search
- Fringe = ***Priority*** queue of nodes to be explored
- Cost function f(n) applied to each node

```
Add initial state to priority queue
While queue not empty
        Node = head(queue)
        If goal?(node) then return node
        Add children of node to queue
```

"expanding the node"

Greedy Best First Algorithm

- Recall: BFS and DFS pick the next node off the frontier based on which was "first in" or "last in".
- Greedy Best First picks the "best" node according to some rule of thumb, called a *heuristic*.

Definition: A *heuristic* is an approximate measure of how close you are to the target.

A heuristic guides you in the right direction.

https://cs.stanford.edu/people/abisee/gs.pdf

A*

- Expands the path with the lowest cost + h value on the frontier
- The frontier is implemented as a priority queue ordered by $f(p) = cost(p) + h(p)$

Admissibility of a heuristic

Def.:

Let c(n) denote the cost of the optimal path from node n to any goal node. A search heuristic h(n) is called admissible if $h(n) \leq c(n)$ for all nodes n, i.e. if for all nodes it is an underestimate of the cost to any goal.

https://www.cs.ubc.ca/~mack/CS322/lectures/2-Search6.pdf

The main drawback of the presented best-first graph search algorithms is their space complexity.

Idea: use the concepts of iterative-deepening DFS

- bounded depth-first search with increasing bounds
- instead of depth we bound f
  (in this chapter $f(n) := g(n) + h(n.\text{state})$ as in A*)

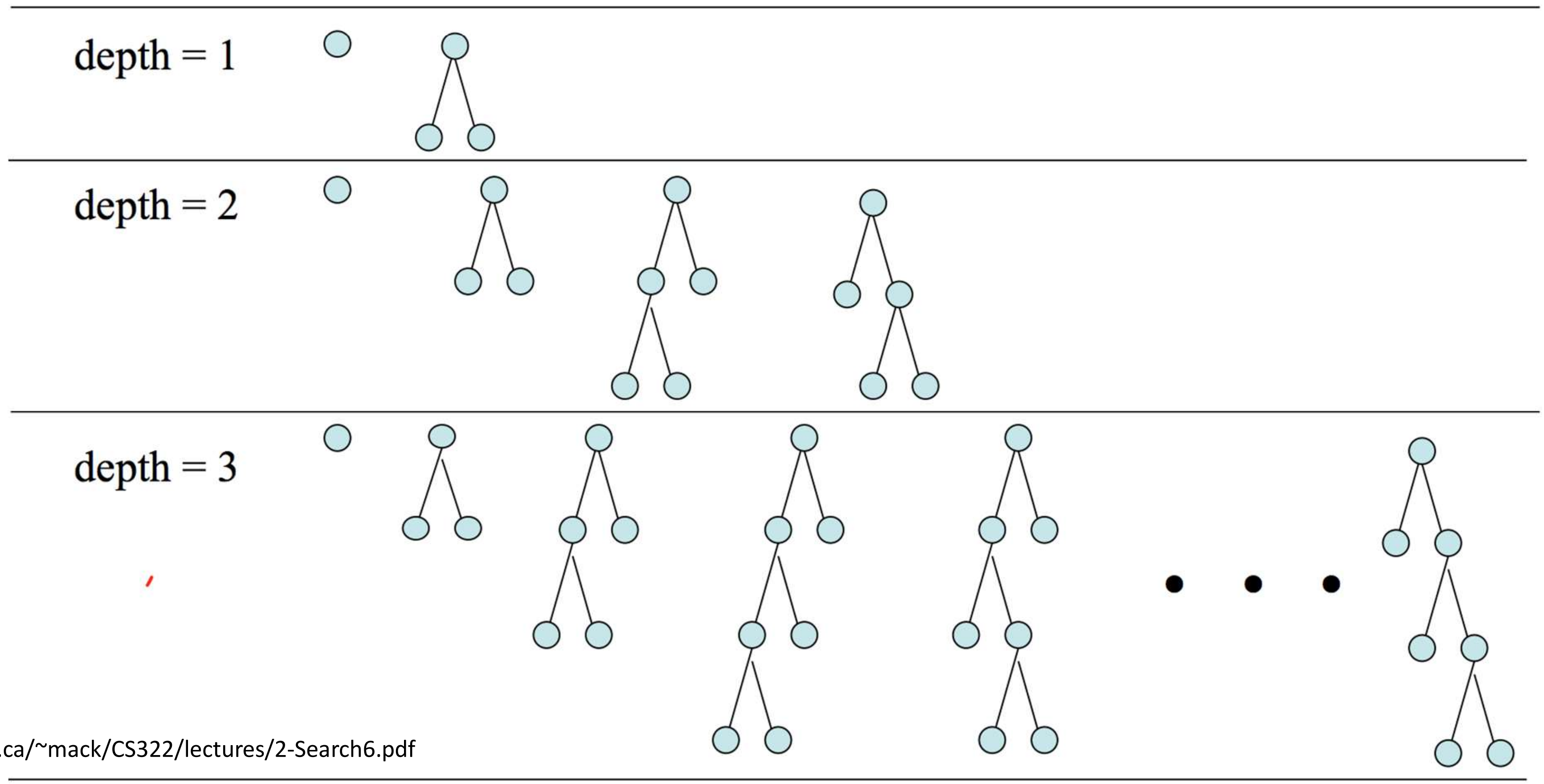
⇝ IDA* (iterative-deepening A*)

- tree search, unlike the previous best-first search algorithms

https://ai.dmi.unibas.ch/_files/teaching/fs17/ai/slides/ai17.pdf

Iterative Deepening DFS (IDS) in a Nutshell

- Use DFS to look for solutions at depth 1, then 2, then 3, etc
  - For depth D, ignore any paths with longer length
  - Depth-bounded depth-first search



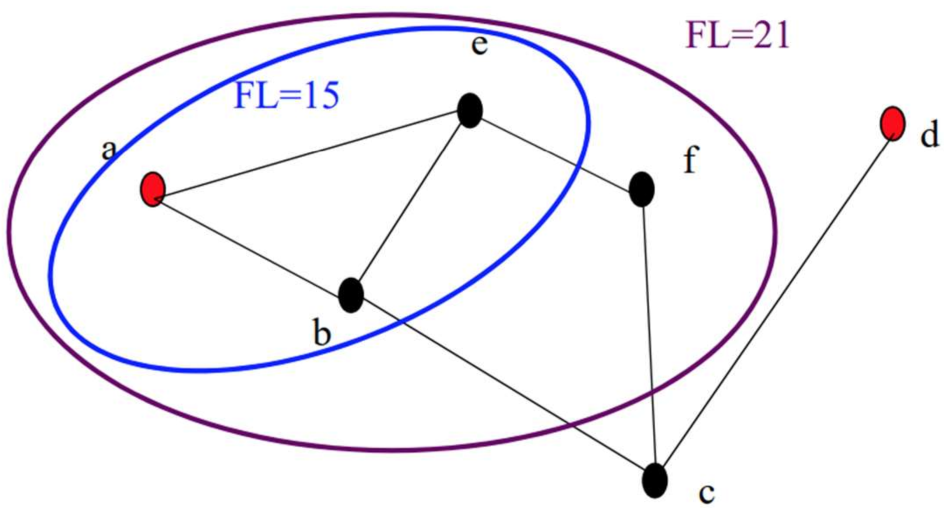
(Heuristic) Iterative Deepening: IDA*

- Like Iterative Deepening DFS
  - But the depth bound is measured in terms of the f value
- If you don't find a solution at a given depth
  - Increase the depth bound:
    to the minimum of the f-values that exceeded the previous bound

https://www.cs.ubc.ca/~mack/CS322/lectures/2-Search6.pdf

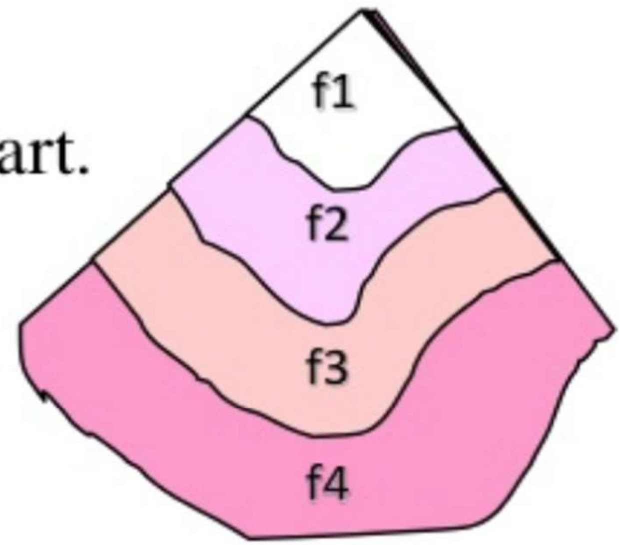
Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f-limit
  - Start with f-limit = h(start)
  - Prune any node if f(node) > f-limit
  - Next f-limit = min-cost of any node pruned



IDA*(Iterative Deepening A*) Search

- Perform depth-first search LIMITED to some f-bound.
- <u>If</u> goal found: ok.
- <u>Else</u>: increase f-bound and restart.



- How to establish the f-bounds?
- - initially: f(S)
- generate all successors
- record the minimal f(succ) > f(S)
- Continue with minimal f(succ) instead of f(S)

https://www.slideshare.net/hemak15/lecture-17-iterative-deepening-a-star-algorithm

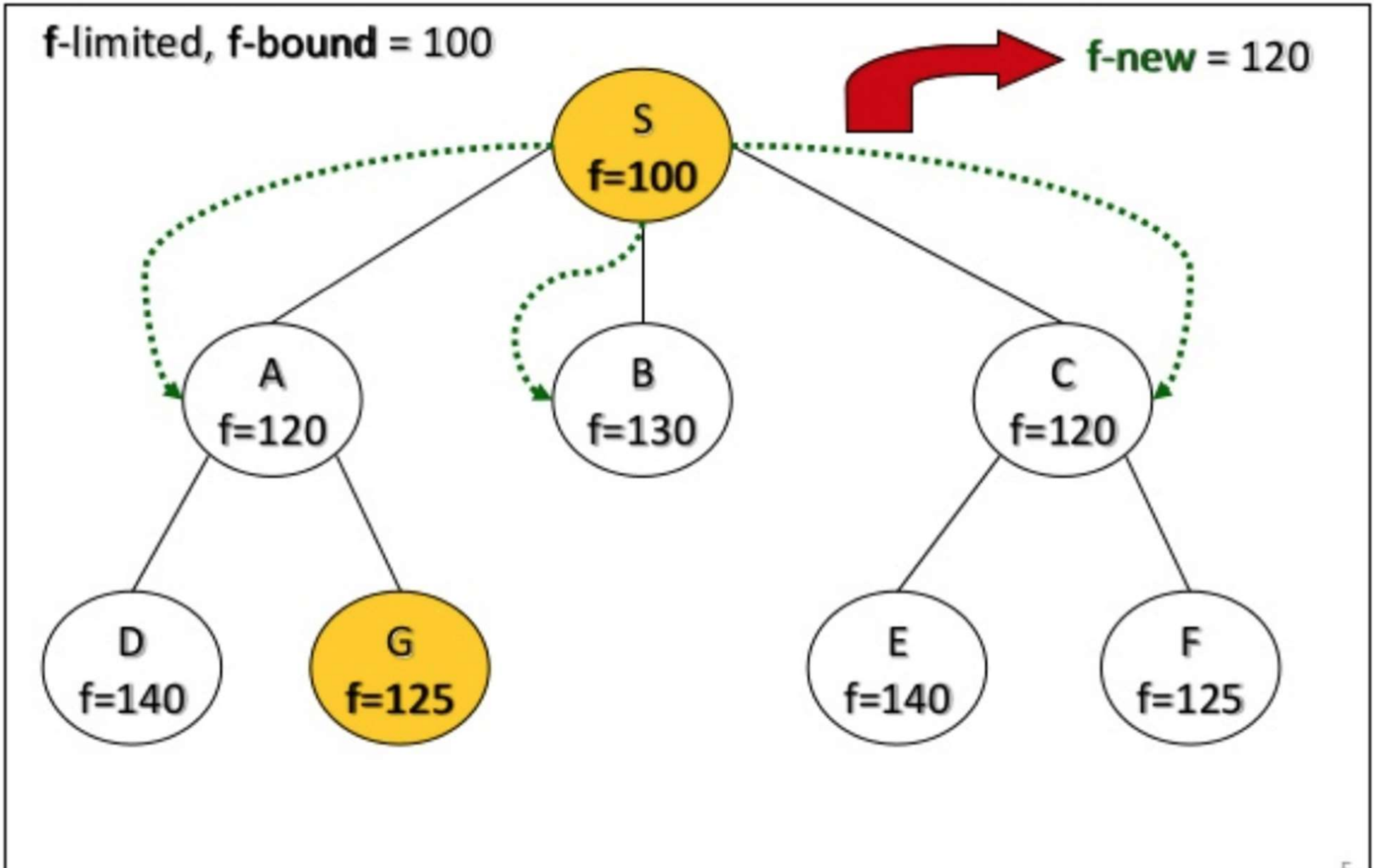
```
path              current search path (acts like a stack)
node              current node (last node in current path)
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
cost(node, succ)  step cost function
is_goal(node)     goal test
successors(node)  node expanding function, expand nodes ordered by g + h(node)
ida_star(root)    return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure
```

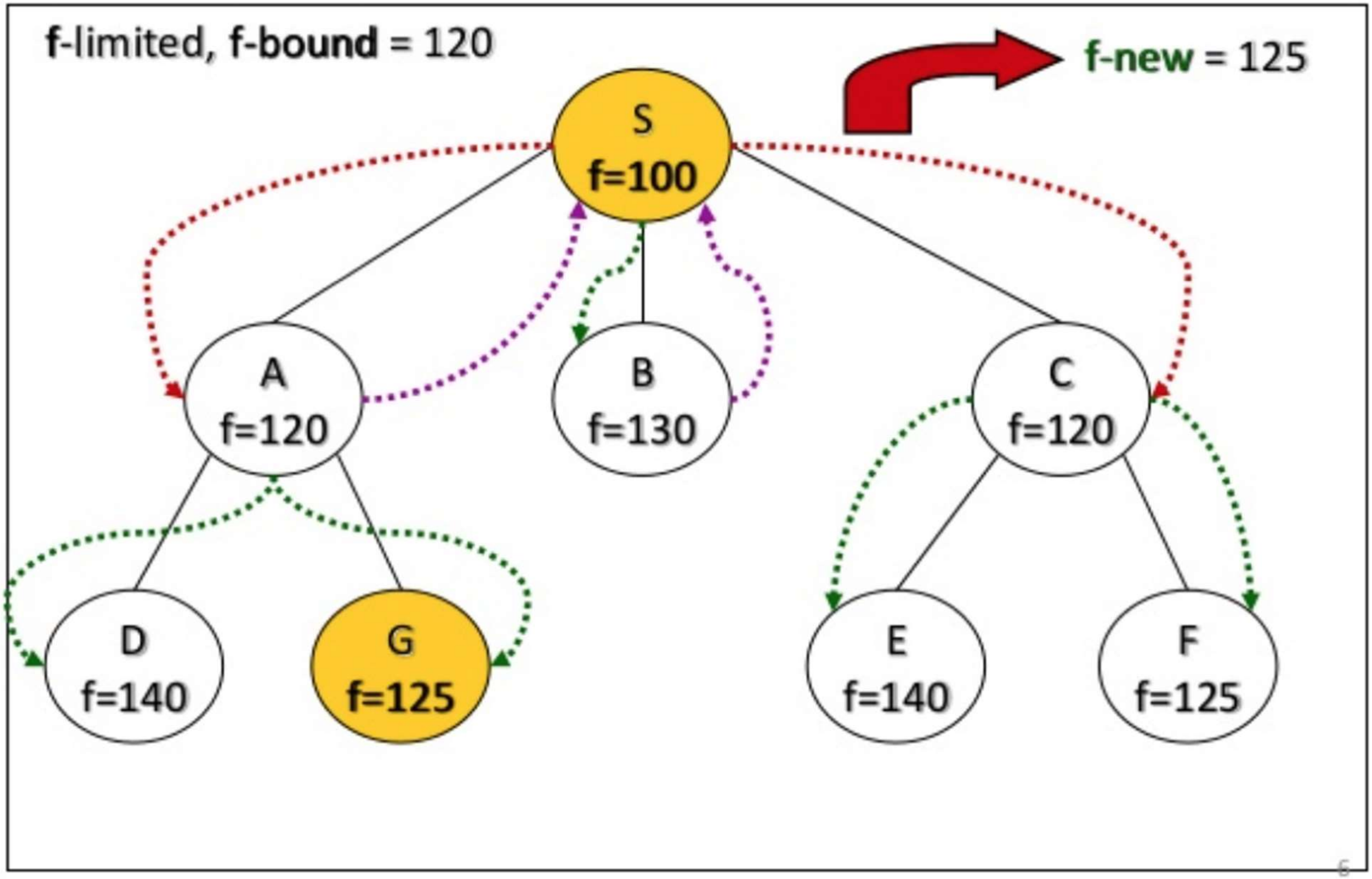
https://en.wikipedia.org/wiki/Iterative_deepening_A*

```
function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t
      path.pop()
    end if
  end for
  return min
end function
```

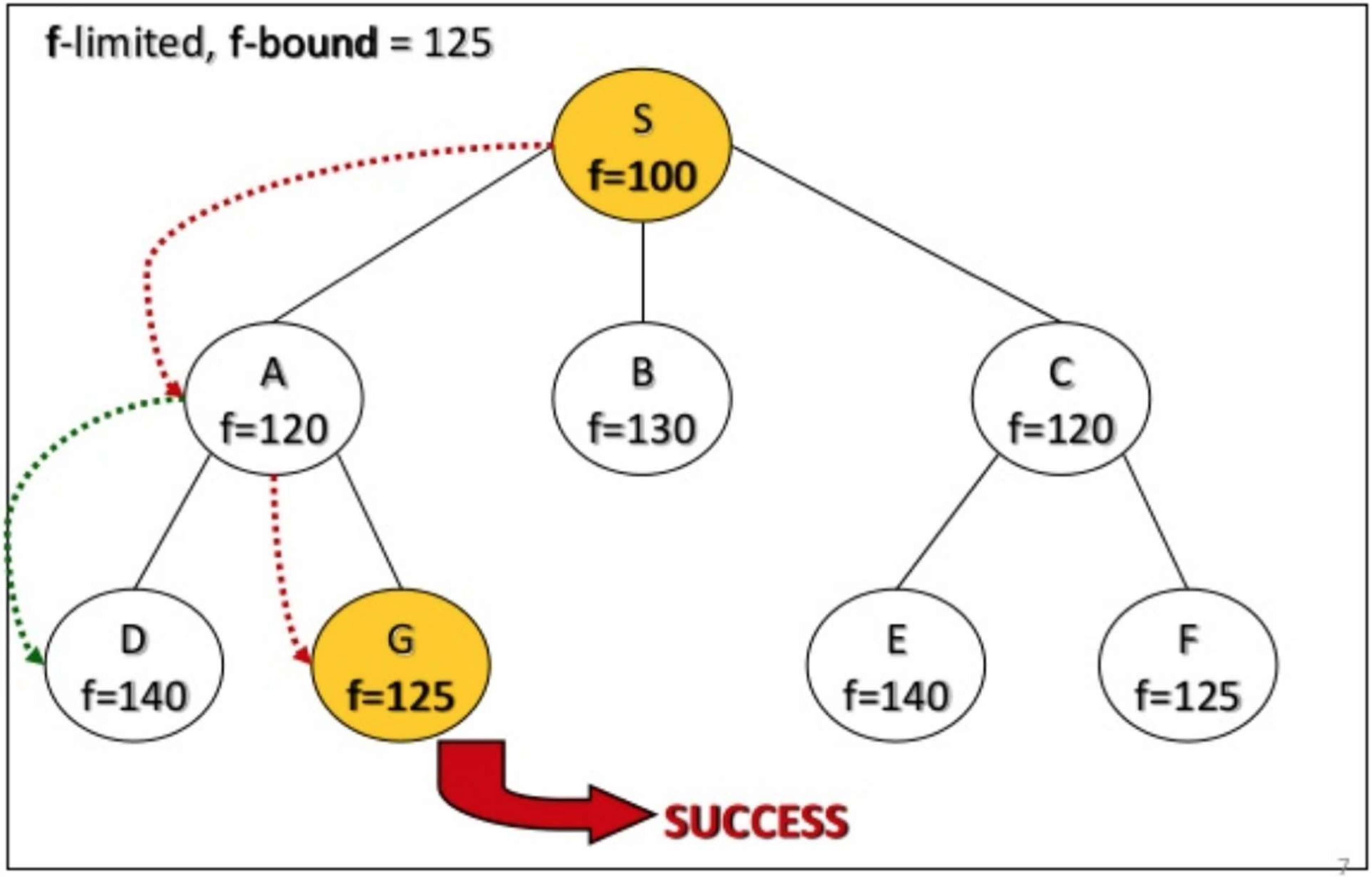
https://en.wikipedia.org/wiki/Iterative_deepening_A*



https://www.slideshare.net/hemak15/lecture-17-iterative-deepening-a-star-algorithm



https://www.slideshare.net/hemak15/lecture-17-iterative-deepening-a-star-algorithm



https://www.slideshare.net/hemak15/lecture-17-iterative-deepening-a-star-algorithm

IDA* Analysis

- Complete & Optimal (a la A*)
- Space usage $\propto$ depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to A*
  - Depends on # unique values of heuristic function
  - In 8 puzzle: few values $\Rightarrow$ close to # A* expands
  - In eastern-europe travel: each f value is unique

    $\Rightarrow$ $1+2+\ldots+n = O(n^2)$ where n=nodes A* expands

    if n is too big for main memory, n^2 is too long to wait!
- Generates duplicate nodes in cyclic graphs

- IDA* is a tree search variant of A*
  based on iterative deepening depth-first search
- main advantage: low space complexity
- disadvantage: repeated work can be significant
- most useful when there are few duplicates

https://ai.dmi.unibas.ch/_files/teaching/fs17/ai/slides/ai17.pdf

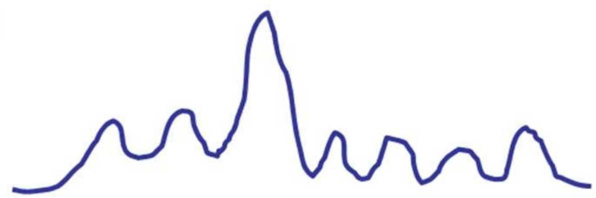
Beam Search

- Idea
  - Best first
  - But discard all but N best items on priority queue
- Evaluation
  - Complete?
    No
  - Time Complexity?
    $O(b^d)$
  - Space Complexity?
    $O(b + N)$

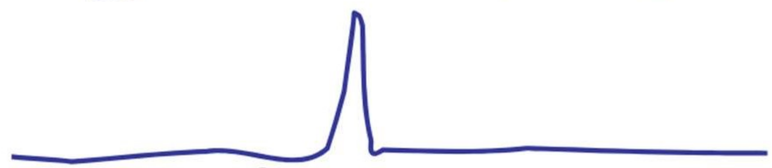
Hill Climbing

"Gradient ascent"

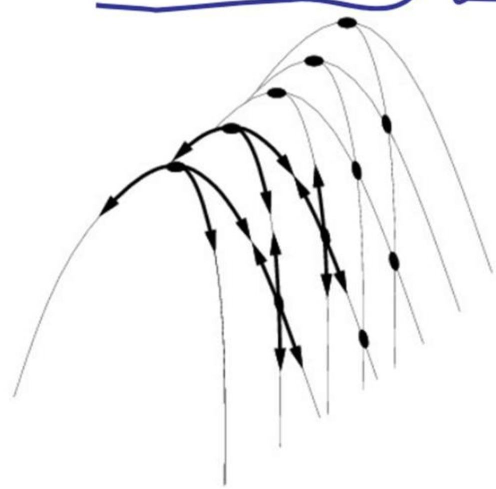
- Idea
  - Always choose best child; no backtracking
  - Beam search with |queue| = 1
- Problems?
  - Local maxima



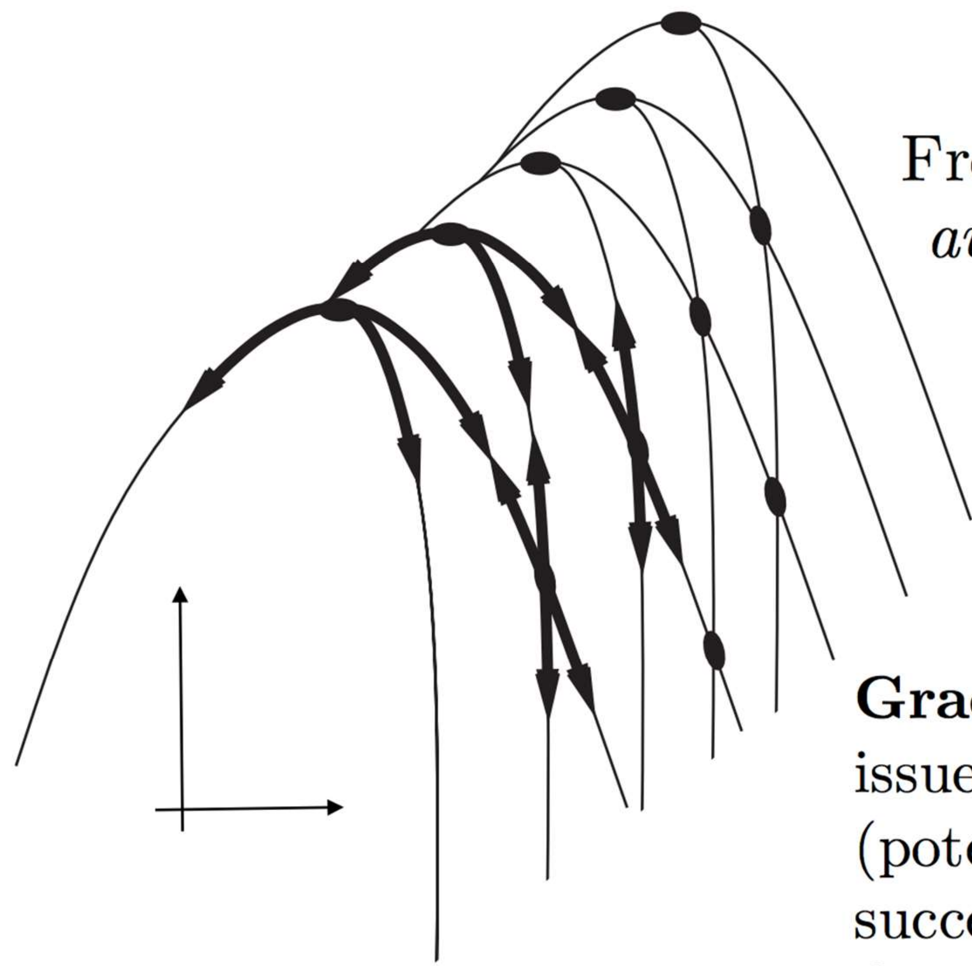
  - Plateaus



  - Diagonal ridges



HILL-CLIMBING CAN GET STUCK!



Diagonal ridges:

From each local maximum all the *available* actions point downhill, but there is an uphill path!

Zig-zag motion, very long ascent time!

Gradient ascent doesn't have this issue: *all* state vector components are (potentially) changed when moving to a successor state, climbing can follow the direction of the ridge

http://www.cs.cmu.edu/~arielpro/15381f16/c_slides/781f16-2a.pdf

Heuristics

It's what makes search actually work

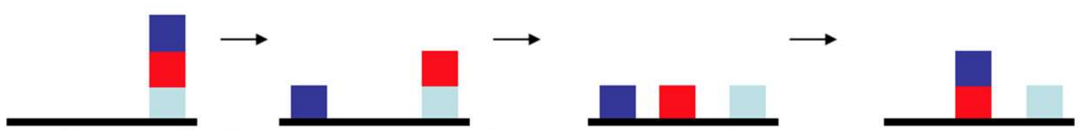
Admissible Heuristics

- $f(x) = g(x) + h(x)$
- g: cost so far
- h: underestimate of remaining costs

Where do heuristics come from?

Relaxed Problems

- Derive admissible heuristic from exact cost of a solution to a relaxed version of problem
  - For blocks world, distance = # move operations
  - heuristic = number of misplaced blocks
  - *What is relaxed problem?*

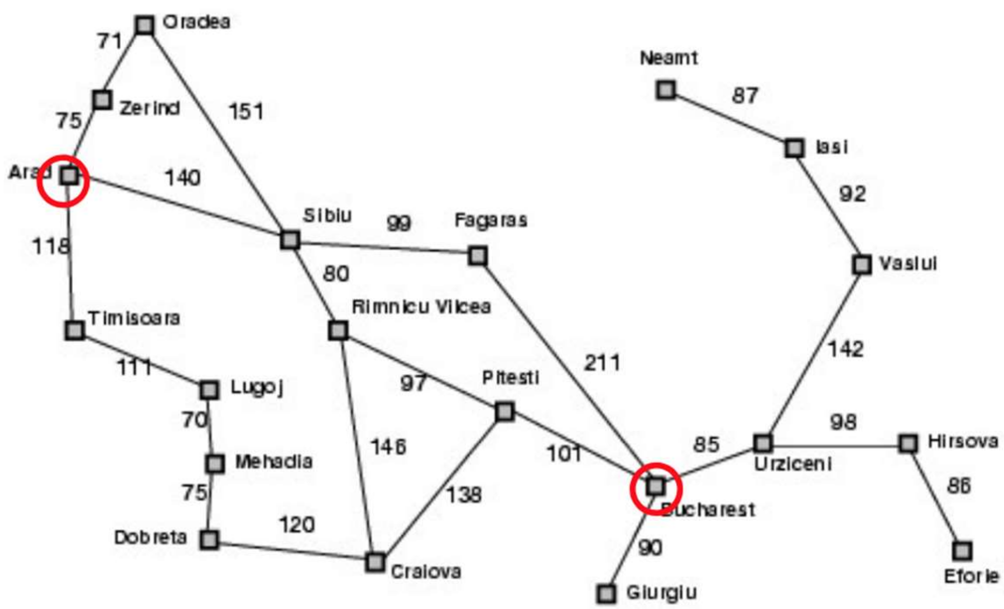


out of place = 2,   true distance to goal = 3

- Cost of optimal soln to relaxed problem ≤ cost of optimal soln for real problem

What’s being relaxed?

Heuristic = Euclidean distance

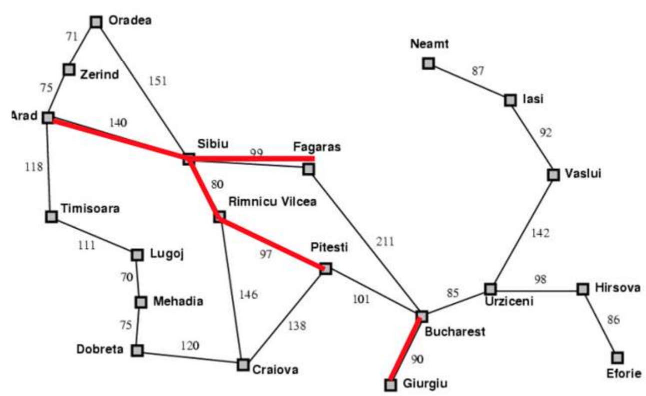


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Traveling Salesman Problem

Objective: shortest path visiting every city



What can be Relaxed?

A number of strategies -- beyond the scope of this lecture, but see:
http://www.math.chalmers.se/Math/Grundutb/CTH/mve165/1112/Lectures/TSPLecture-120426.pdf
for more information, if you’re curious.

Heuristics for eight puzzle

7	2	3
5	1	6
8	3	

→

1	2	3
4	5	6
7	8	

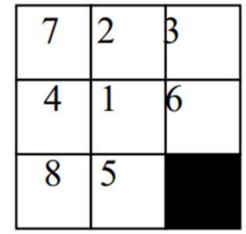
start goal

- What can we relax?

h1 = number of tiles in wrong place

h2 = Σ distances of tiles from correct loc

Importance of Heuristics



h1 = number of tiles in wrong place

D	IDS	A*(h1)
2	10	6
4	112	13
6	680	20
8	6384	39
10	47127	93
12	364404	227
14	3473941	539
18		3056
24		39135

Importance of Heuristics

7	2	3
4	1	6
8	5	

h1 = number of tiles in wrong place

h2 = Σ distances of tiles from correct loc

D	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
18		3056	363
24		39135	1641

Decrease effective branching factor

Need More Power!

Performance of Manhattan Distance Heuristic

▪ 8 Puzzle	< 1 second
▪ 15 Puzzle	1 minute
▪ 24 Puzzle	65000 years

Need even better heuristics!

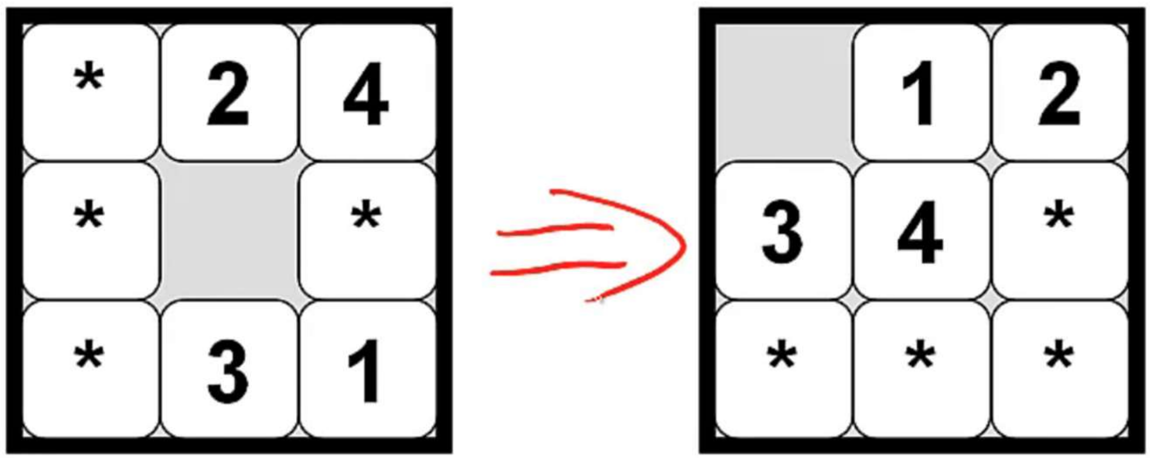
Adapted from Richard Korf presentation

Subgoal Interactions

- Manhattan distance assumes
  - Each tile can be moved independently of others
- Underestimates because
  - Doesn't consider interactions between tiles

1	2	3
4	6	5
7	8	

Adapted from Richard Korf presentation



cost of the optimal solution of sub-problem
≤ cost of the optimal solution of complete problem

Open Education Edinburgh
Published on Feb 26, 2014

https://www.youtube.com/watch?v=HZWV4uOJWk8

Pattern Databases

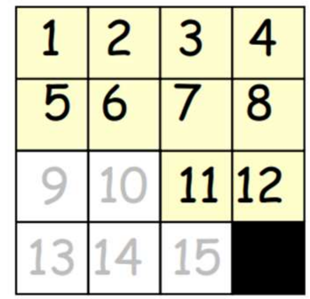
- idea: pre-compute and store the solution costs for all possible sub-problems in database
- computing heuristic = DB lookup
- construct DB by searching backwards from the goal state and recording costs
  - very expensive operation, but needs to be computed only once

Open Education Edinburgh
Published on Feb 26, 2014

https://www.youtube.com/watch?v=HZWV4uOJWk8

Pattern Databases

[Culberson & Schaeffer 1996]



- Pick any subset of tiles
  - E.g., 3, 7, 11, 12, 13, 14, 15
  - (or as drawn)
- Precompute a table
  - Optimal cost of solving just these tiles
  - For all possible configurations
    - 57 Million in this case
  - Use A* or IDA*
    - State = position of just these tiles (& blank)

Adapted from Richard Korf presentation

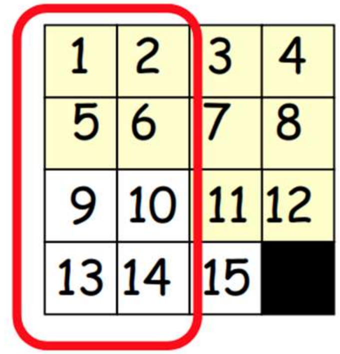
Using a Pattern Database

- As each state is generated
  - Use position of chosen tiles as index into DB
  - Use lookup value as heuristic, h(n)

  - Admissible?

Combining Multiple Databases

- Can choose another set of tiles
  - Precompute multiple tables
- How combine table values?



- E.g. Optimal solutions to Rubik's cube
  - First found w/ IDA* using pattern DB heuristics
  - Multiple DBs were used (dif cubie subsets)
  - Most problems solved optimally in 1 day
  - Compare with *574,000 years* for IDDFS

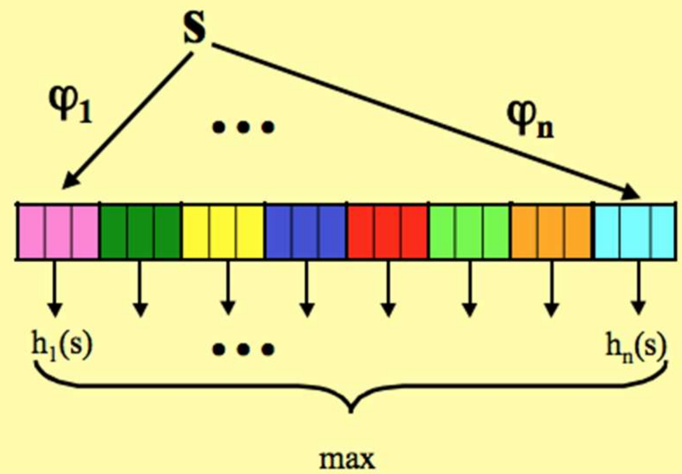
Adapted from Richard Korf presentation

Efficiency

Time for the preprocessing to create a PDB is usually negligible compared to the time to solve one problem-instance with no heuristic.

Memory is the limiting factor.

Many small pattern databases



https://webdocs.cs.ualberta.ca/~holte/CMPUT651/pdb20041031.pdf

Rubik's Cube

PDB Size	n	Nodes Generated
13,305,600	8	2,654,689
17,740,800	6	**2,639,969**
26,611,200	4	3,096,919
53,222,400	2	5,329,829
106,444,800	1	**61,465,541**

Summary

State Space	Best n	Ratio
(3x3)-puzzle	**10**	**3.85**
9-pancake	**10**	**8.59**
(8,4)-Topspin (3 ops)	**9**	**3.76**
(8,4)-Topspin (8 ops)	**9**	**20.89**
(3x4)-puzzle	**21+**	**185.5**
Rubik's Cube	**6**	**23.28**
15-puzzle (additive)	**5**	**2.38**
24-puzzle (additive)	**8**	**1.6 to 25.1**

$$\text{RATIO} = \frac{\text{\#nodes generated using } \textbf{one} \text{ PDB of size M}}{\text{\#nodes generated using } \textbf{n} \text{ PDBs of size M/n}}$$

https://webdocs.cs.ualberta.ca/~holte/CMPUT651/pdb20041031.pdf

Drawbacks of Standard Pattern DBs

- Since we can only take *max*
  - Diminishing returns on additional DBs
- Would like to be able to ***add*** values

Adapted from Richard Korf presentation

Disjoint Pattern DBs

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- Partition tiles into disjoint sets
  - For each set, precompute table
    - E.g. 8 tile DB has 519 million entries
    - And 7 tile DB has 58 million
- During search
  - Look up heuristic values for each set
  - *Can add values without overestimating!*

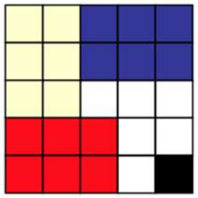
  - Manhattan distance is a special case of this idea where each set is a single tile

Adapted from Richard Korf presentation

Performance

- 15 Puzzle: 2000x speedup *vs* Manhattan dist
  - IDA* with the two DBs shown previously solves 15 Puzzles optimally in 30 milliseconds

- 24 Puzzle: 12 million x speedup *vs* Manhattan
  - IDA* can solve random instances in 2 days.
  - Requires 4 DBs as shown
    - Each DB has 128 million entries
  - Without PDBs: 65,000 years



Adapted from Richard Korf presentation