Uninformed Search

Chapter 3

(Based on slides by Stuart Russell, Subbarao Kambhampati, Dan Weld, Oren Etzioni, Henry Kautz, Richard Korf, and other UW-AI faculty)

What is a State?

- All information about the environment
- All information necessary to make a decision for the task at hand.

Agent’s Knowledge Representation

Type	State representation	Focus
Atomic	States are indivisible; No internal structure	Search on atomic states;
Propositional (aka Factored)	States are made of state variables that take values (Propositional or Multi-valued or Continuous)	Search+inference in logical (prop logic) and probabilistic (bayes nets) representations
Relational	States describe the objects in the world and their inter-relations	Search+Inference in predicate logic (or relational prob. Models)
First-order	+functions over objects	Search+Inference in first order logic (or first order probabilistic models)

Illustration with Vacuum World

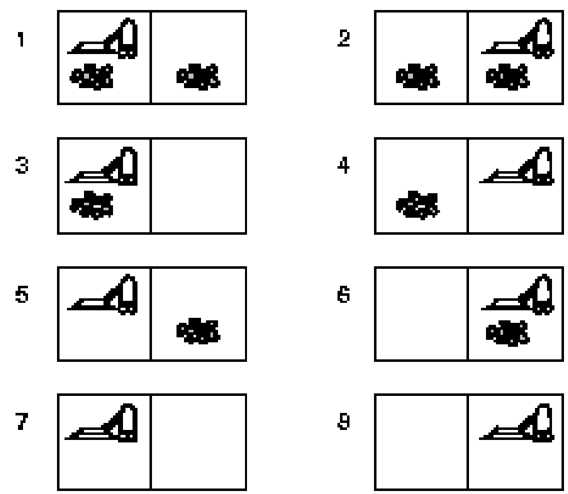
Atomic:

S1, S2…. S8

state is seen as an indivisible snapshot

All Actions are SXS matrices..

If you add a second roomba the state space *doubles*



Propositional/Factored:

States made up of 3 state variables

Dirt-in-left-room T/F

Dirt-in-right-room T/F

Roomba-in-room L/R

Each state is an assignment of Values to state variables

2^3 Different states

Actions can just mention the variables they affect

Note that the representation is compact (logarithmic in the size of the state space)

If you add a second roomba, the Representation increases by just one More state variable.

If you want to consider "noisiness" of rooms, we need *two* variables, one for Each room

Relational:

World made of objects: Roomba; L-room, R-room

Relations: In (<robot>, <room>); dirty(<room>)

If you add a second roomba, or more rooms, only the objects increase.

If you want to consider noisiness, you just need to add one other relation

Atomic Agent

Input:

- Set of states
- Operators [and costs]
- Start state
- Goal state [test]

Output:

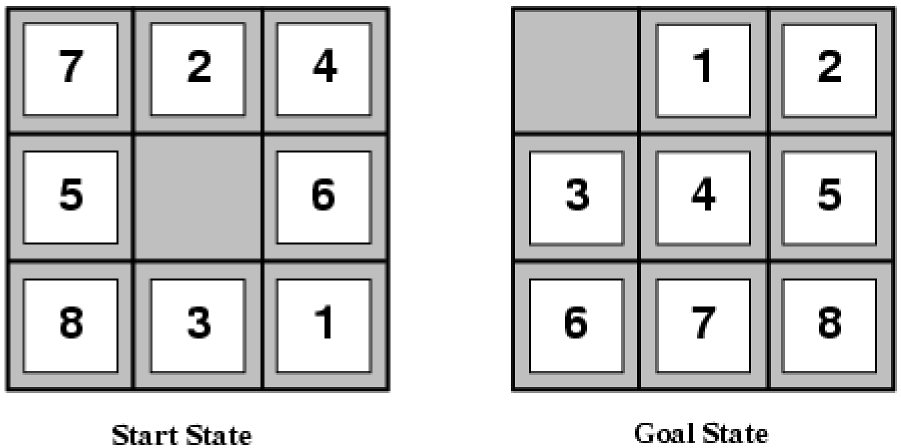
- Path: start $\Rightarrow$ a state satisfying goal test
- [May require shortest path]

Why is search interesting?

- Many (all?) AI problems can be formulated as search problems!

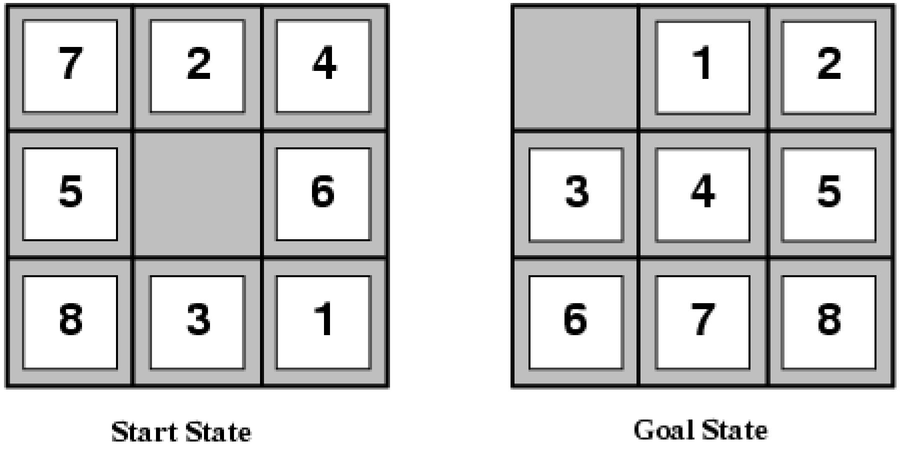
- Examples:
  - Path planning
  - Games
  - Natural Language Processing
  - Machine learning
  - …

Example: The 8-puzzle



- states?
- actions?
- goal test?
- path cost?

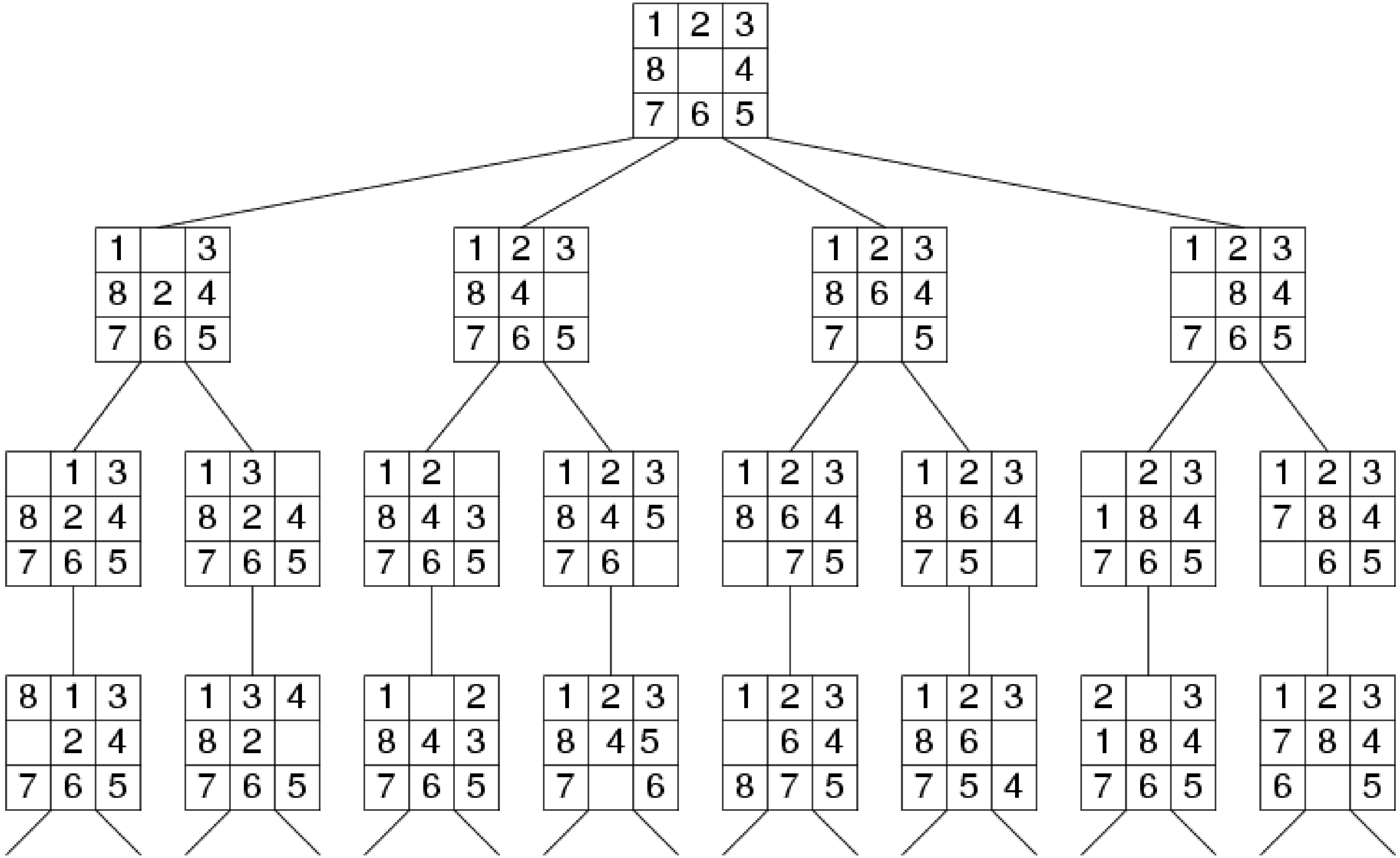
Example: The 8-puzzle



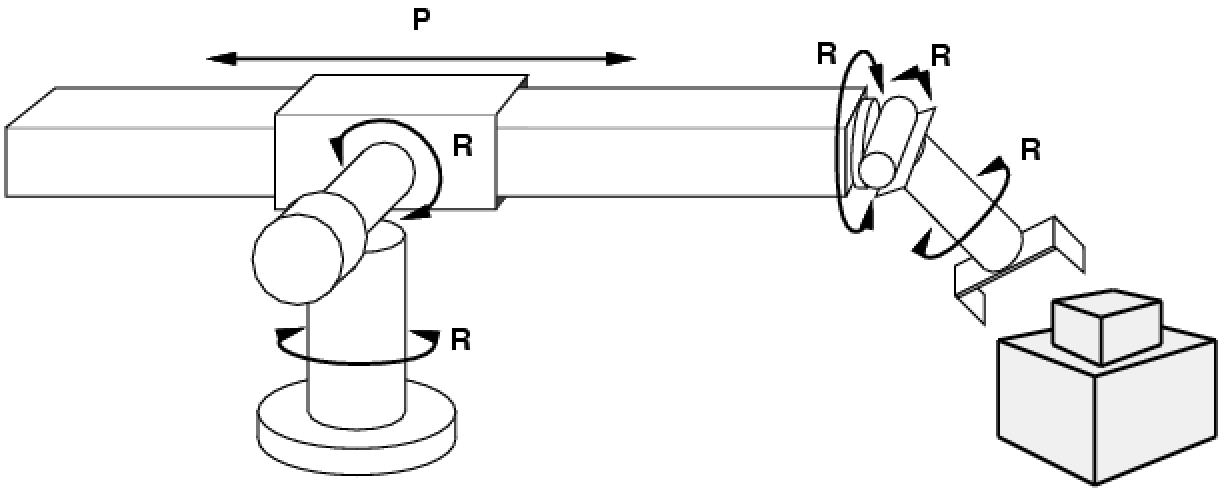
- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move
-

- [Note: optimal solution of *n*-Puzzle family is NP-hard]

Search Tree Example:
Fragment of 8-Puzzle Problem Space



Example: robotic assembly



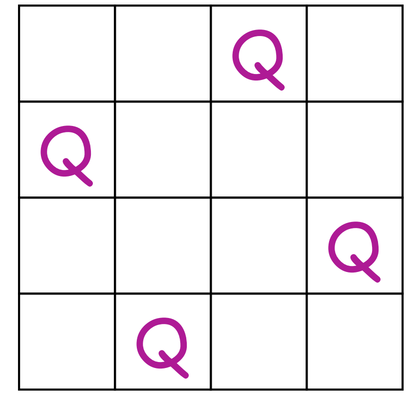
- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
-
- actions?: continuous motions of robot joints
-
- goal test?: complete assembly
-
- path cost?: time to execute
-

Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
-
- Formulate goal:
  - be in Bucharest
  -
- Formulate problem:
  - states: various cities
  - actions: drive between cities
  -
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
  -

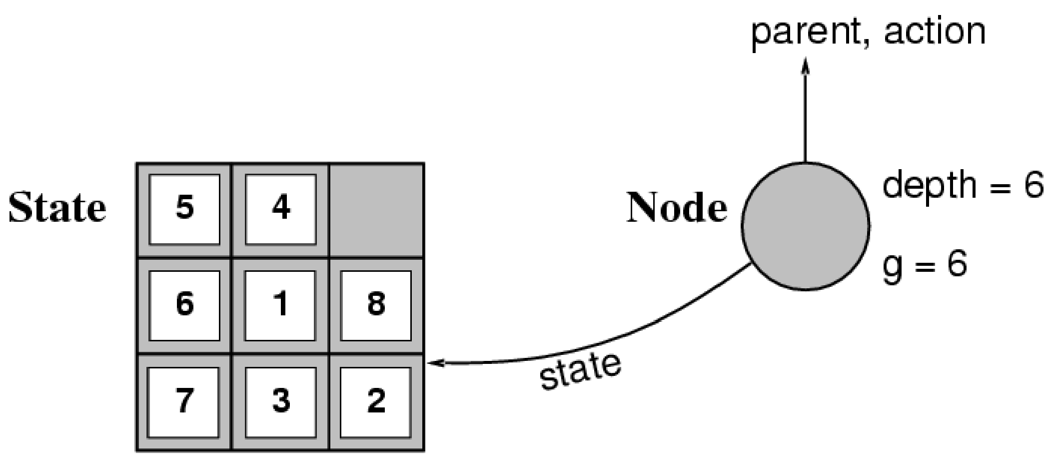
Example: N Queens

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output



Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.
-

Search strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
  - systematicity: does it visit each state at most once?

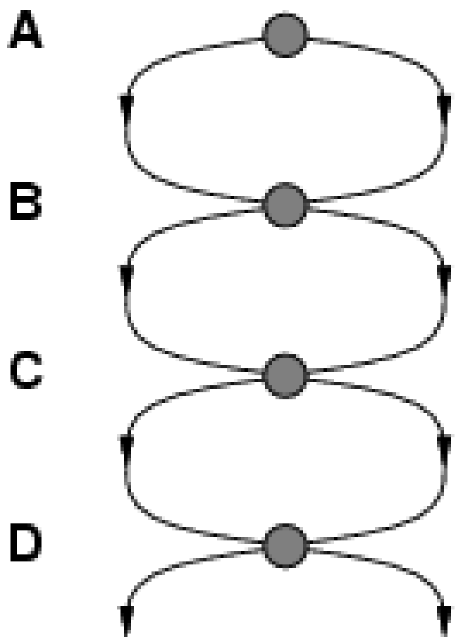
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the shallowest solution
  - *m*: maximum depth of the state space (may be ∞)

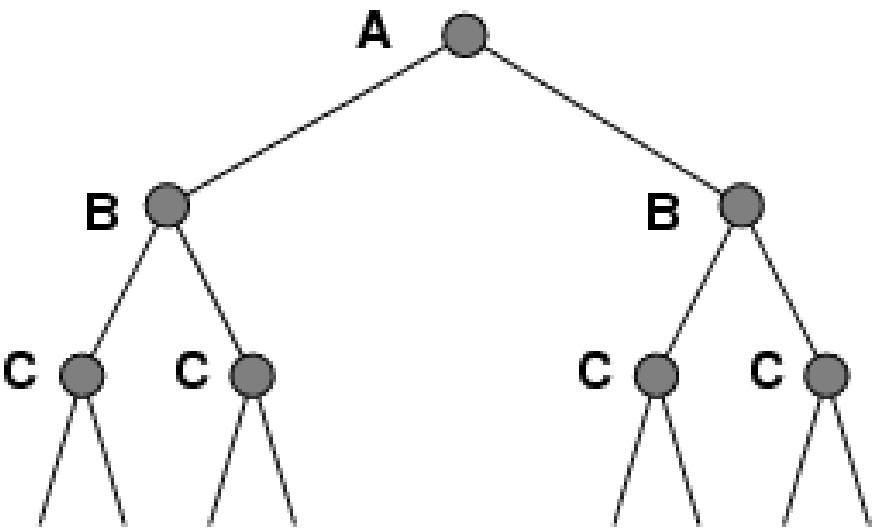
Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Repeated states

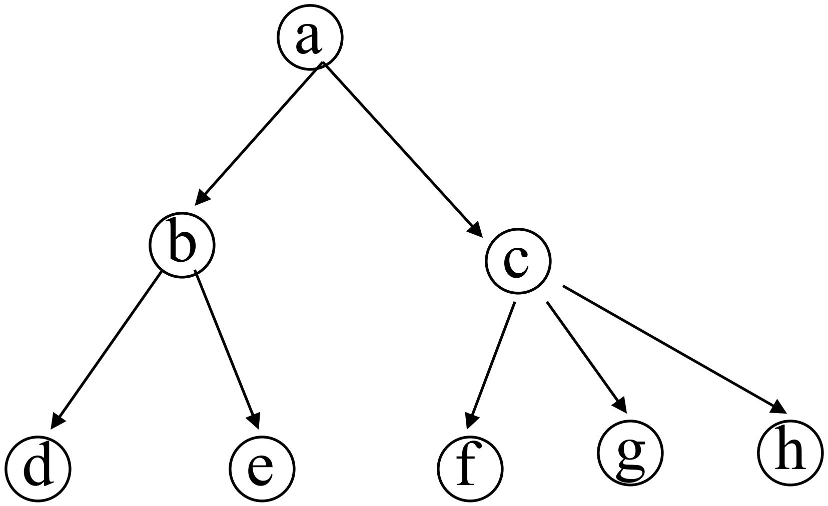
- Failure to detect repeated states can turn a linear problem into an exponential one!





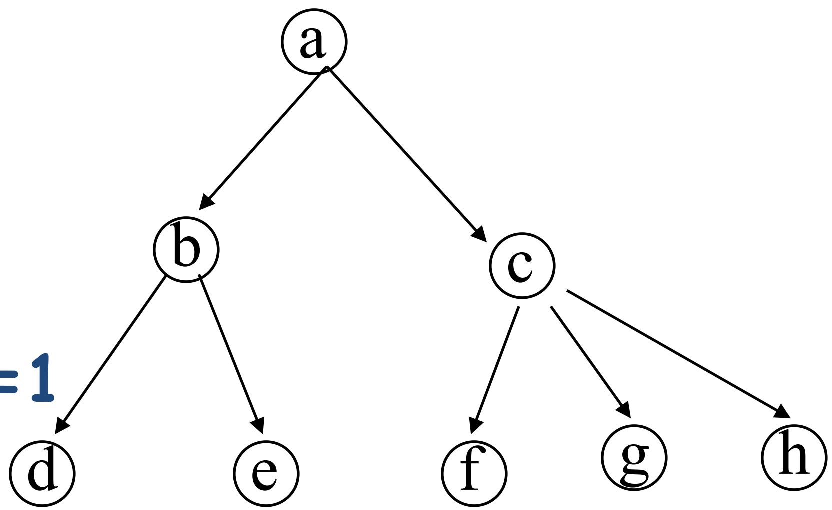
Depth First Search

- Maintain stack of nodes to visit
- Evaluation
  - Complete? No
  - Time Complexity? $O(b^m)$
  - Space Complexity? $O(bm)$



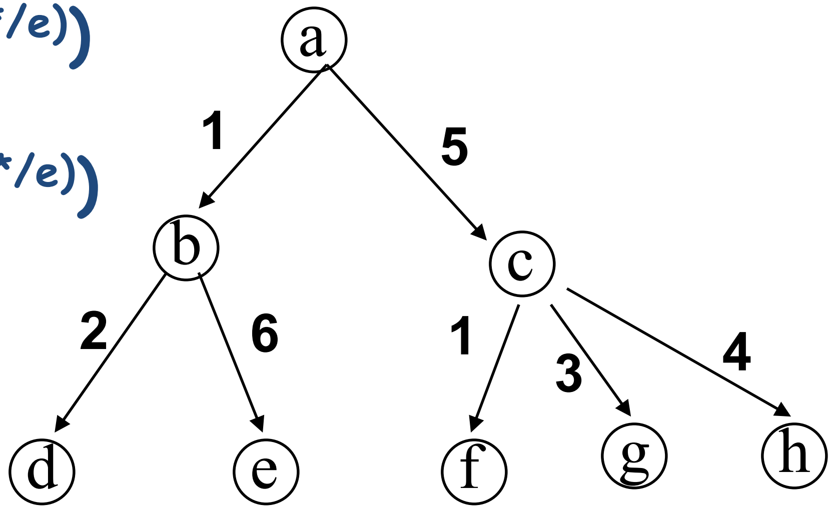
Breadth First Search: shortest first

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^d)$
  - Space Complexity? $O(b^d)$
  - Optimal? Yes, if stepcost=1



Uniform Cost Search: cheapest first

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^{(C^*/e)})$
  - Space Complexity? $O(b^{(C^*/e)})$
  - Optimal? Yes



DFS

http://www.youtube.com/watch?v=dtoFAvtVE4U

UCS

http://www.youtube.com/watch?v=z6IUnb9ktkE

Memory Limitation

- Suppose:

  2 GHz CPU

  1 GB main memory

  100 instructions / expansion

  5 bytes / node

  200,000 expansions / sec

  Memory filled in 100 sec … < 2 minutes

Time vs. Memory

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

Idea 1: Beam Search

- Maintain a constant sized frontier
- Whenever the frontier becomes large
  - Prune the worst nodes

Optimal: no

Complete: no

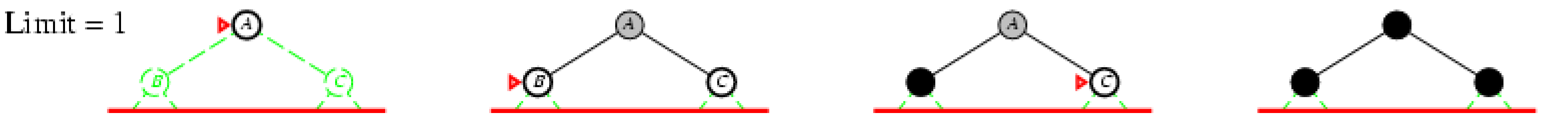
Idea 2: Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

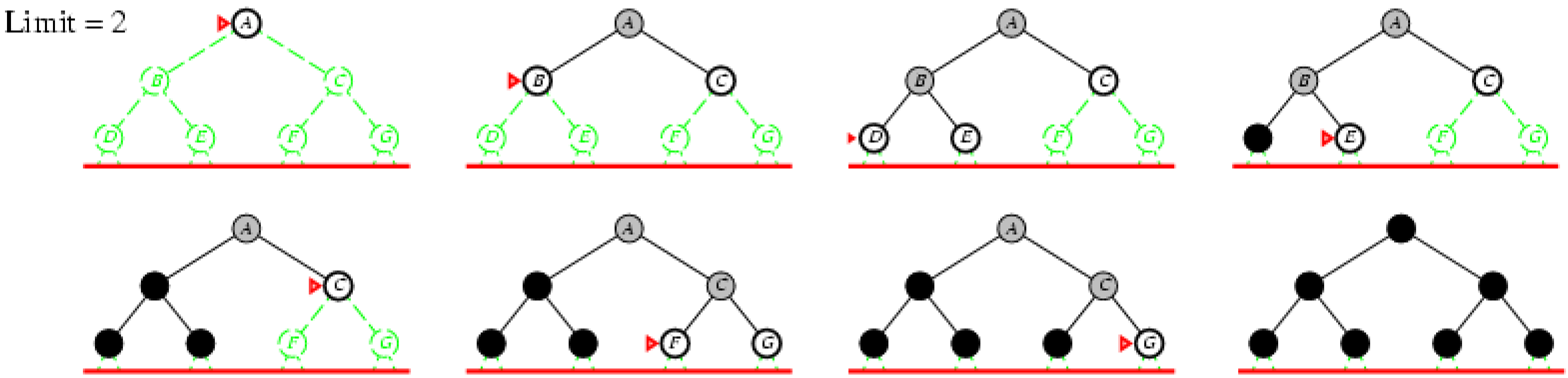
Iterative deepening search l =0

Limit = 0

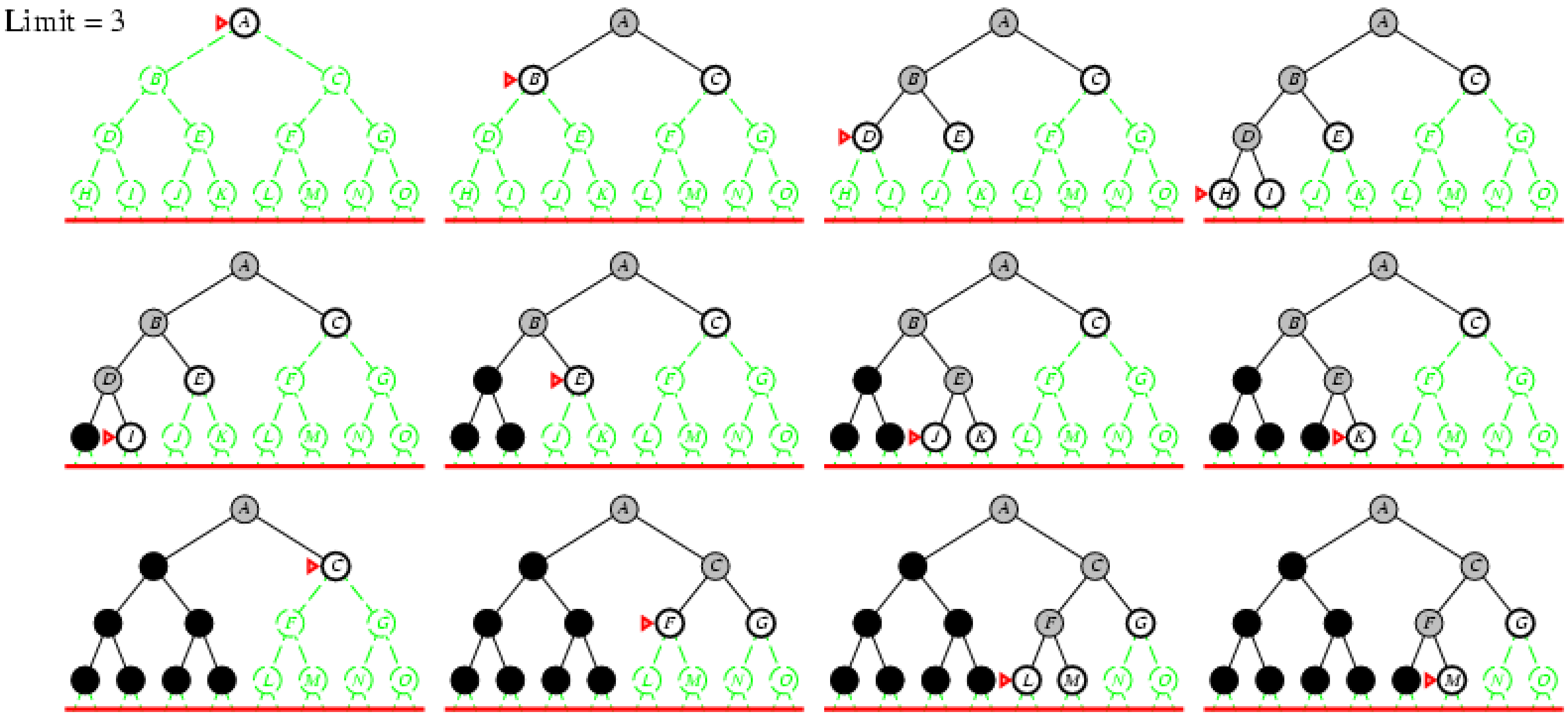
Iterative deepening search *l* =1



Iterative deepening search *l* =2



Iterative deepening search *l* =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:
  - $N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:
  - $N_{IDS} = (d+1)b^0 + d\ b\hat{}1 + (d-1)b\hat{}2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$

- Asymptotic ratio: (b+1)/(b-1)

- For *b = 10*, *d = 5*,
-
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  -
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$
  -

- Overhead = (123,456 - 111,111)/111,111 = 11%

Iterative deepening search

- Complete?
  - Yes

- Time?
  - $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

- Space?
  - $O(bd)$

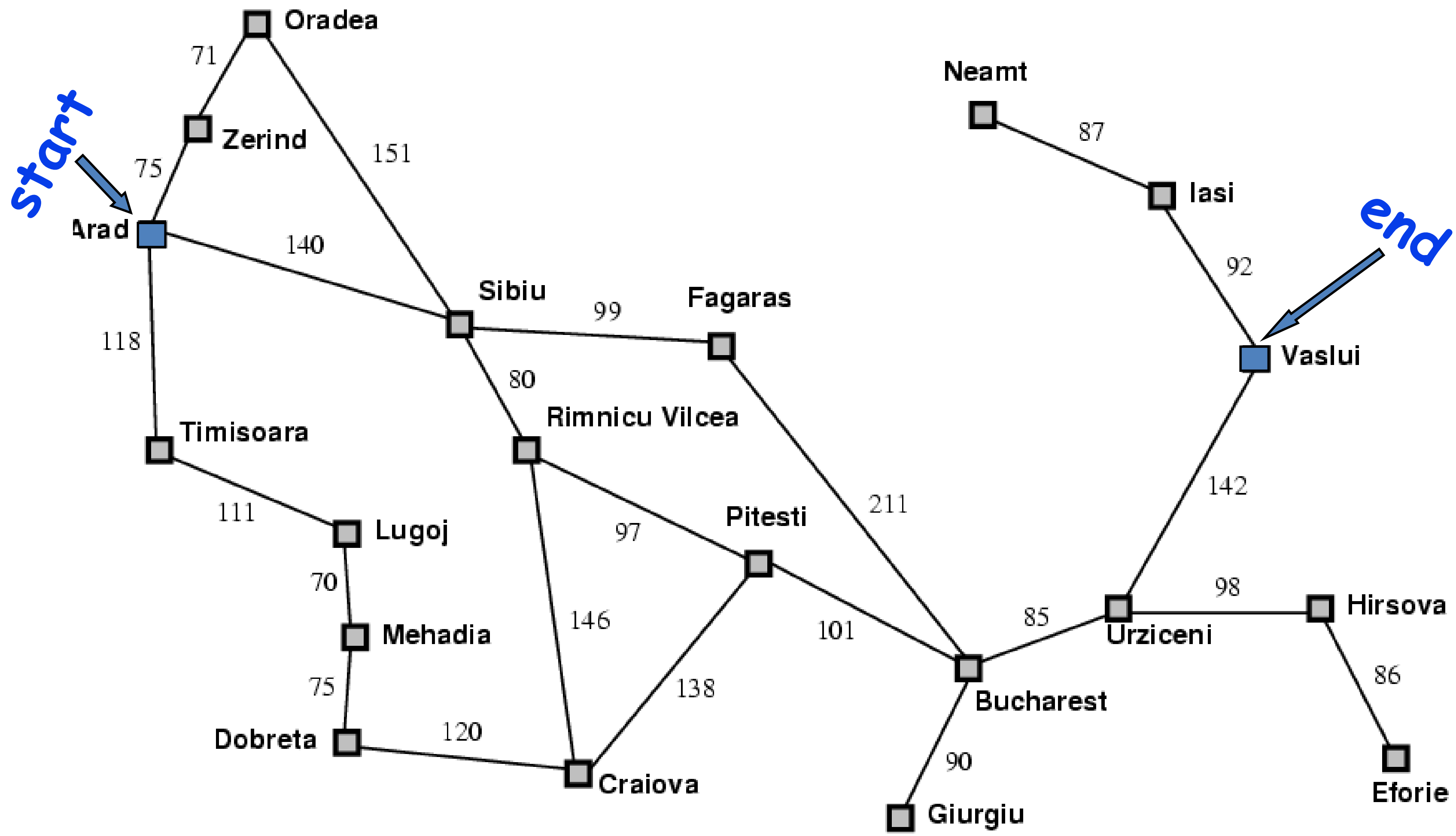
- Optimal?
  - Yes, if step cost = 1
  - Can be modified to explore uniform cost tree (iterative lengthening)

- Systematic?

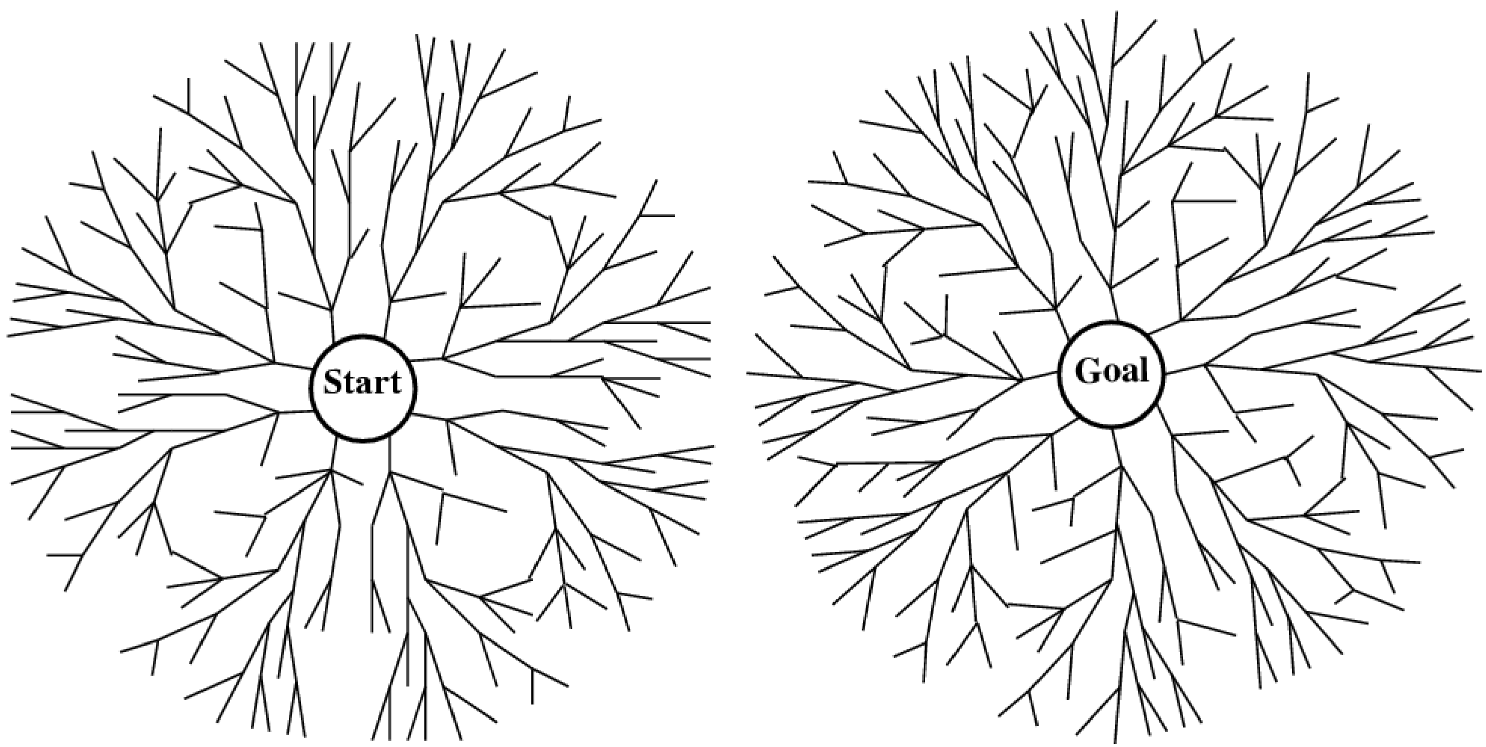
Cost of Iterative Deepening

b	ratio ID to DLS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Forwards *vs.* Backwards



vs. Bidirectional



When is bidirectional search applicable?

- **Generating predecessors is easy**
- **Only 1 (or few) goal states**

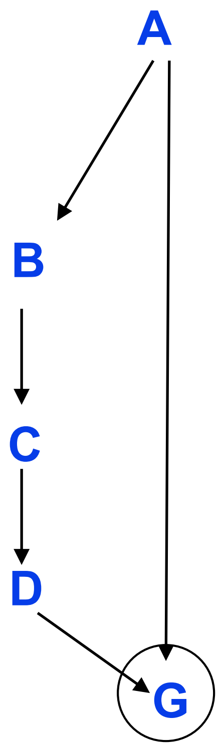
Bidirectional search

- Complete? Yes
- Time?
  – $O(b^{d/2})$
- Space?
  – $O(b^{d/2})$
- Optimal?
  – Yes if uniform cost search used in both directions

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

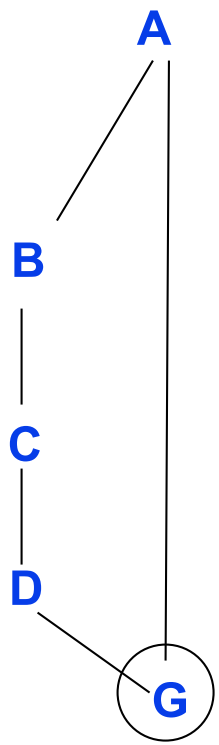


BFS: A,B,G

DFS: A,B,C,D,G

IDDFS: (A), (A, B, G)

Note that IDDFS can do fewer expansions than DFS on a graph shaped search space.



BFS: A,B,G

DFS: A,B,A,B,A,B,A,B,A,B

IDDFS: (A), (A, B, G)

Note that IDDFS can do fewer expansions than DFS on a graph shaped search space.

Search on undirected graphs or directed graphs with cycles…

Cycles galore…

Graph (instead of tree) Search: Handling repeated nodes

- Repeated expansions is a bigger issue for DFS than for BFS or IDDFS
  - Trying to remember all previously expanded nodes and comparing the new nodes with them is infeasible
  - Space becomes exponential
  - duplicate checking can also be expensive

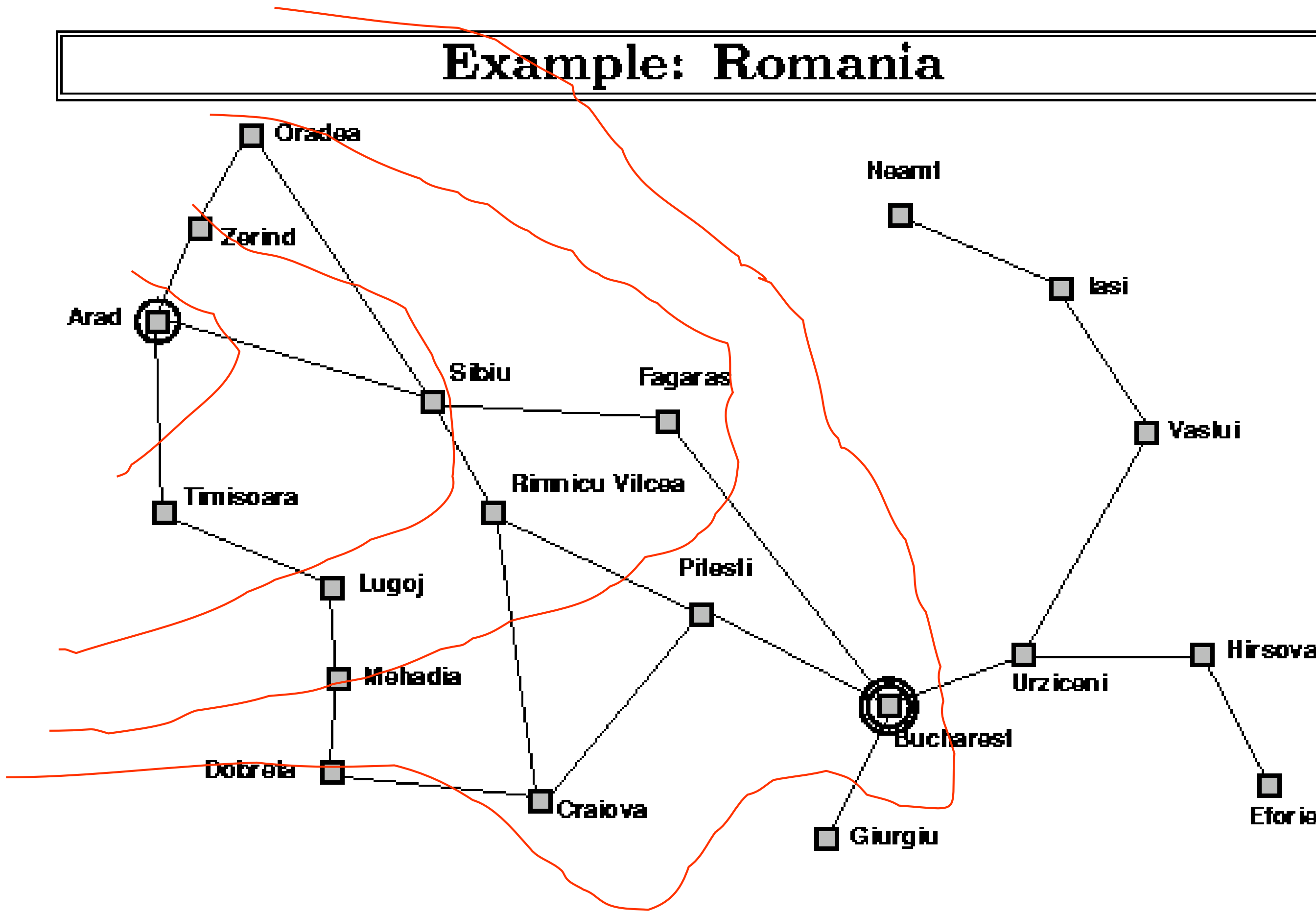
- Partial reduction in repeated expansion can be done by
  - Checking to see if any children of a node n have the same state as the parent of n
  - Checking to see if any children of a node n have the same state as any ancestor of n (at most d ancestors for n—where d is the depth of n)

chapter03 - GSview

File Edit Options View Orientation Media Help

File: chapter03 206, 460pt Page: "5" 5 of 63

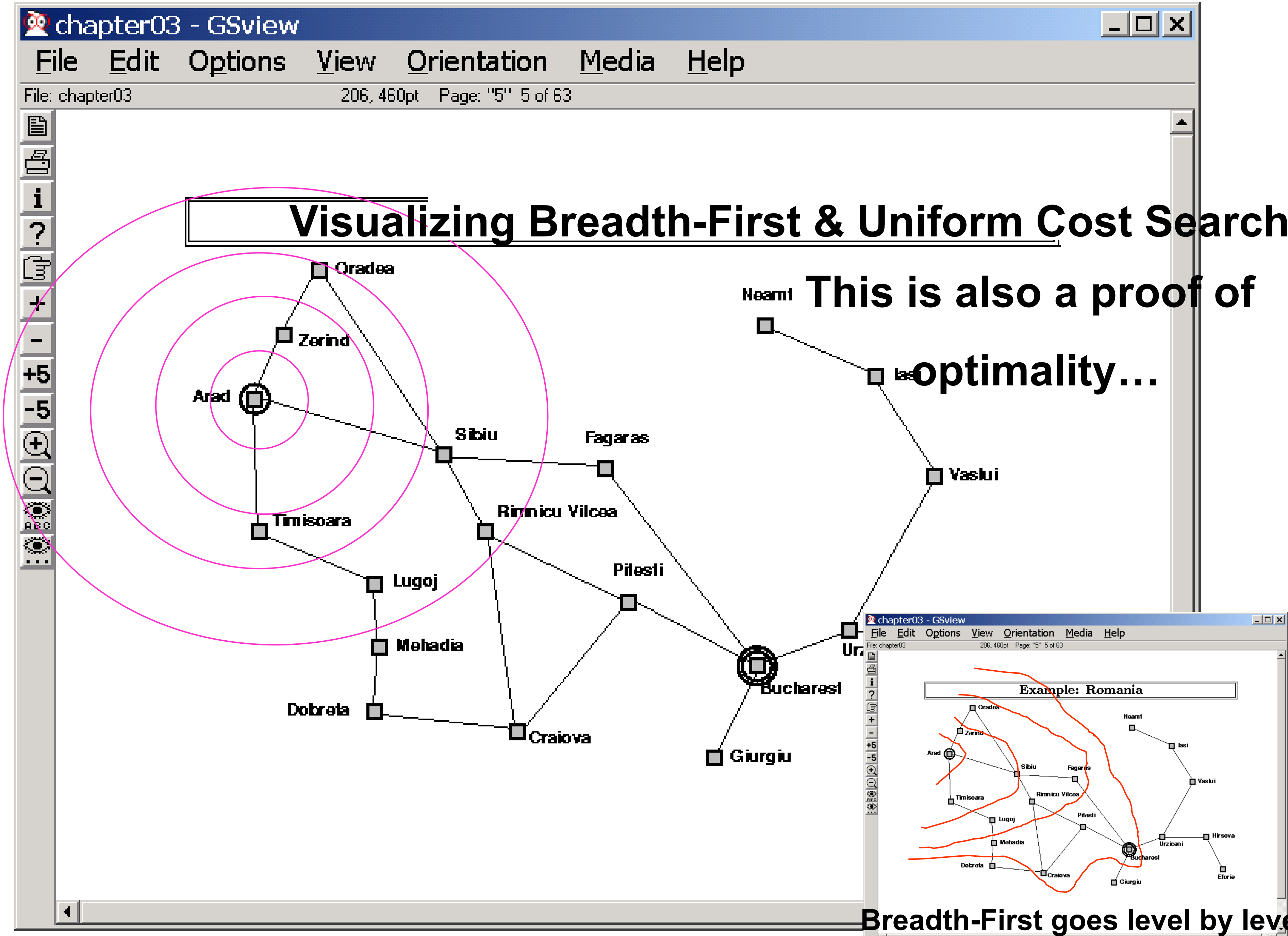
Example: Romania



Breadth-First goes level by level

Visualizing Breadth-First & Uniform Cost Search

This is also a proof of optimality…



Breadth-First goes level by leve

Problem

- All these methods are slow   (blind)

- Solution   → add guidance ("heuristic estimate")

  → "informed search"