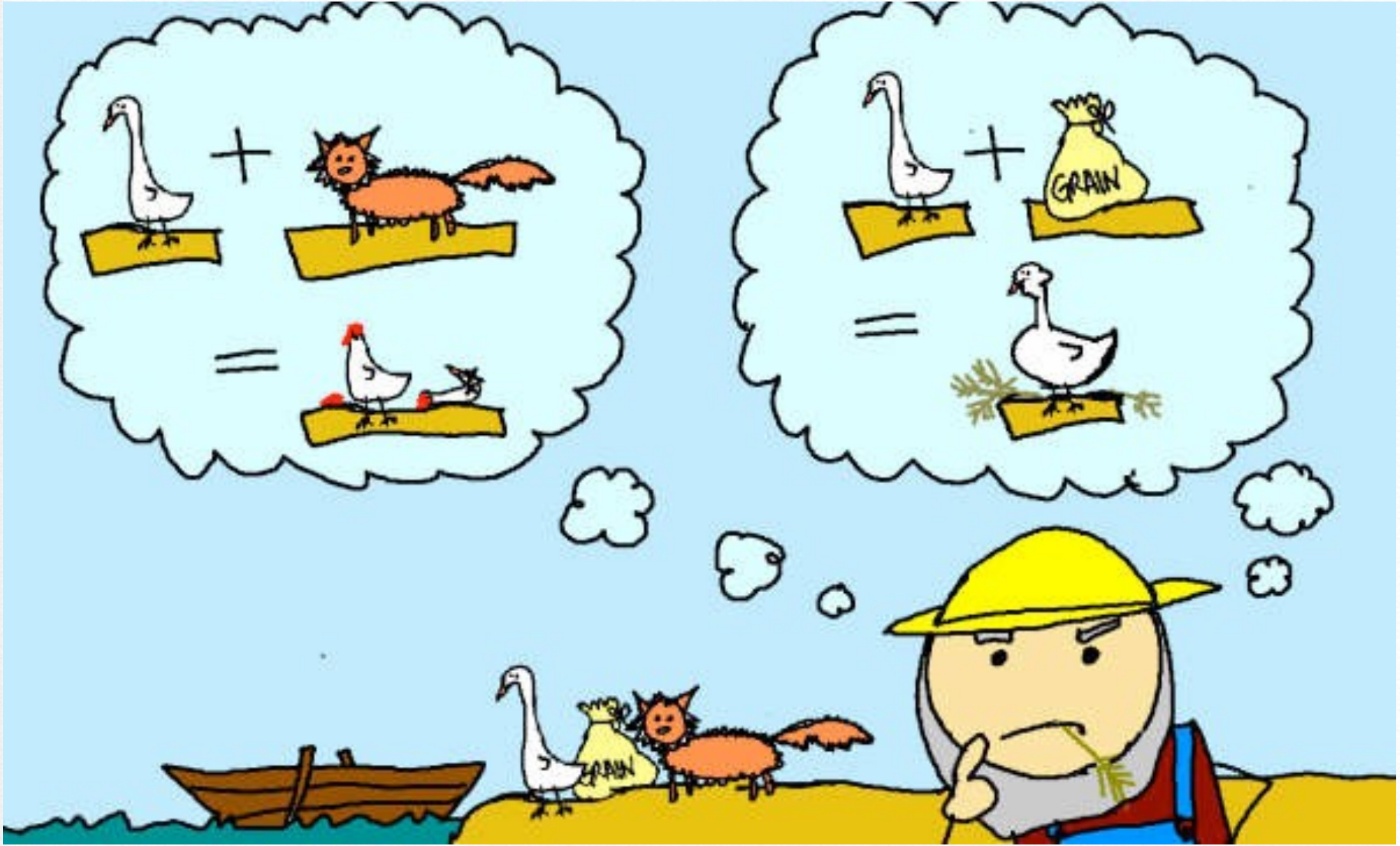
STATE SPACES & BLIND SEARCH

M. Anthony Kapolka III
Wilkes University
CS 340 - Fall 2019

Issues in Search

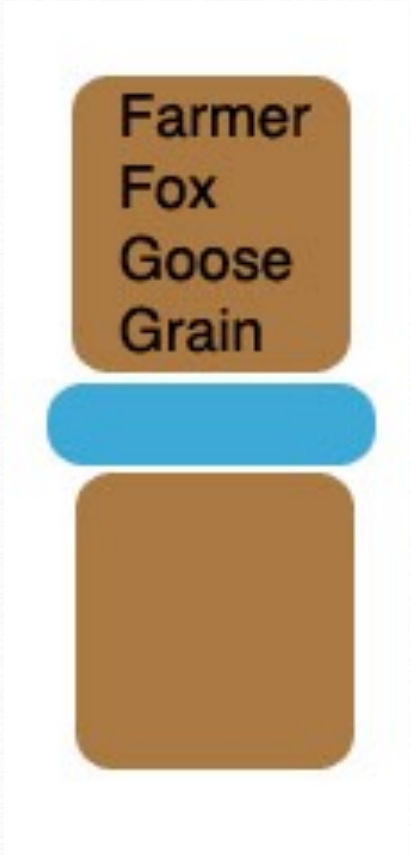
- Knowledge Representation
- Topology (of State Space)
- Algorithms
- Forward or Backward


GRAIN
GRAIN

The Farmer, Fox, Goose and Grain...

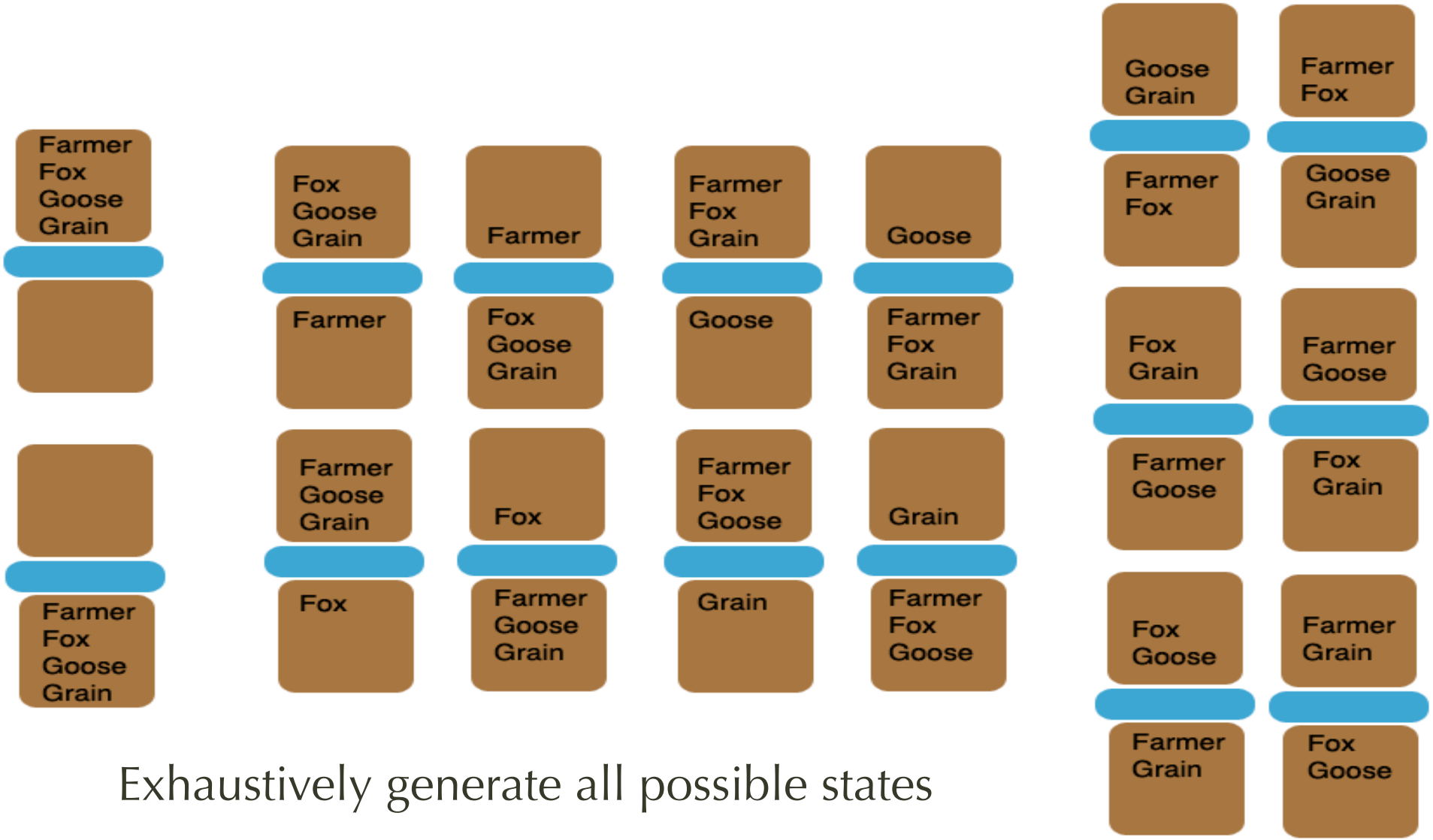
A farmer wants to move himself, a silver fox, a fat goose, and some tasty grain across a river from the west side to the east side. Unfortunately, his boat is so small he can only take one of his possessions across on any trip. Worse yet, an unattended fox will eat a goose, and an unattended goose will eat grain, so the farmer must not leave the fox and goose alone, or the goose and grain alone.

What should he do to get everything across?

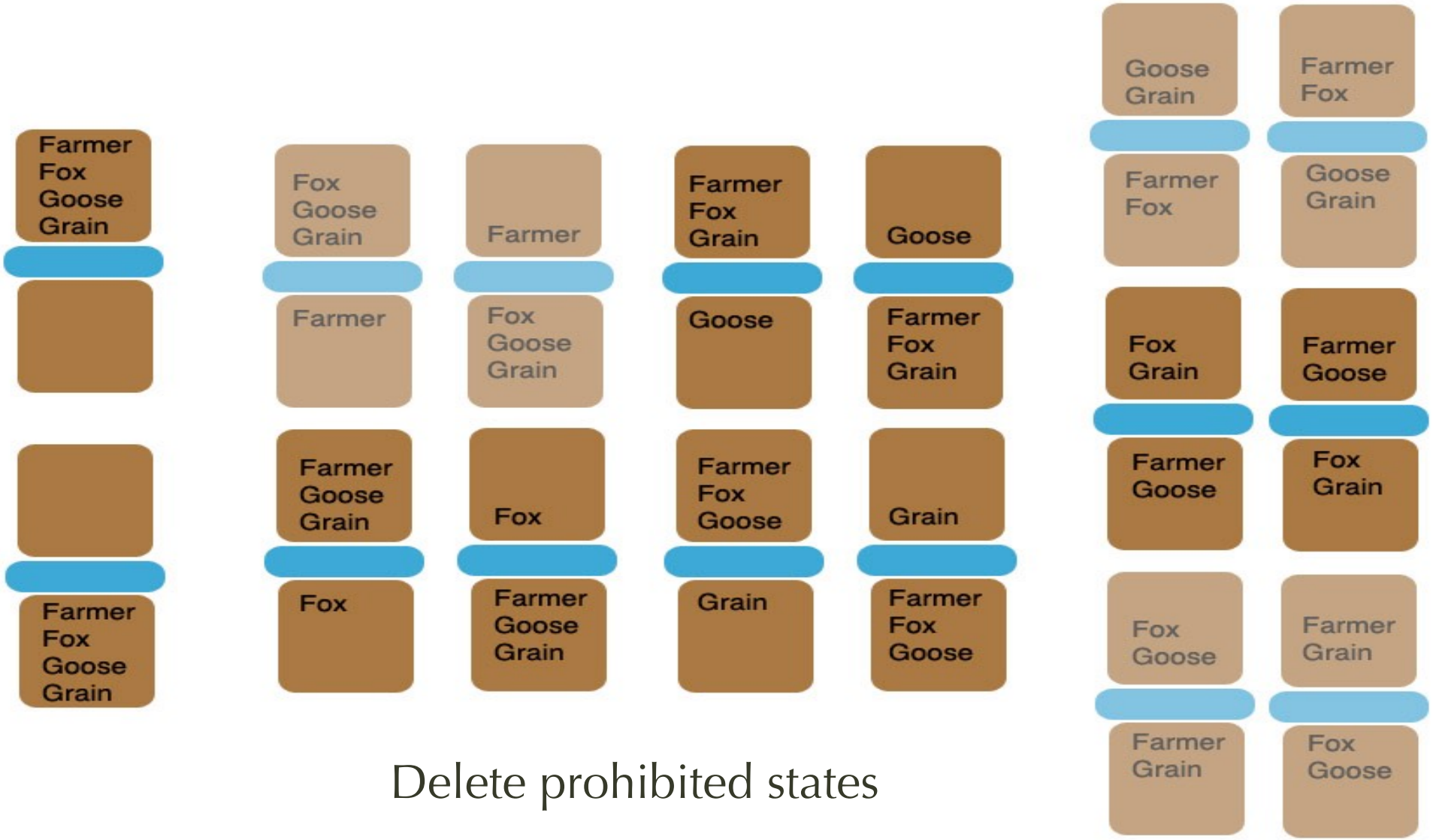


Start State

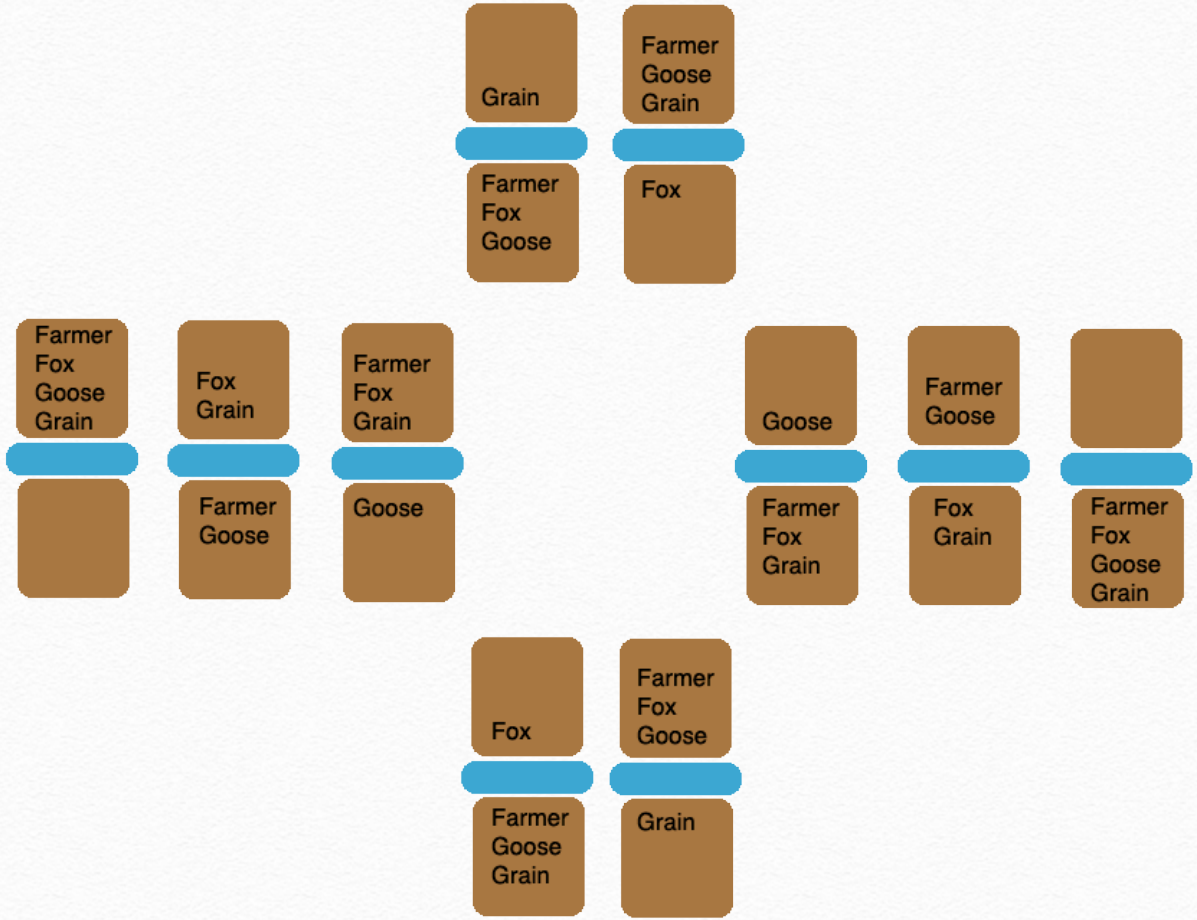
Goal State



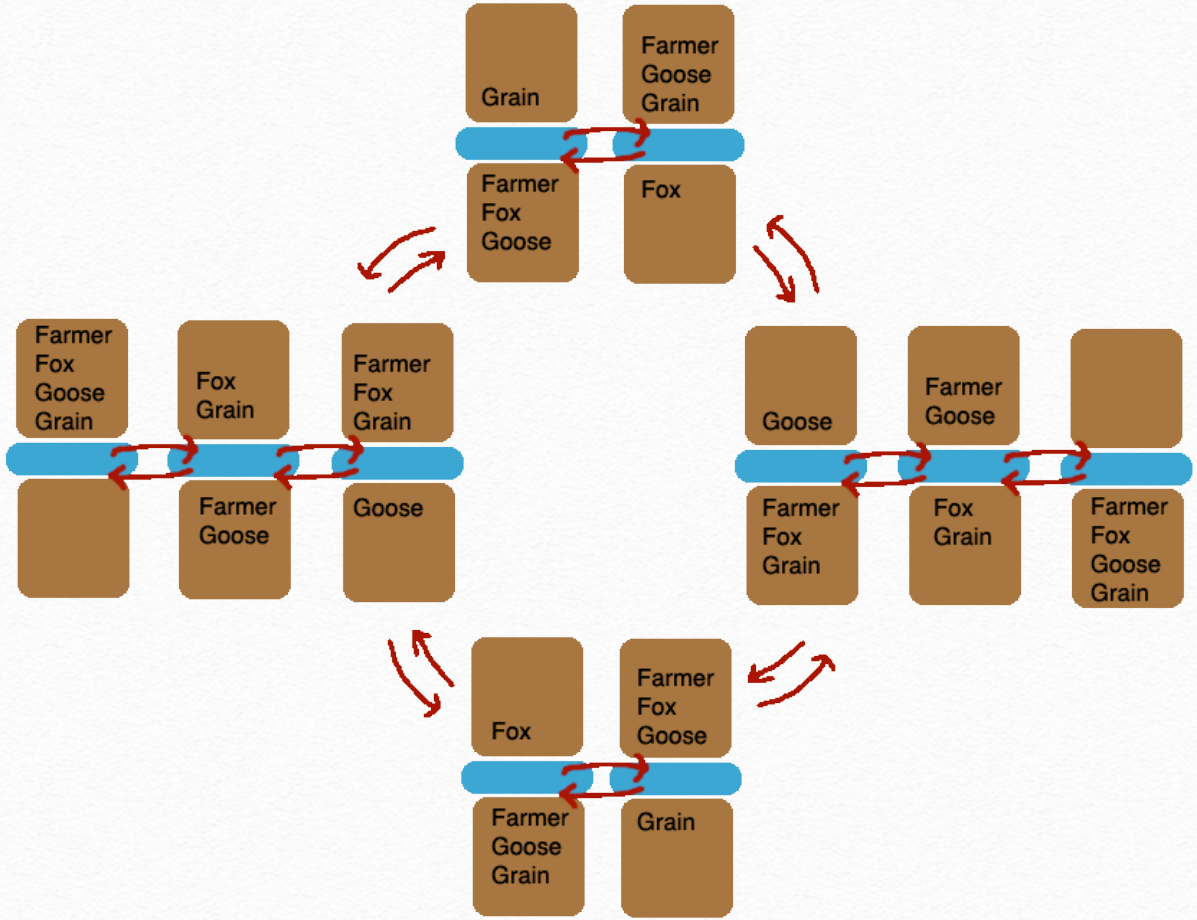
Exhaustively generate all possible states



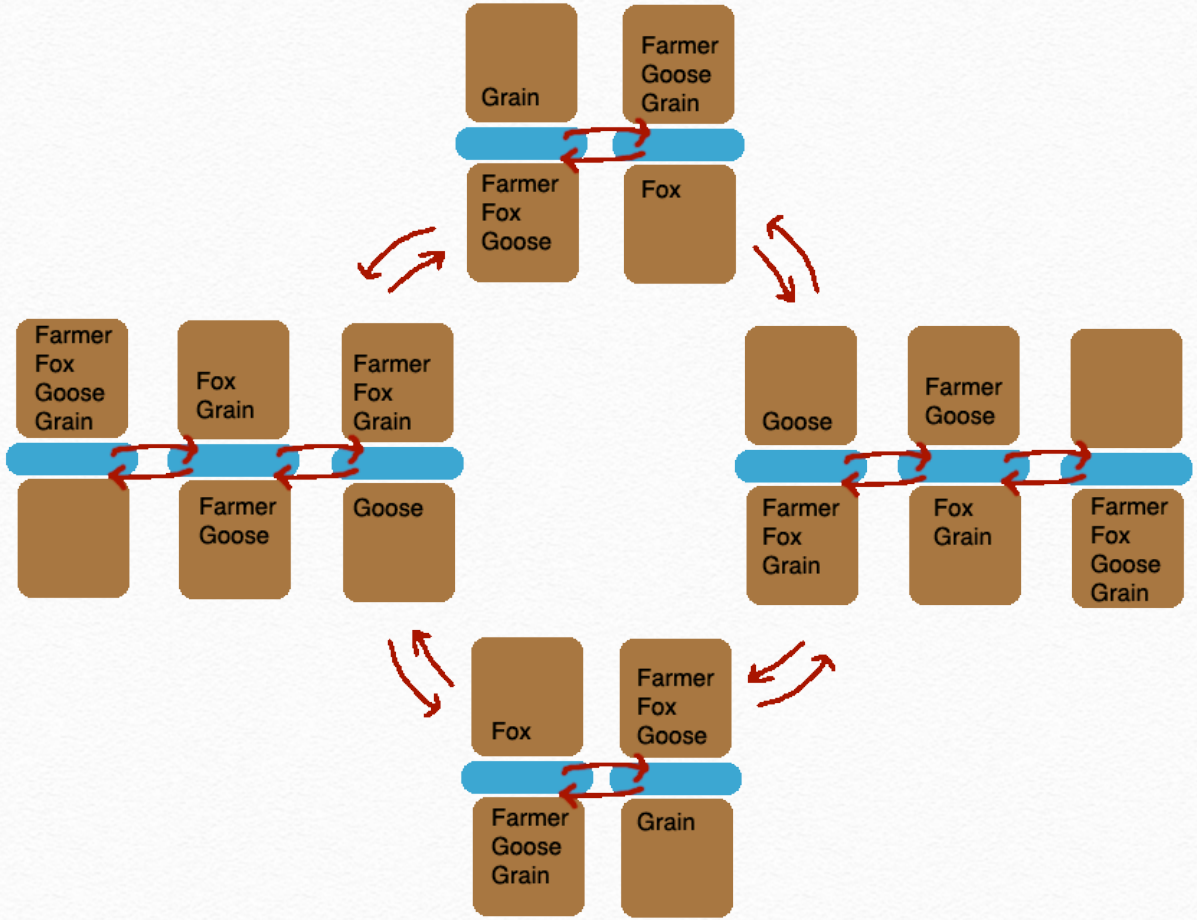
Delete prohibited states



Reorganize states



Solutions are obvious.



What is the topology?

Problem Space Topology

Problem Space (Newell & Simon, 1972)

Research in human cognition

We search for a solution
among a set of possible solutions

A Brief Discourse on Context and History


miss-
ion-
ary.
i
eat
people
i
eat
people
2 small

Missionaries and Cannibals

Three missionaries and three cannibals come to a river and find a boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten.

How shall they cross?

Problem Space Topology

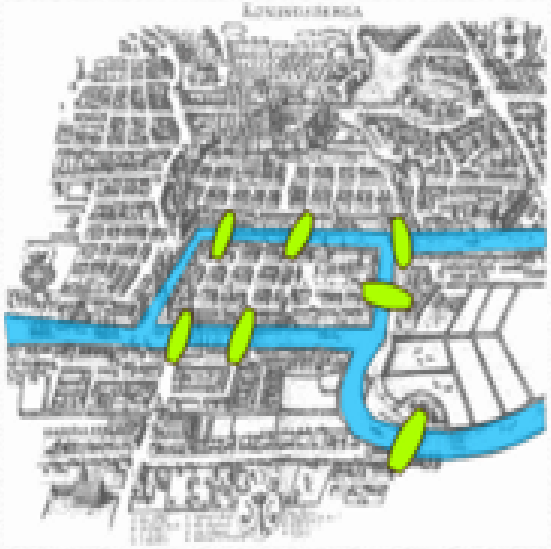
- Topology
  - Organization of problem space
  - Standard data structures, including graphs and trees

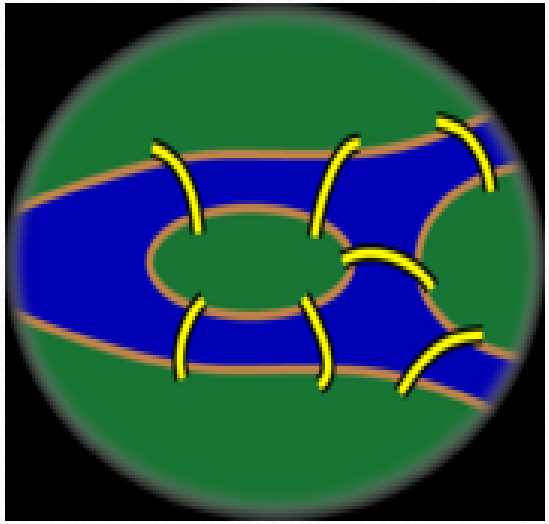
Seven Bridges of Königsberg

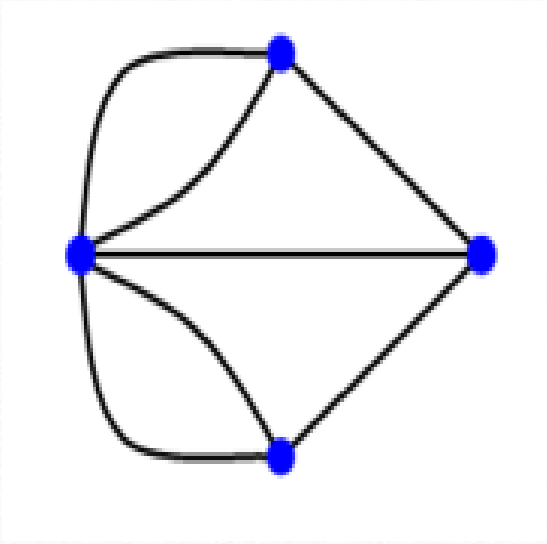
Task: Walk around, crossing over all bridges once and only once.

Seven Bridges of Königsberg

solved by Leonhard Euler, 1736.



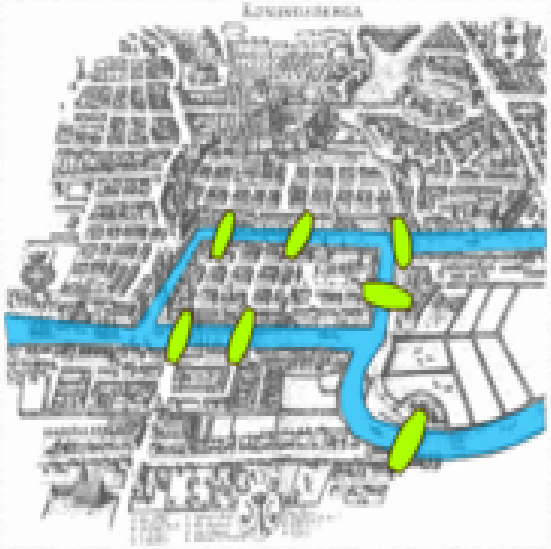


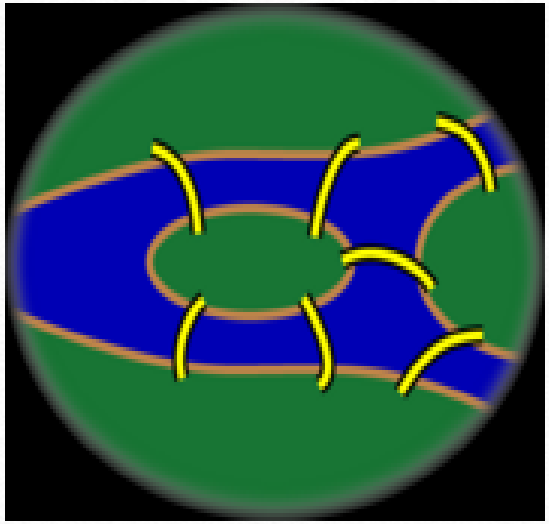


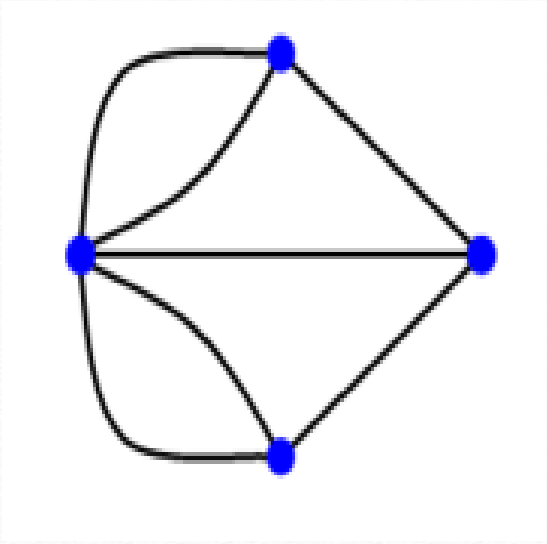
Illustrations from Wikipedia, the free encyclopedia.

Seven Bridges of Königsberg

solved by Leonhard Euler, 1736.





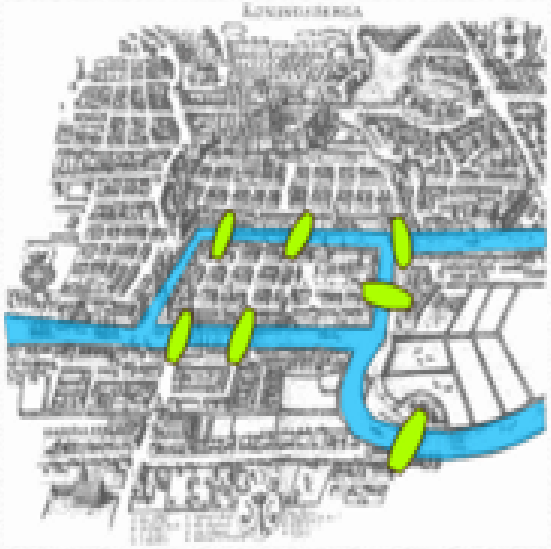


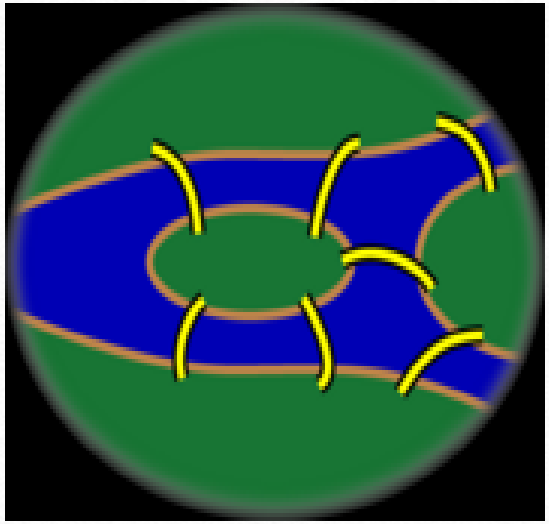
Eulerian circuit possible?

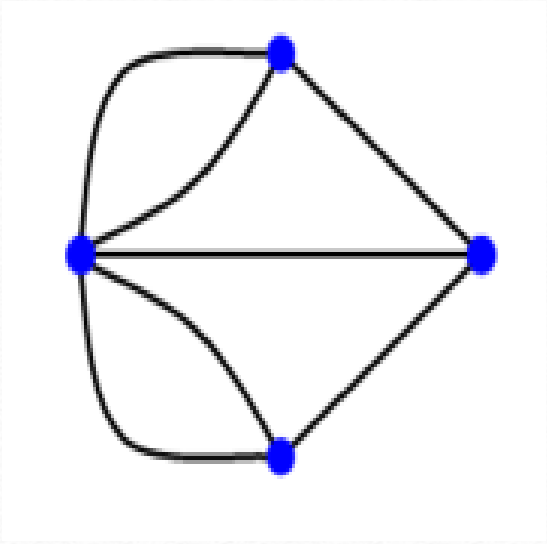
Illustrations from Wikipedia, the free encyclopedia.

Seven Bridges of Königsberg

solved by Leonhard Euler, 1736.





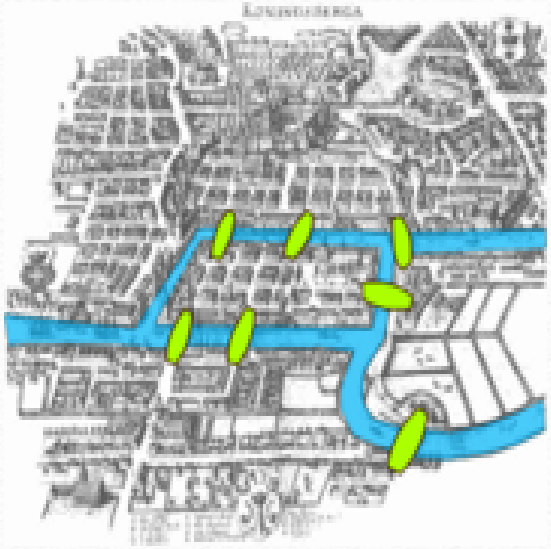


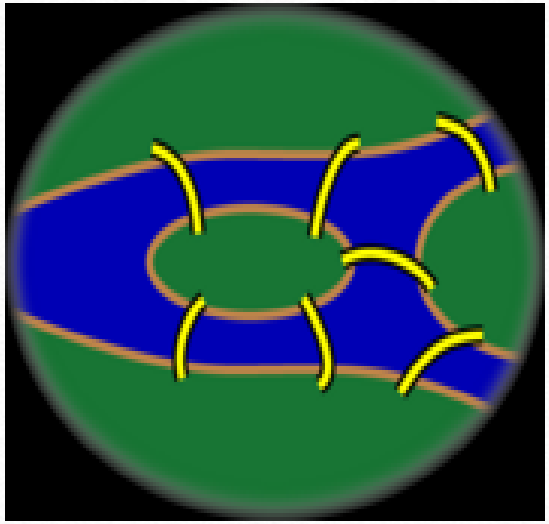
Eulerian circuit possible? Not if any nodes have odd degree!

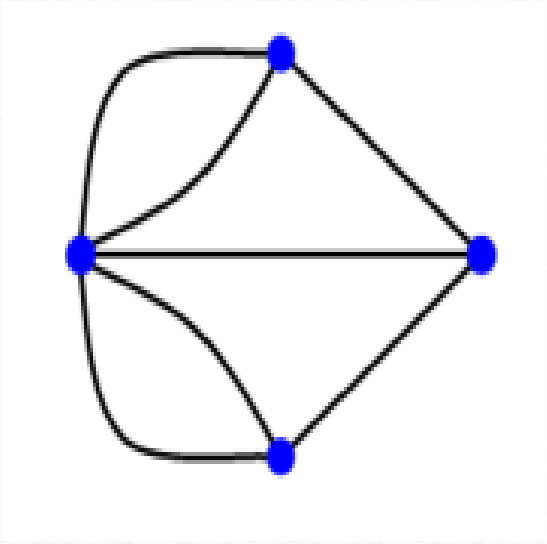
Illustrations from Wikipedia, the free encyclopedia.

Seven Bridges of Königsberg

solved by Leonhard Euler, 1736.





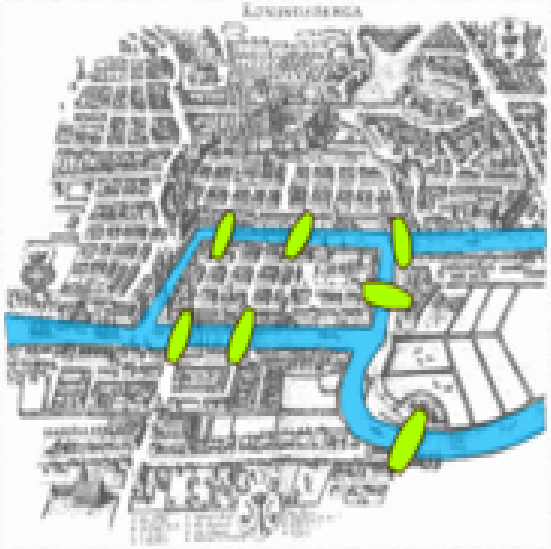


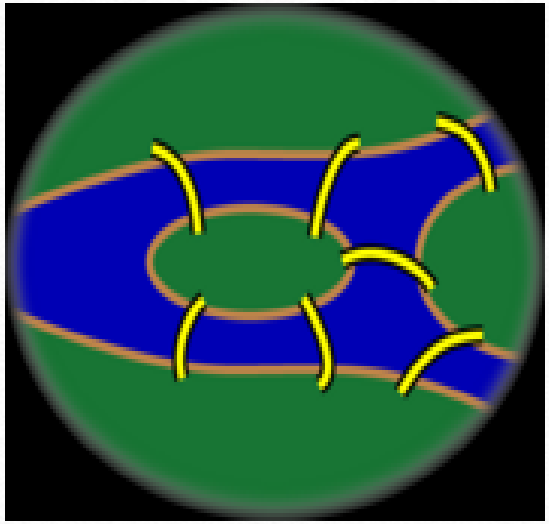
Eulerian walk possible?

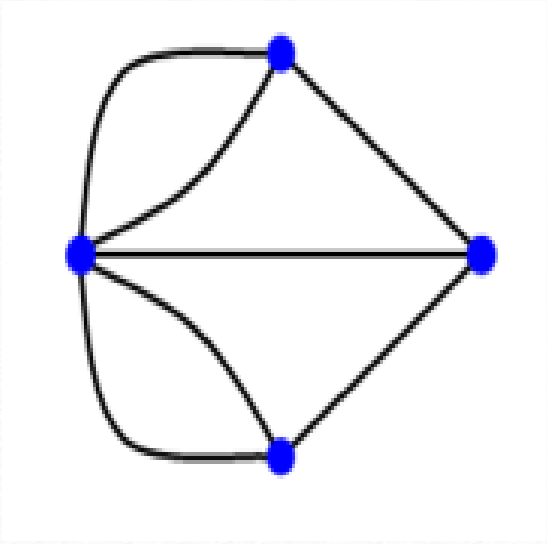
Illustrations from Wikipedia, the free encyclopedia.

Seven Bridges of Königsberg

solved by Leonhard Euler, 1736.







Eulerian walk possible?

Only if 0 or 2 nodes have odd degree!

Illustrations from Wikipedia, the free encyclopedia.

Trees vs. Graphs

Main difference: trees are directed, acyclic

Lots of tree traversal & graph algorithms

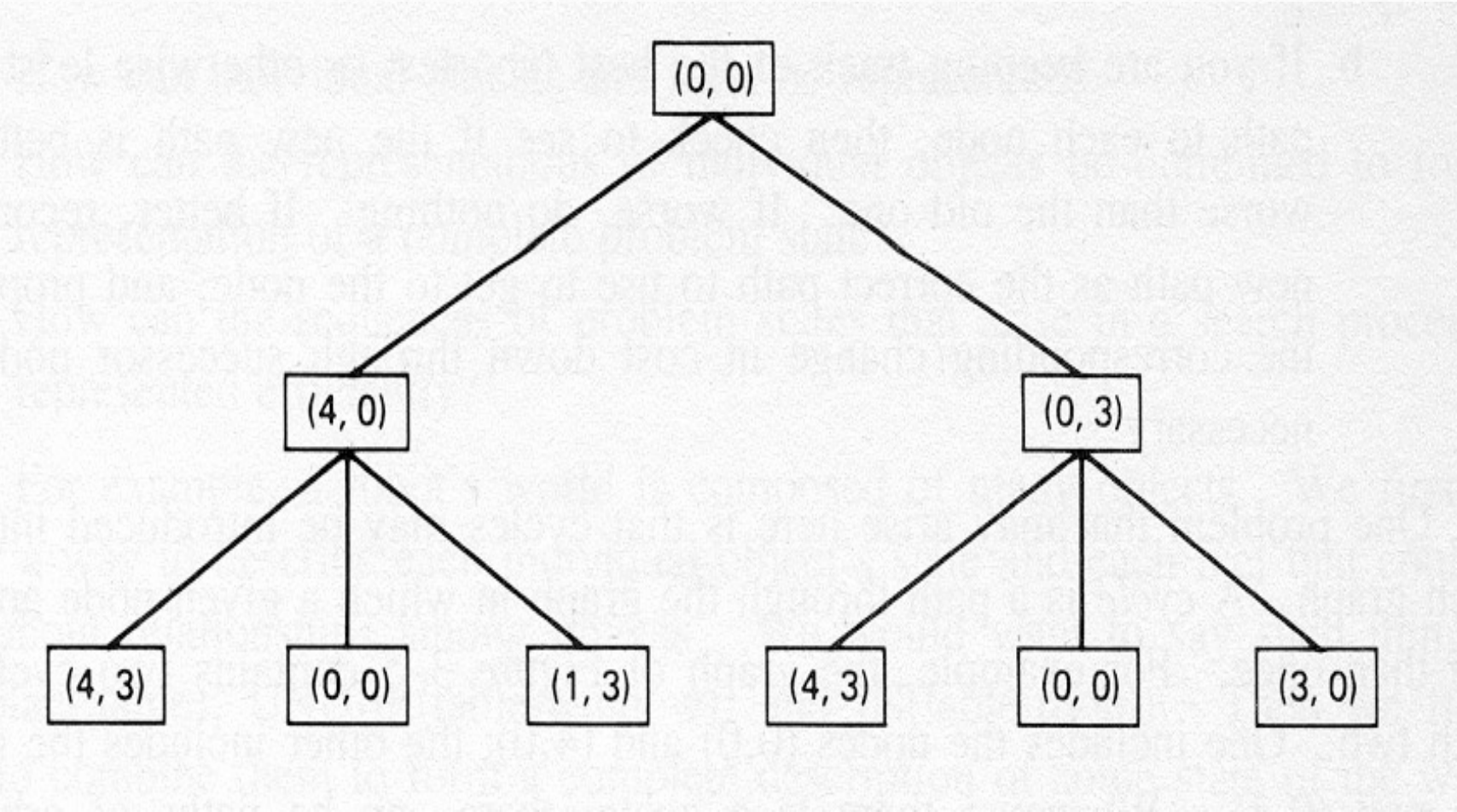
Nodes in trees may be repeatedly generated

Can convert tree traversals into graph search

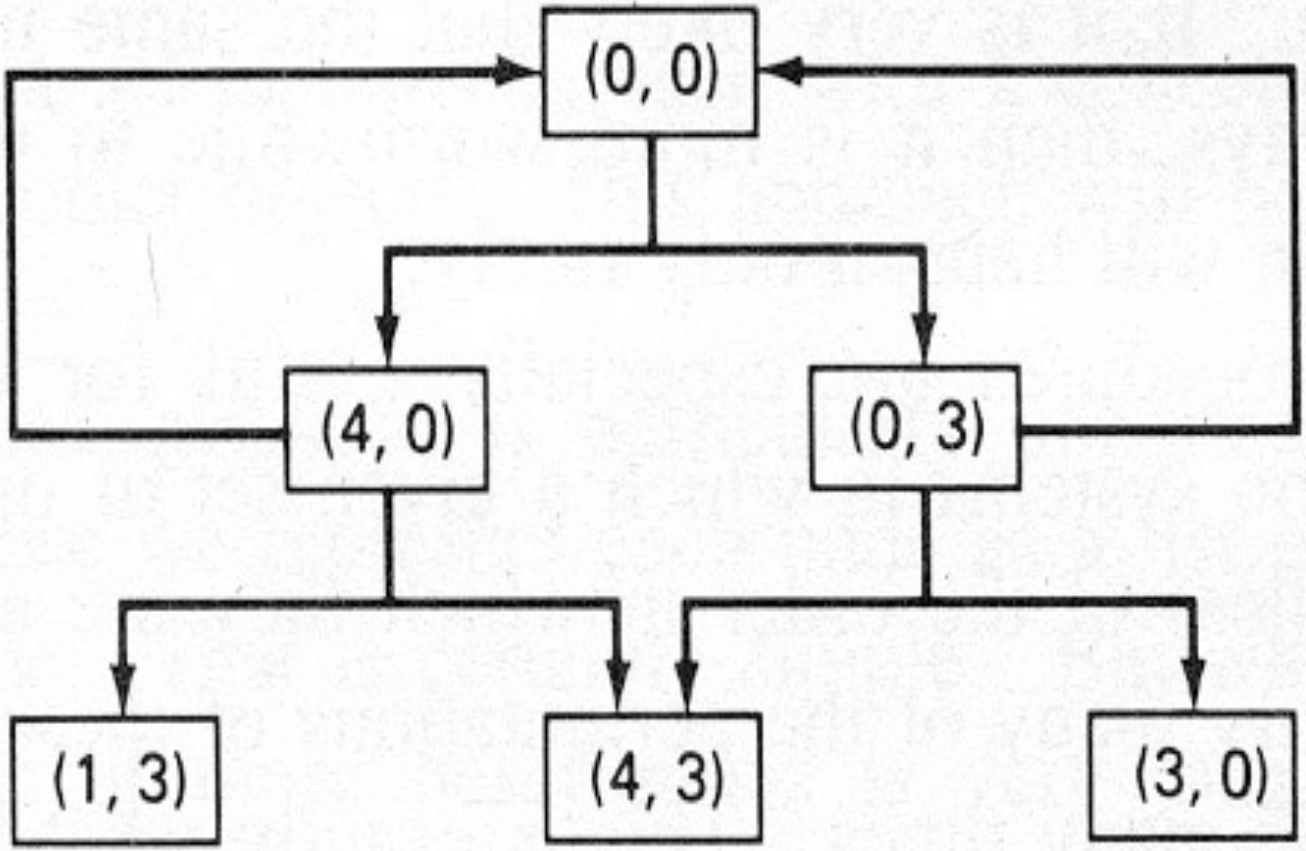
Water Jug Example

You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measurement markings on it. There is a tap to fill them with water.

How can you get exactly two gallons of water into the four gallon jug?

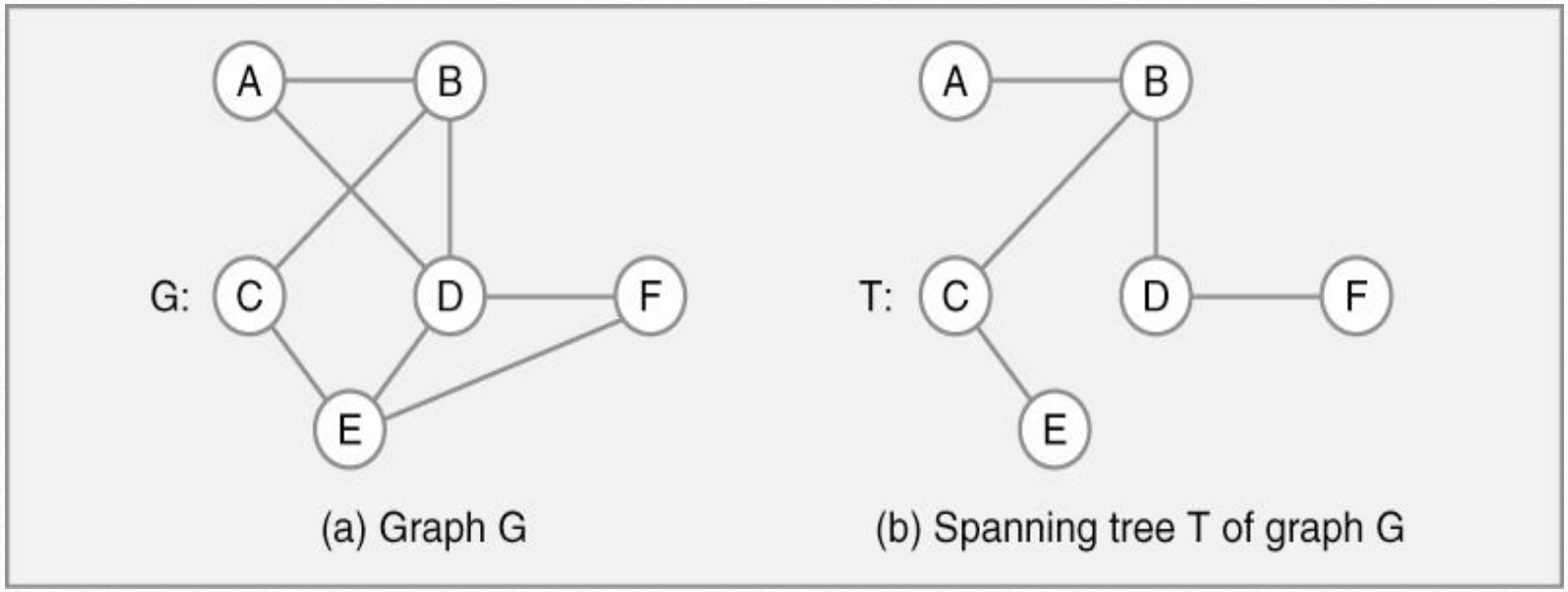


[Rich 1983]



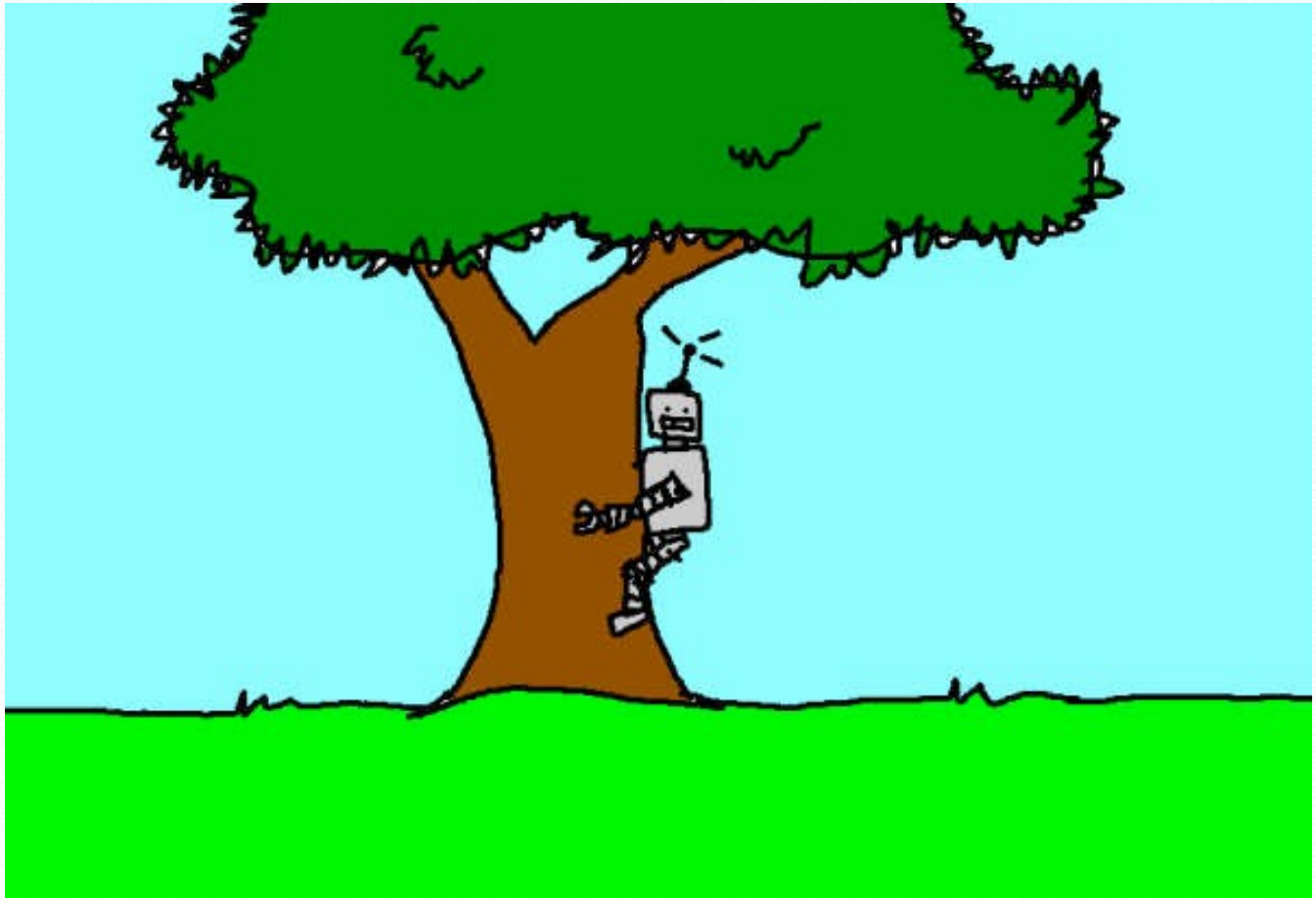
[Rich 1983]

Spanning Tree



All but N-1 edges are dropped to break cycles

Tree Traversal Algorithms

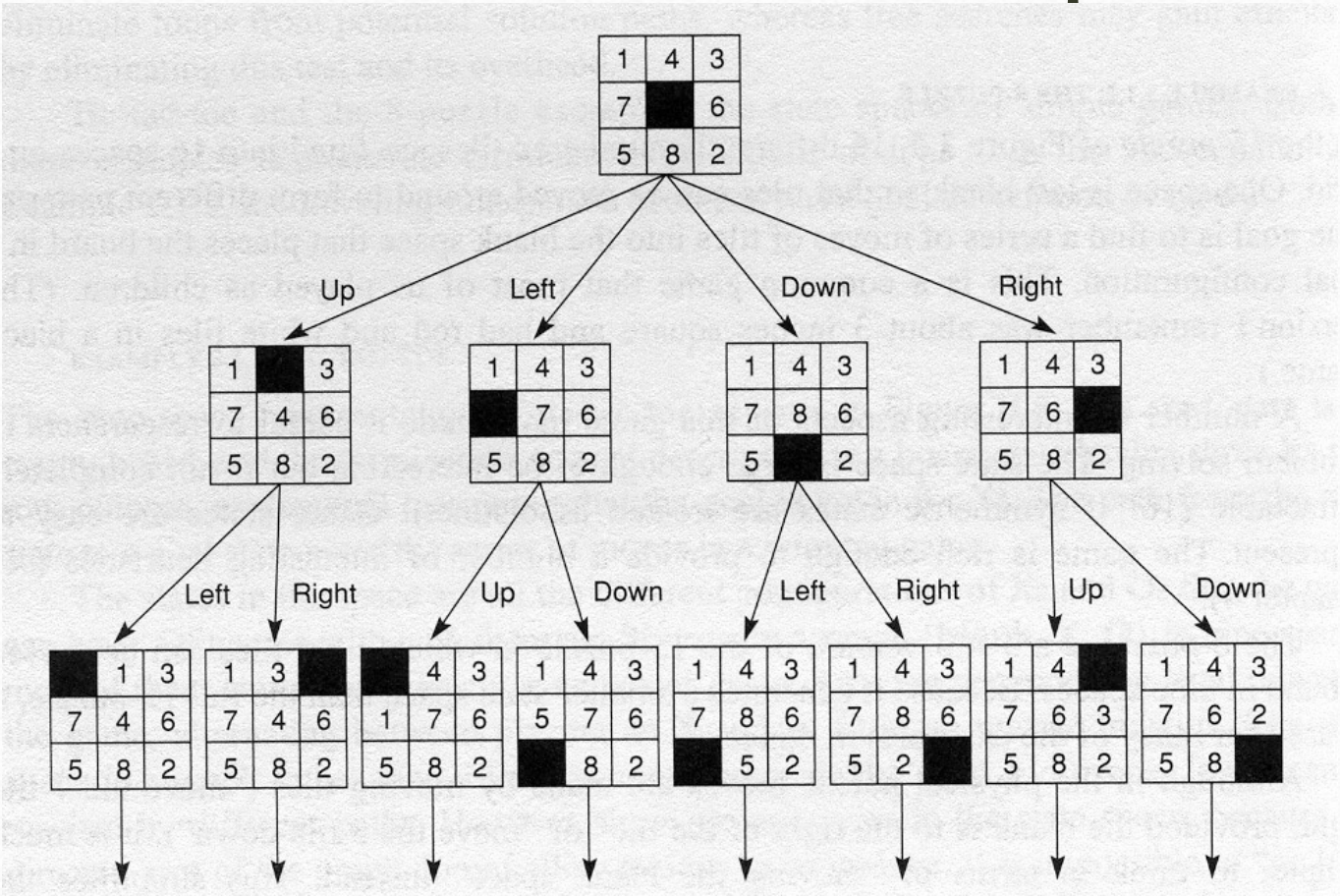


The 8-tile Puzzle

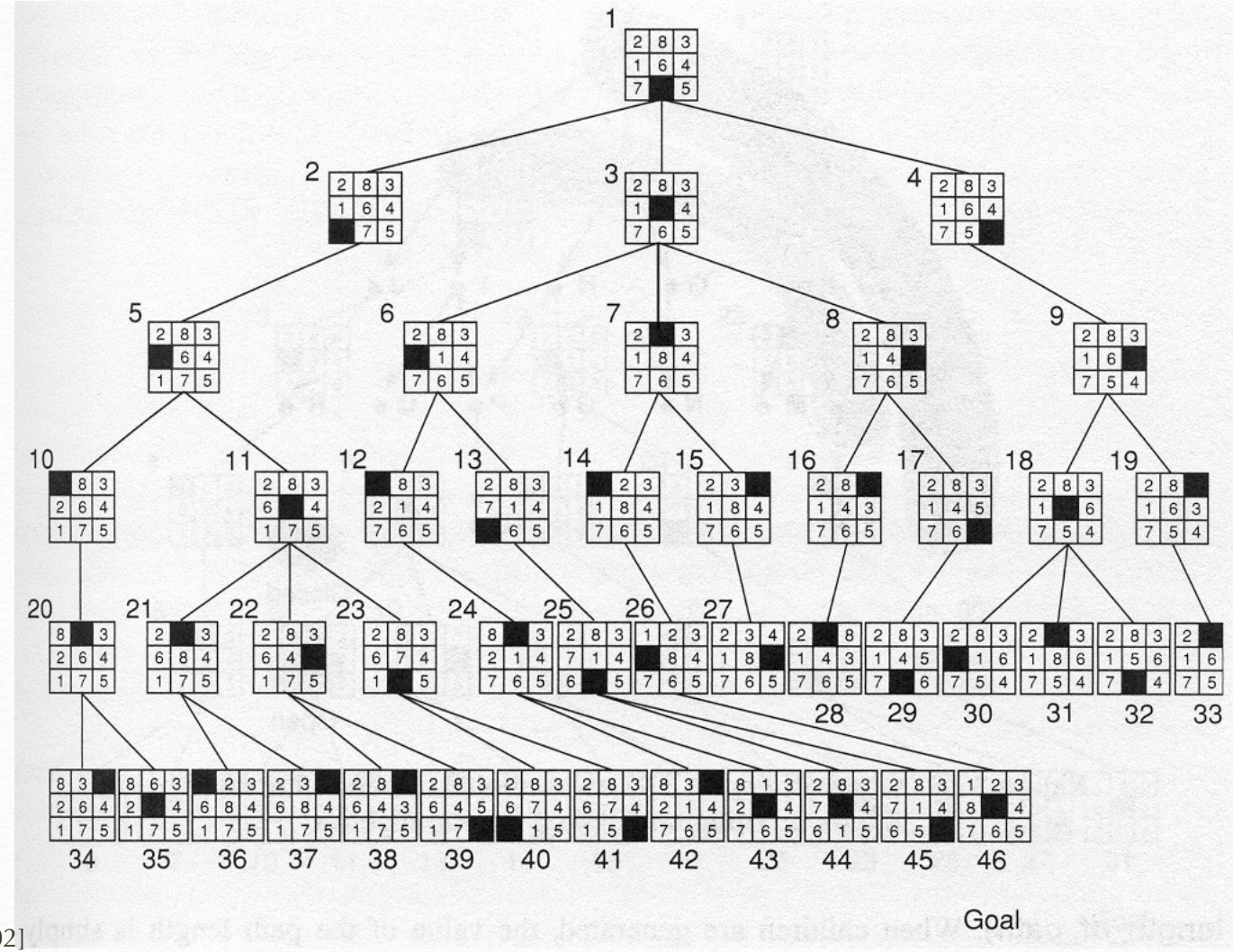
Start State

Goal State

8-tile Puzzle State Space

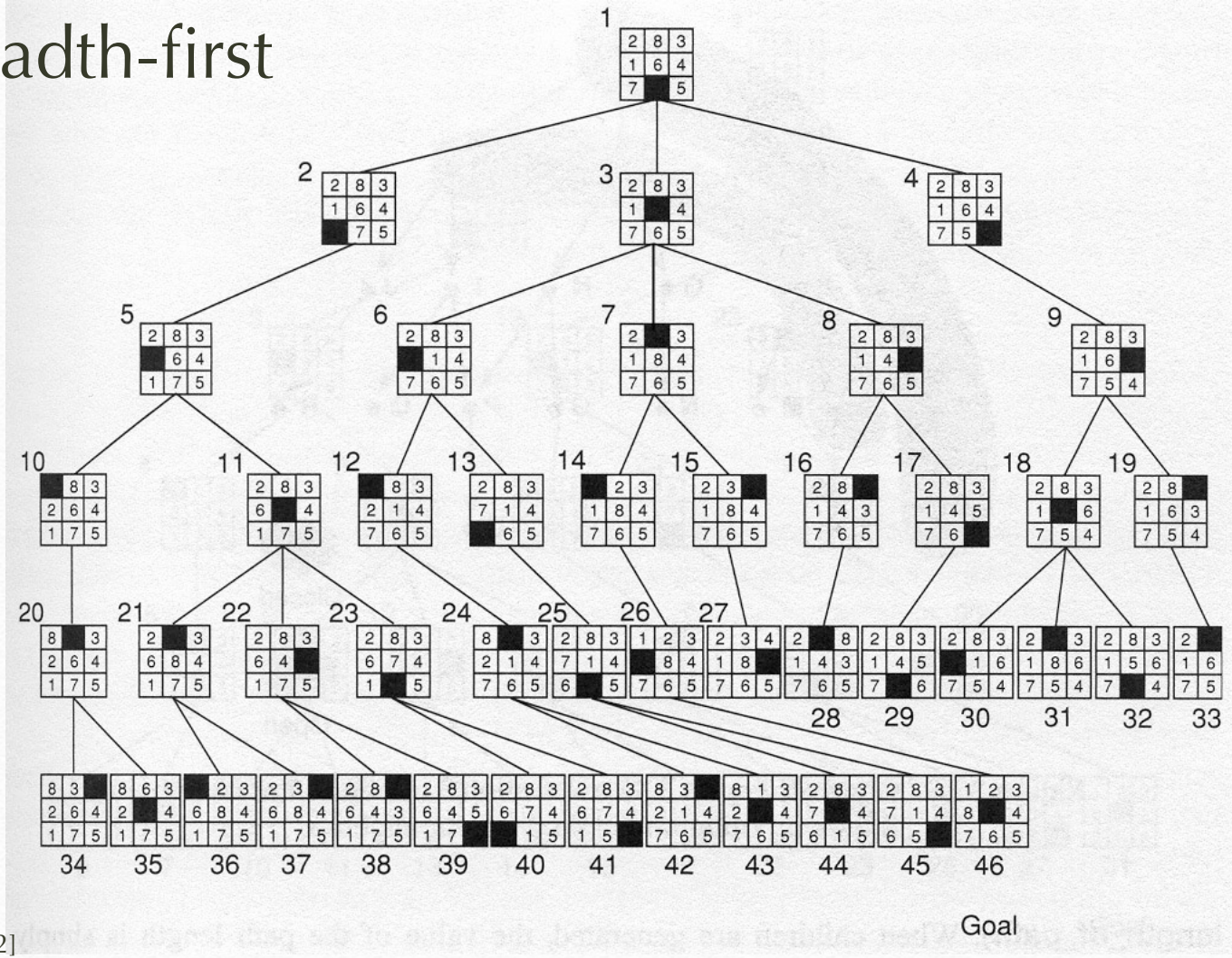


[Lugar 2002]


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
Goal

[Lugar 2002]

Breadth-first



[Lugar 2002]

Breadth-first search

Explores space in level-by-level fashion

- Always finds shortest path to goal
- When branching is great, takes up much space
- For some problems this makes it unusable.

```
function breadth-first-search
 open := [Start]
    done := [ ]
 while open != [ ] do
   X := open.dequeue()
   if (X = Goal) return success
        else
         generate children states of X
         discard any children in open, done
         for each child C
               open.enqueue(C)
         done.add(X)
 return failure
```

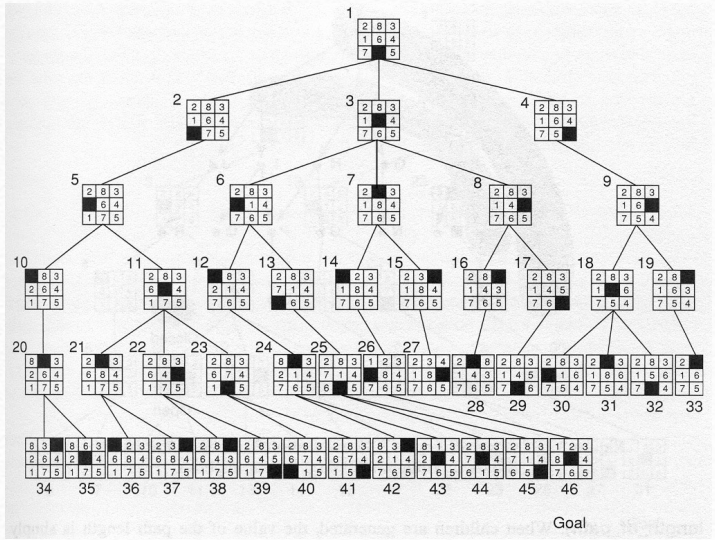
```
function breadth-first-search
 open := [Start]
    done := [ ]
 while open != [ ] do
   X := open.dequeue()
   if (X = Goal) return success
        else
         generate children states of X
         discard any children in open, done
         for each child C
               open.enqueue(C)
         done.add(X)
 return failure
```

Time: $O(b^{depth})$

Space: $O(b \cdot depth)$

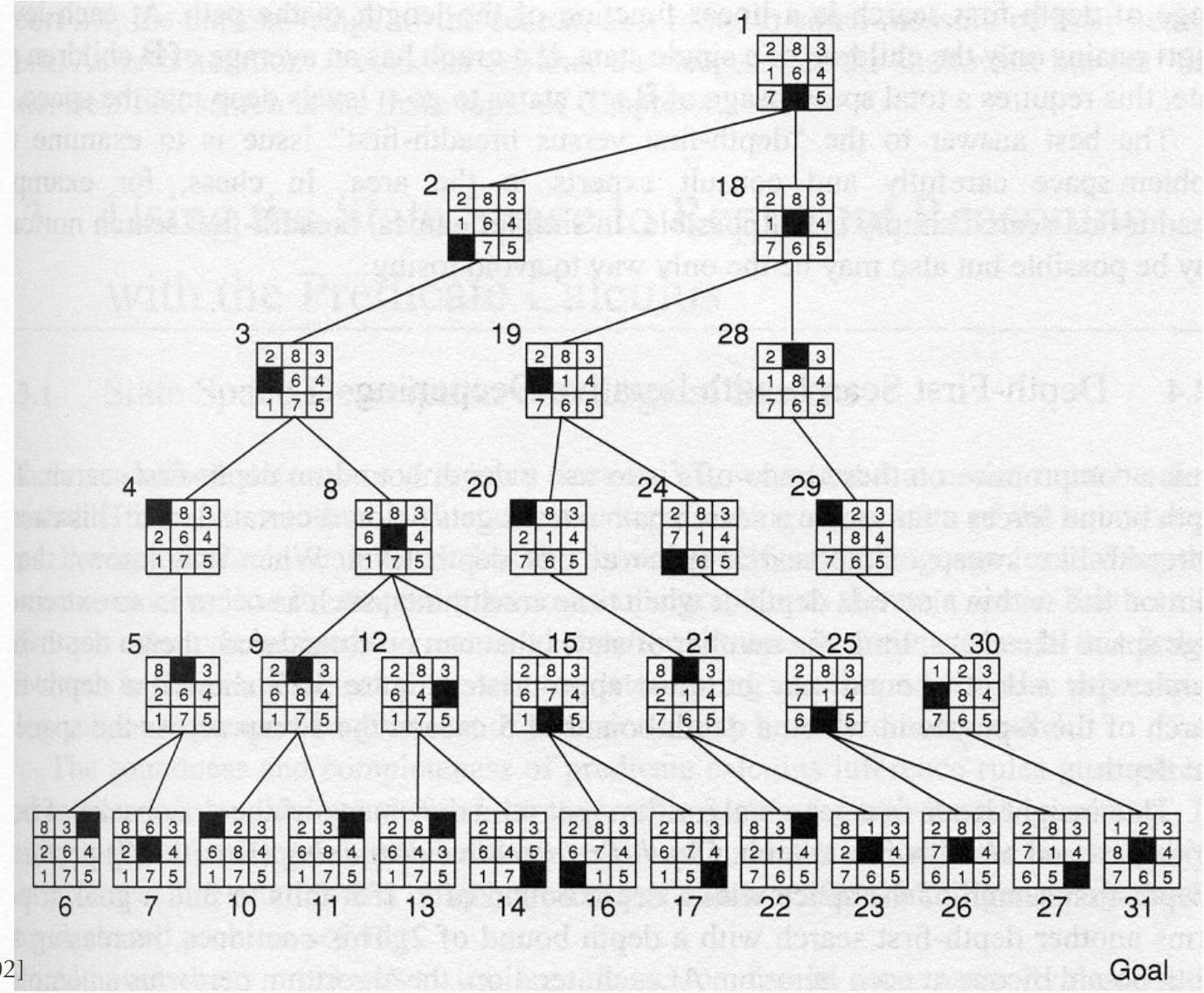
function **breadth-first-search**

```
 open := [Start]
    done := [ ]
 while open != [ ] do
   X := open.dequeue()
   if (X = Goal) return success
        else
         generate children states of X
         discard any children in open, done
         for each child C
               open.enqueue(C)
         done.add(X)
 return failure
```



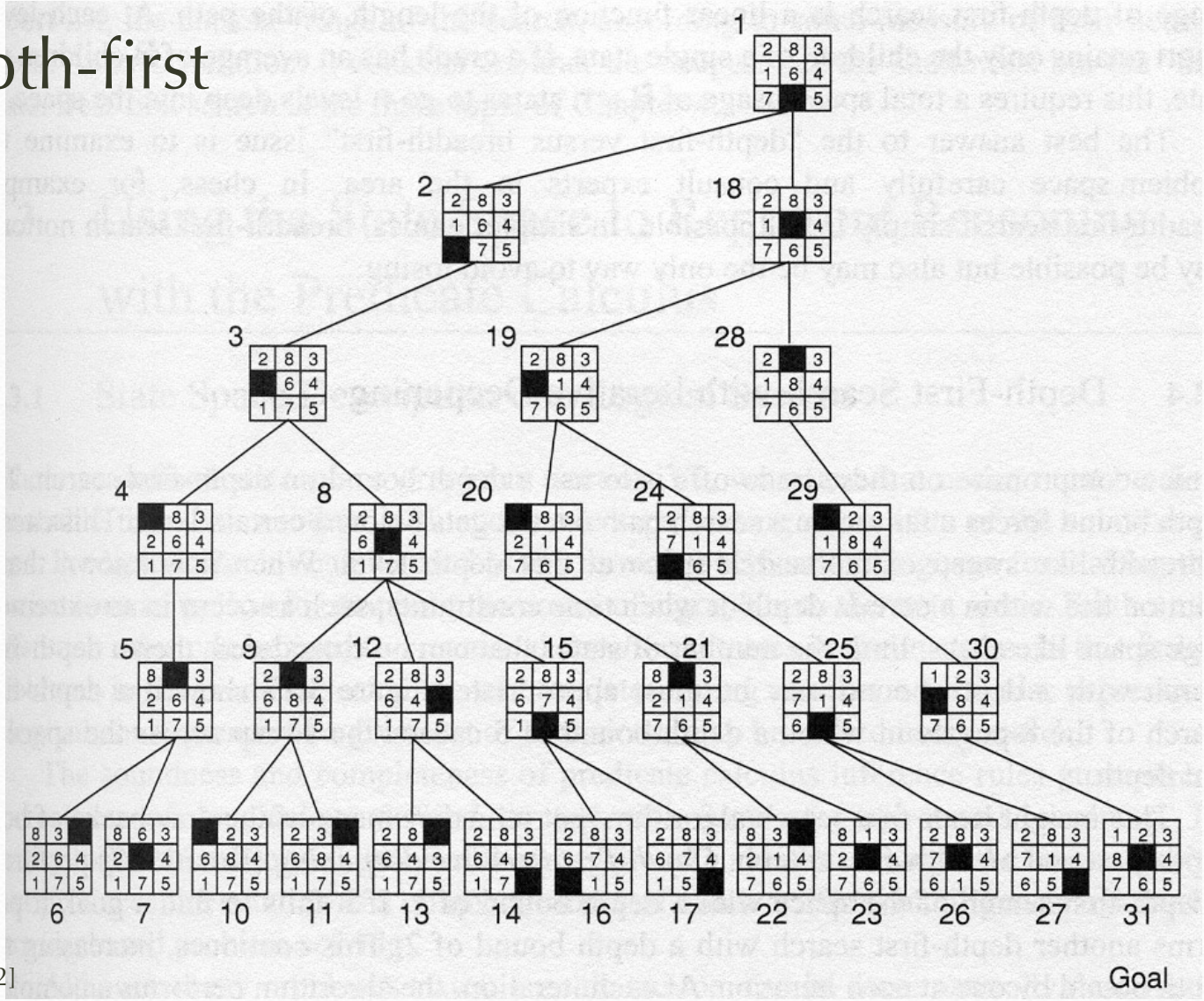
Time: $O(b^{depth})$

Space: $O(b \cdot depth)$


1
2
18
3
19
28
4
8
20
24
29
5
9
12
15
21
25
30
6
7
10
11
13
14
16
17
22
23
26
27
31
Goal

[Lugar 2002]

Depth-first



[Lugar 2002]

Depth-first search

- Follows each path to its conclusion
- May not find the shortest state
- Quickly gets deep into problem space
- Good when there are (many) deep solutions
- Space efficient when high branching factor

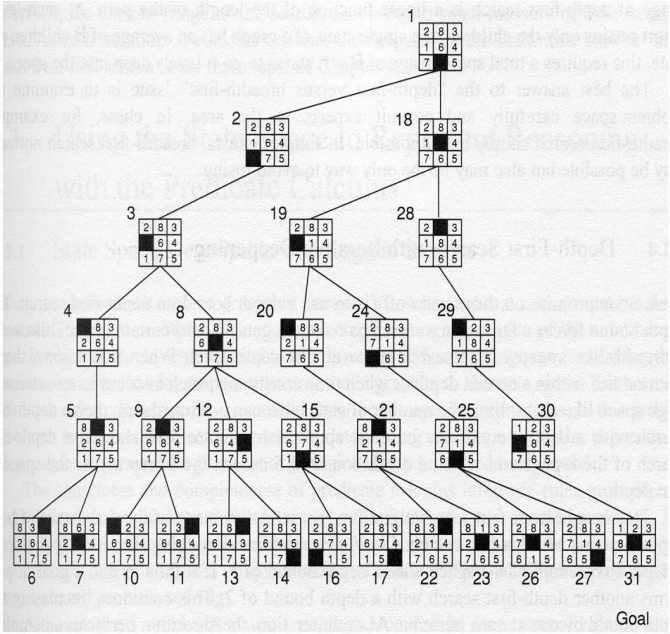
```
function depth-first-search
 open := [Start]
 done := [ ]
 while open != [ ] do
   X := open.pop()
   if (X = Goal) return success
        else
         generate children states of X
         discard any children in open, done
         for each child C
               open.push(C)
         done.add(X)
 return failure
```

```
function depth-first-search
 open := [Start]
 done := [ ]
 while open != [ ] do
   X := open.pop()
   if (X = Goal) return success
        else
         generate children states of X
         discard any children in open, done
         for each child C
              open.push(C)
         done.add(X)
 return failure
```

Time: $O(b^{maxdepth})$

Space: $O(b \cdot maxdepth)$

function **depth-first-search**
 open := [Start]
 done := [ ]
 while open != [ ] do
  X := open.pop()
  if (X = Goal) return success
      else
       generate children states of X
       discard any children in open, done
       for each child C
            open.push(C)
       done.add(X)
 return failure



Time: $O(b^{maxdepth})$

Space: $O(b \cdot maxdepth)$

Depth-limited search

- Depth first search with depth limit
- Don’t add/search nodes deeper than limit
- May not find a solution; but may
  - find shallower solution
  - find solution more quickly

Depth-limited search

- Depth first search with depth limit
- Don't add/search nodes deeper than limit
- May not find a solution; but may
  - find shallower solution
  - find solution more quickly

Time: $O(b^{limit})$

Space: $O(b \cdot limit)$

Iterative deepening

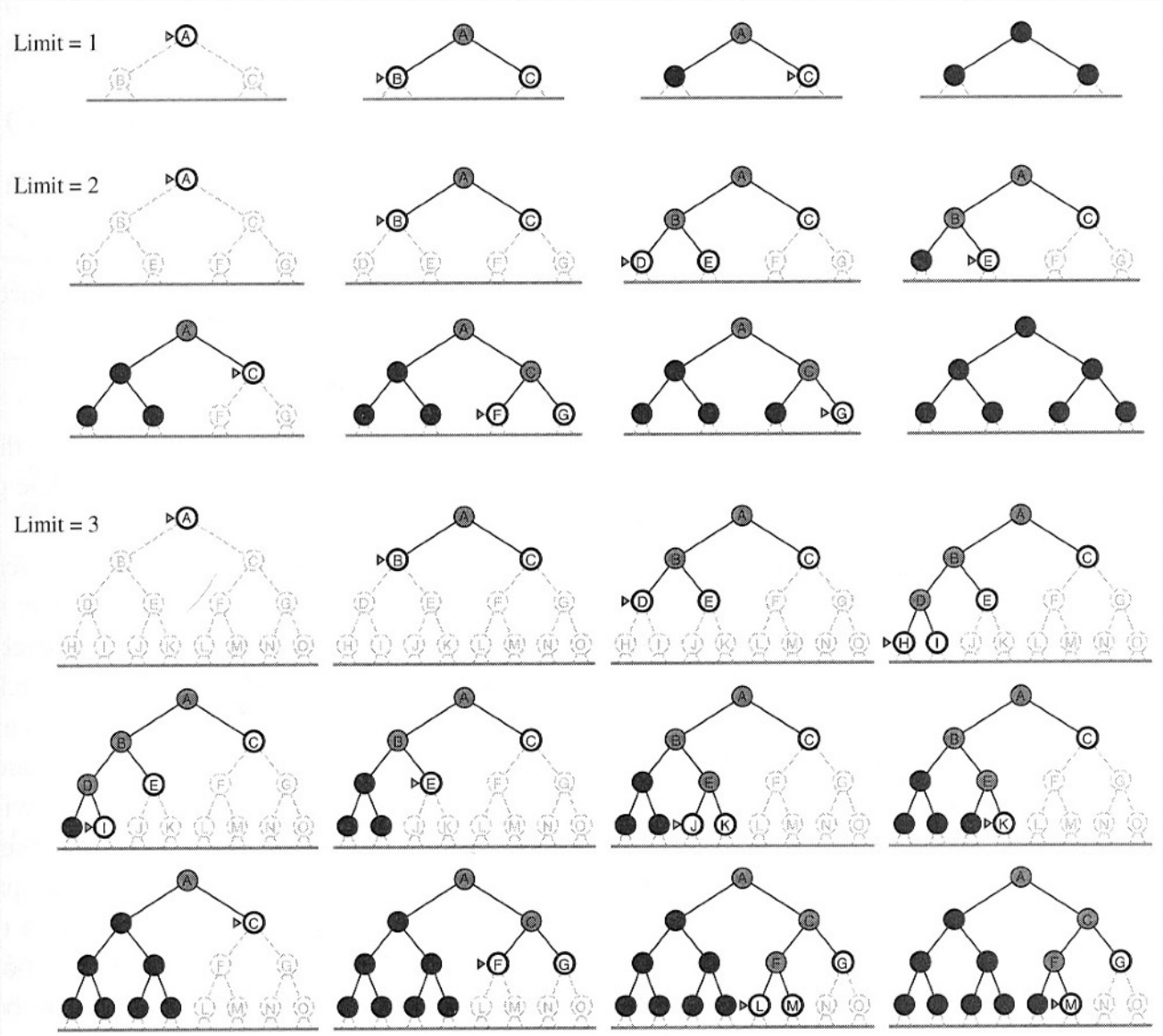
- Depth-first search increasing depth limit
- Combines depth and breadth-first search
- Finds shallowest path
- Preferred method when search space is large and depth of solution is unknown.

Iterative deepening

- Depth-first search increasing depth limit
- Combines depth and breadth-first search
- Finds shallowest path
- Preferred method when search space is large and depth of solution is unknown.

Time: $O(b^{limit})$

Space: $O(b \cdot limit)$



[Russell & Norvig 2003]

Two Assumptions

Two Assumptions

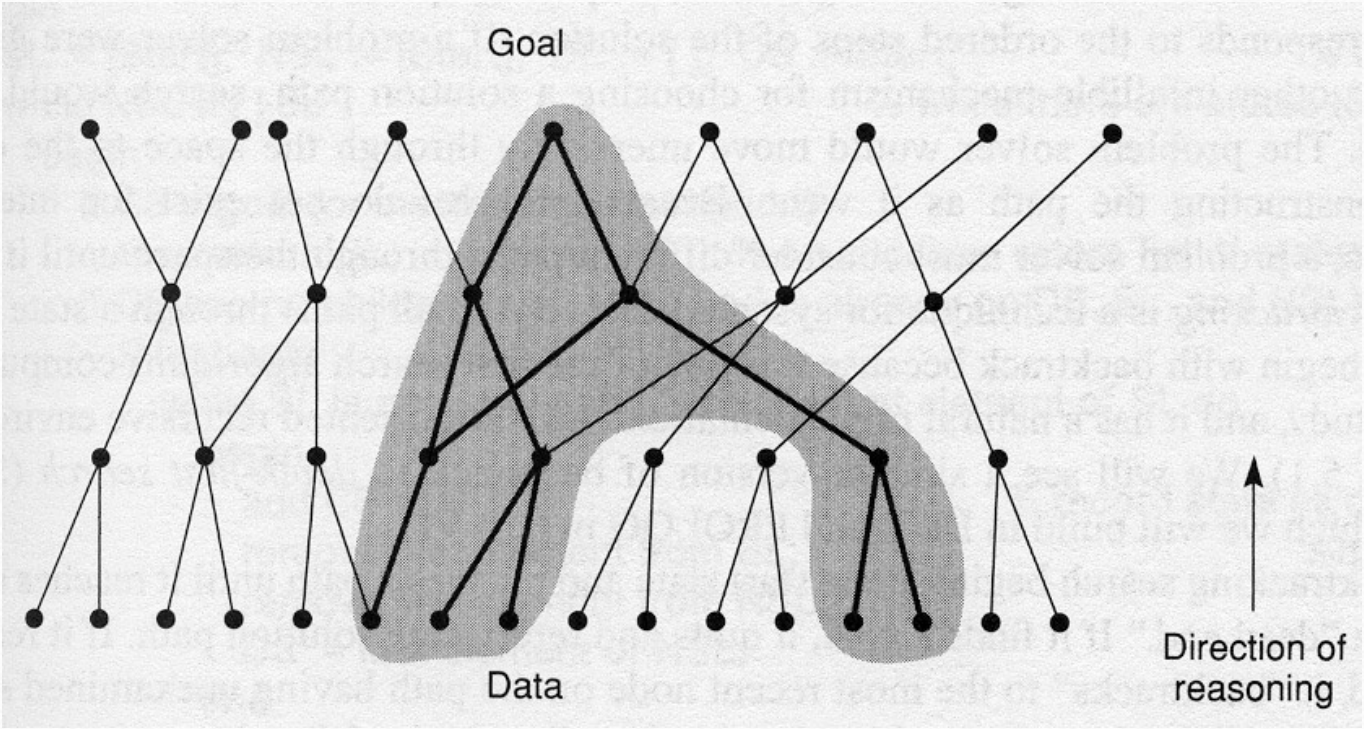
Working from start state to goal state.
That is, in a forward direction.

Two Assumptions

Working from start state to goal state.
That is, in a forward direction.

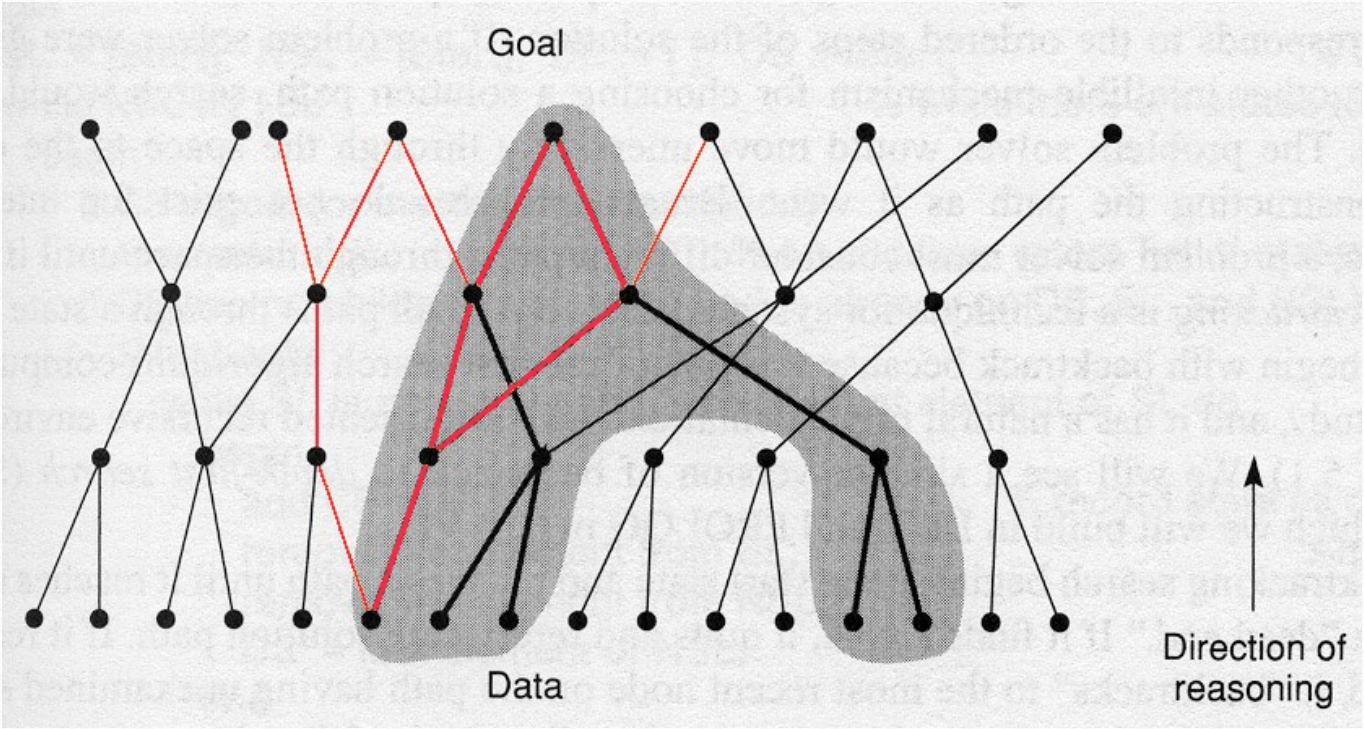
Searching blind, e.g. exhaustively.

Forward (data-driven)

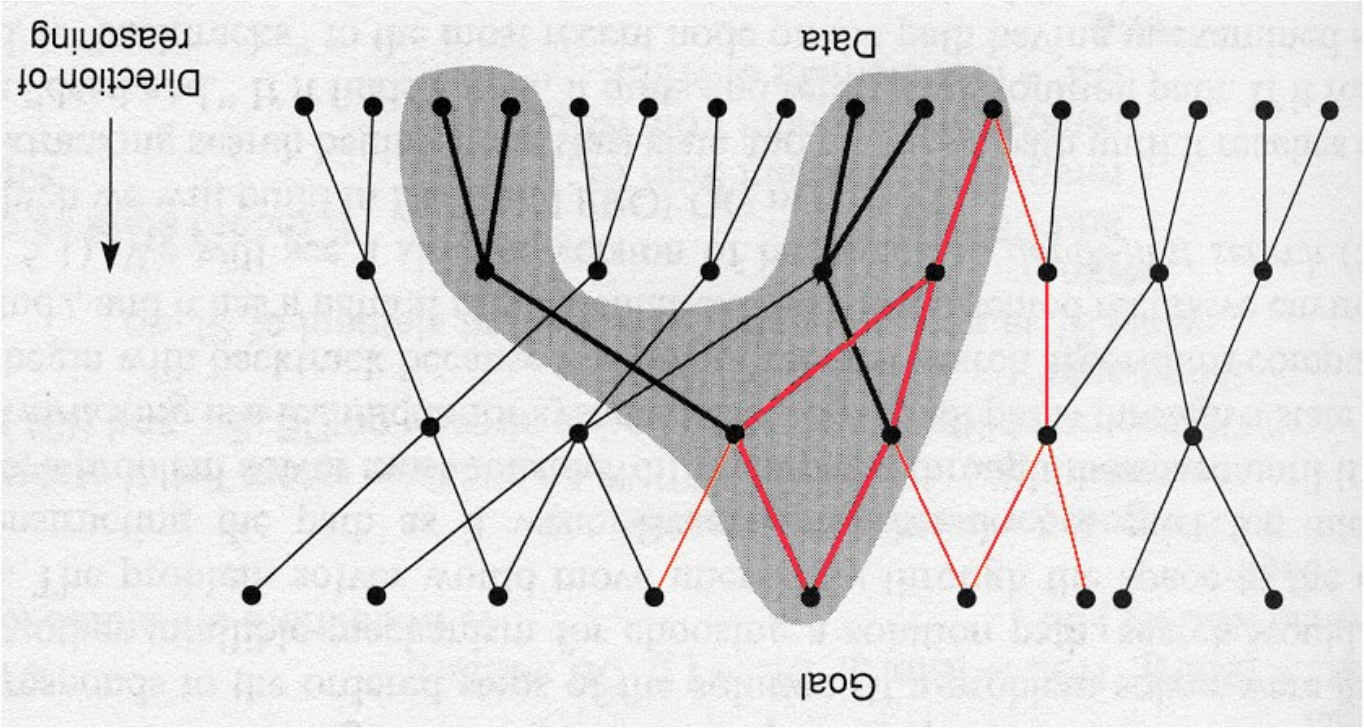


[Lugar 2002]

Forward (data-driven)



Forward (data-driven)

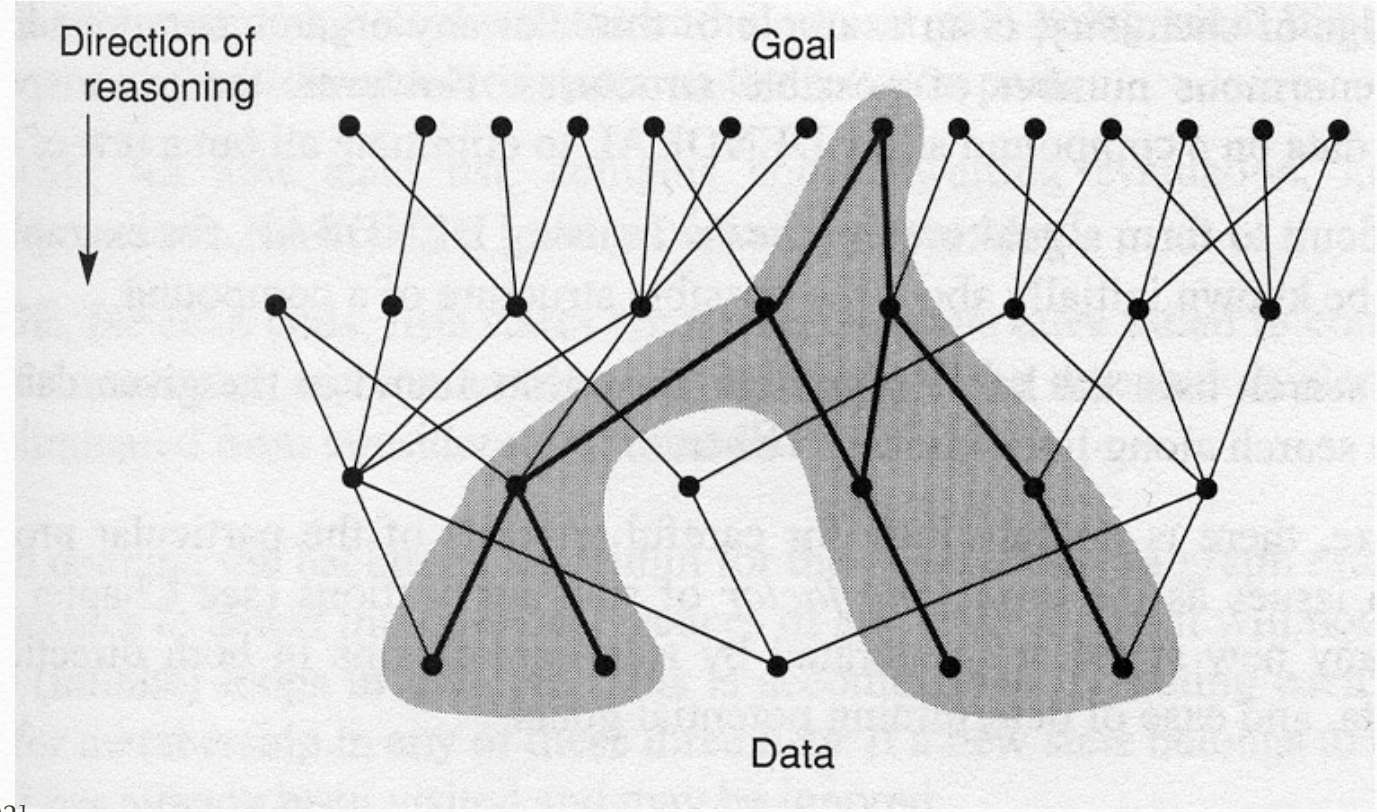


Forward (data-driven)

Necessary when you can identify the goal but can’t state it.

Preferred if there are a large number of goals, and few branches

Backward (goal-directed)



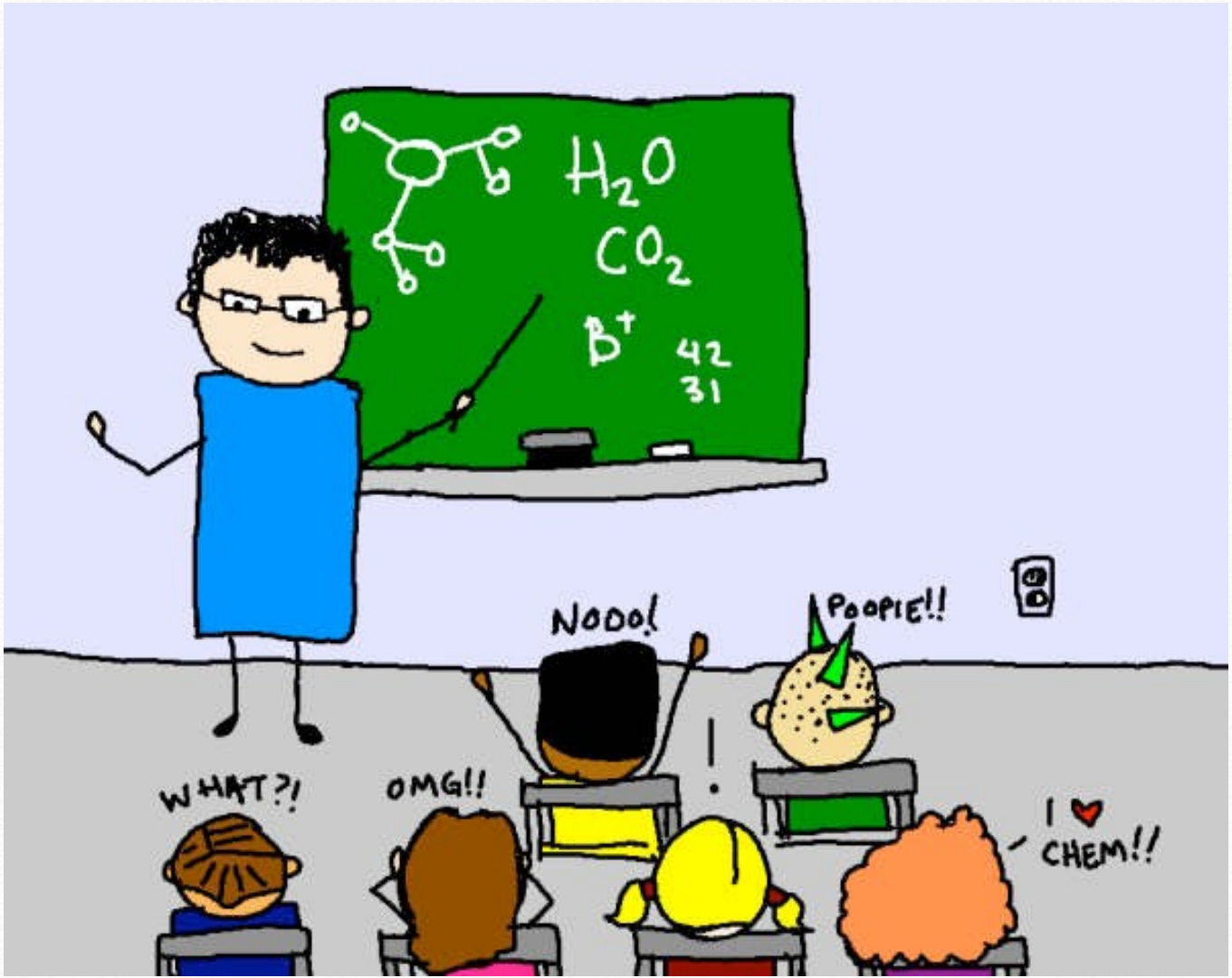
[Lugar 2002]

Backward (goal-directed)

Preferred if there is a clearly identifiable goal, and forward search has many branches

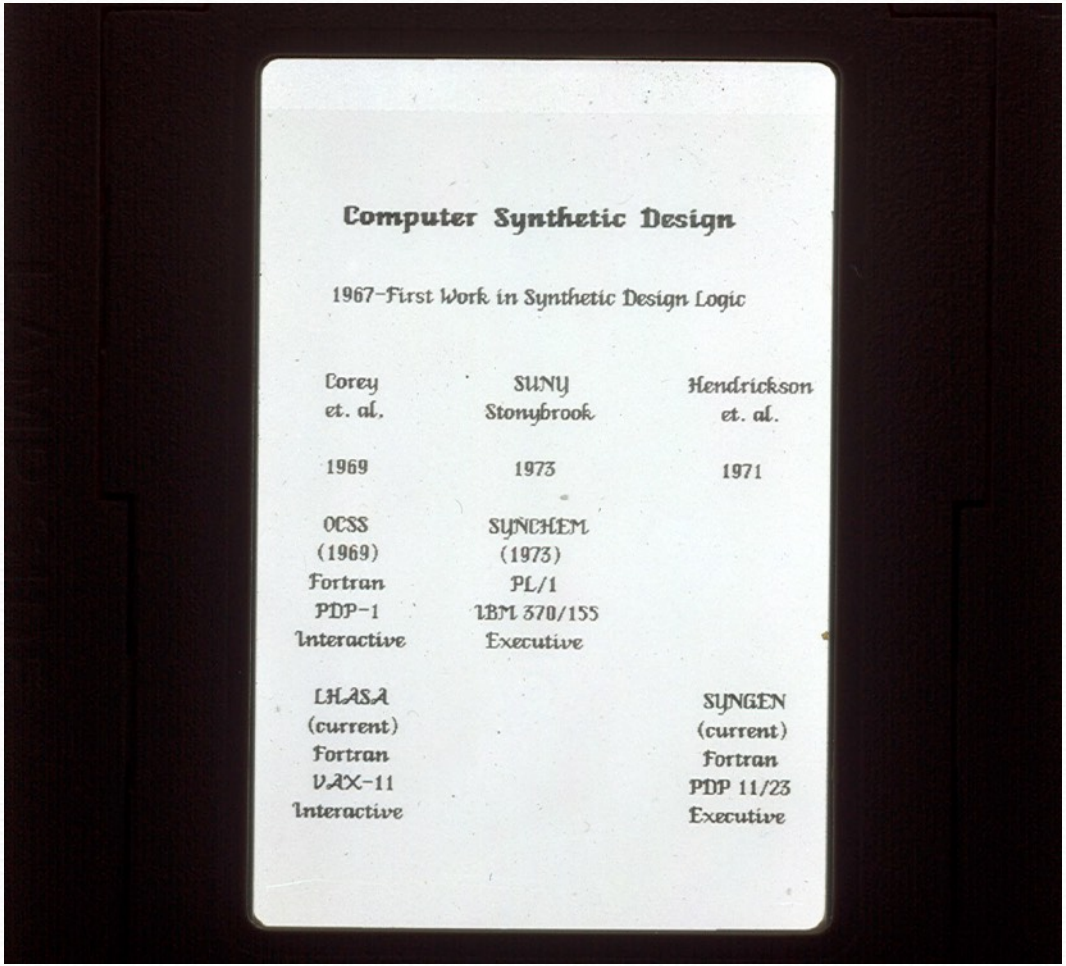
Preferred if you have potential for gaining more information

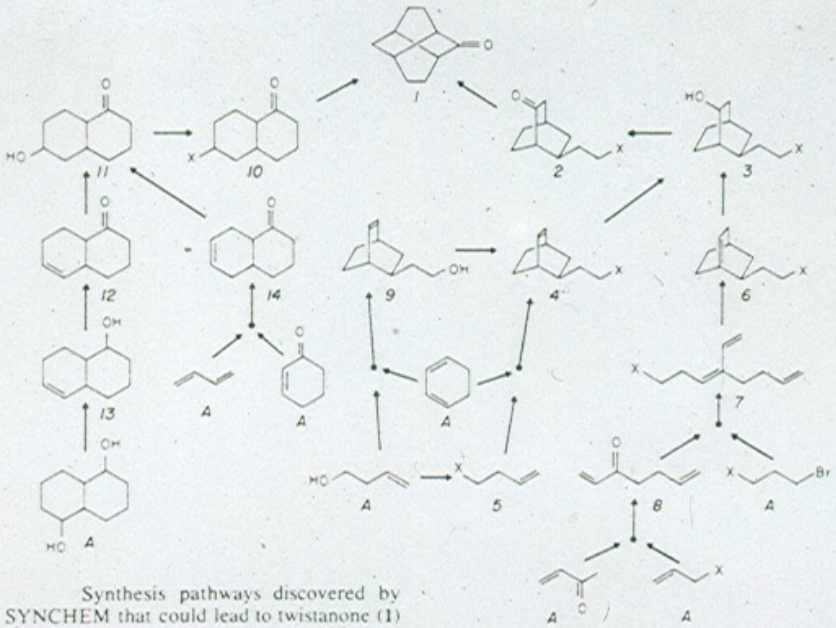
e.g. in Medicine, can perform diagnostic tests


H_2O
CO_2
B^+
42
31
NOOO!
POOPIE!!
WHAT?!
OMG!!
!
I ♥ CHEM!!

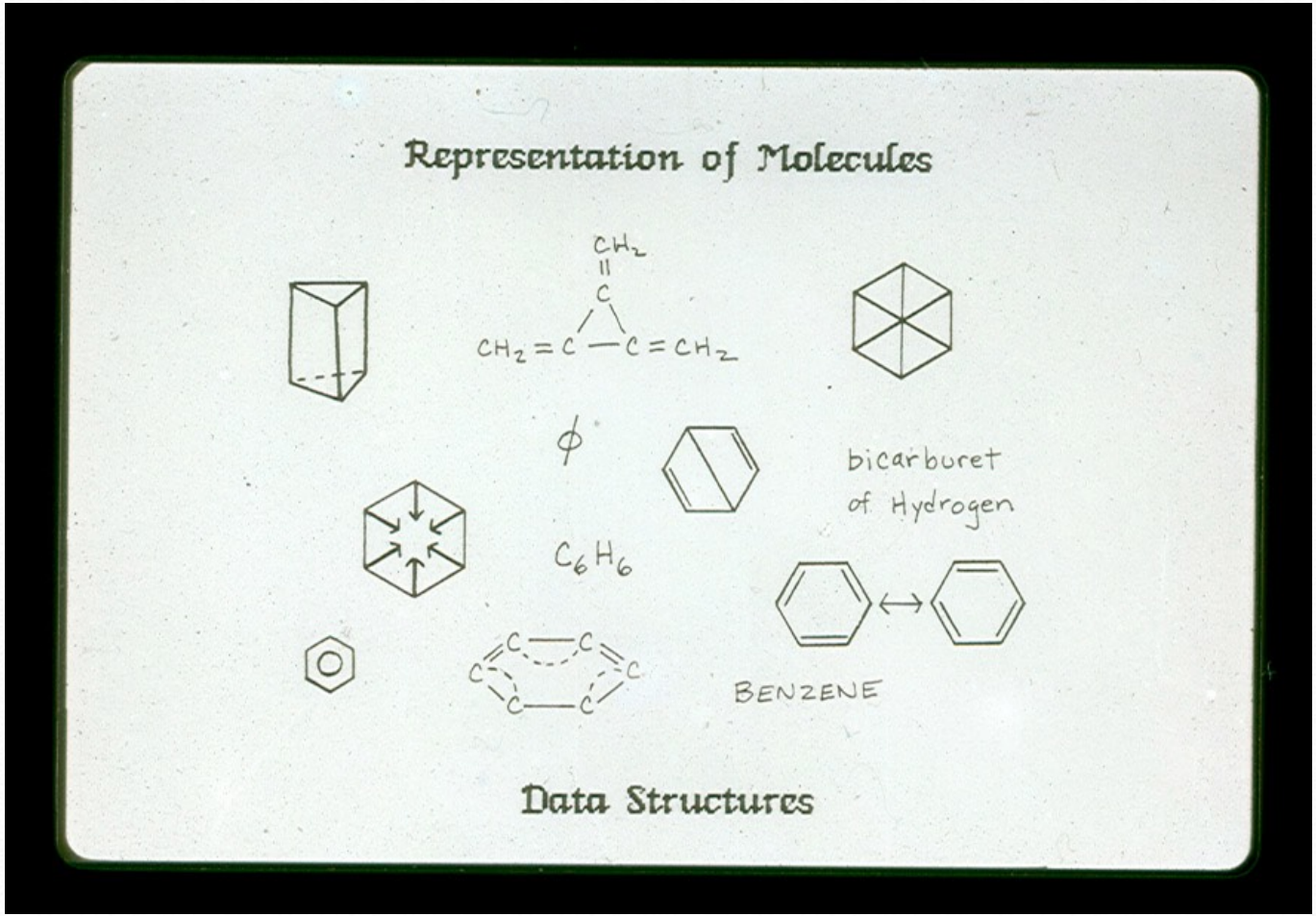

Synthia: A Synthetic Design Program
M. Anthony Kapolka III
CH 500
Spring 1987

Your instructor as Chemist.


Computer Synthetic Design
1967-First Work in Synthetic Design Logic
Corey et. al.
SUNY Stonybrook
Hendrickson et. al.
1969
1973
1971
OCSS (1969) Fortran PDP-1 Interactive
SYNCHEM (1973) PL/1 IBM 370/155 Executive
LHASA (current) Fortran VAX-11 Interactive
SYNGEN (current) Fortran PDP 11/23 Executive



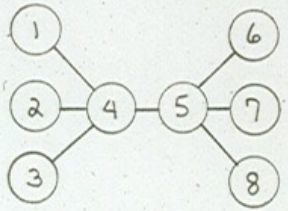
Synthesis pathways discovered by SYNCHEM that could lead to twistanone (1) from available starting materials. Reaction types used: alkylation alpha to a ketone (1 ← 2, 1 ← 10, 8 ← A + A); oxidation of a secondary alcohol (2 ← 3, 12 ← 13); hydration of an alkene (3 ← 4, 3 ← 6, 11 ← 12, 13 ← A); Diels-Alder reaction (4 ← 5 + A, 6 ← 7, 9 ← A + A, 14 ← A + A); Wittig reaction (7 ← 8 + A); and replacement of an alcohol by a better leaving group (10 ← 11, 4 ← 9, 5 ← A). All compounds labeled A were found by SYNCHEM on its list of available compounds.


Representation of Molecules
CH_2
C
$CH_2 = C - C = CH_2$
ϕ
bicarburet
of Hydrogen
C_6H_6
BENZENE
Data Structures

Adjacency Matrix

	1	2	3	4	5	6	7	8
1	0	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0	0
4	1	1	1	0	1	0	0	0
5	0	0	0	1	0	1	1	1
6	0	0	0	0	1	0	0	0
7	0	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0	0

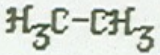
Unidigraph

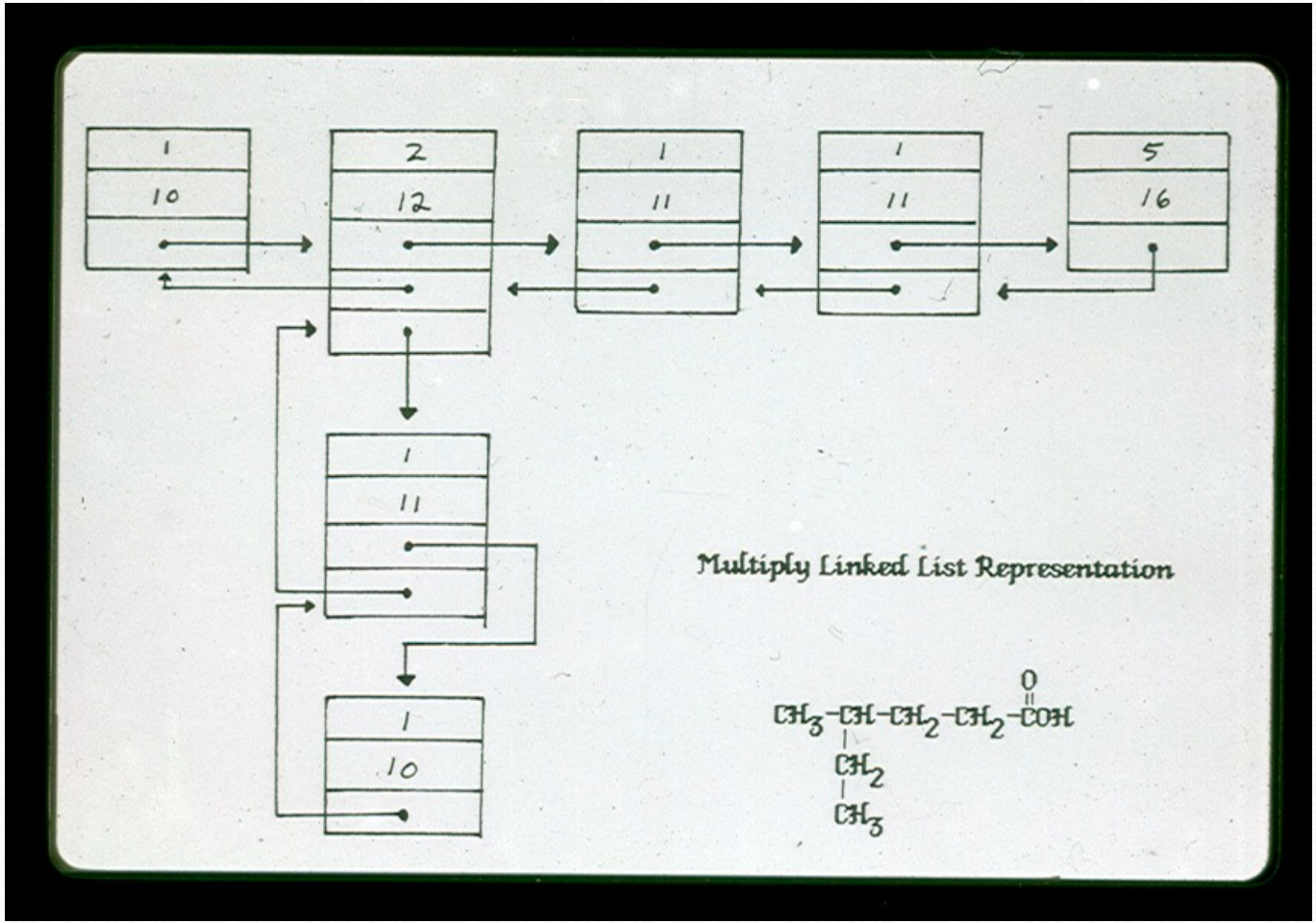


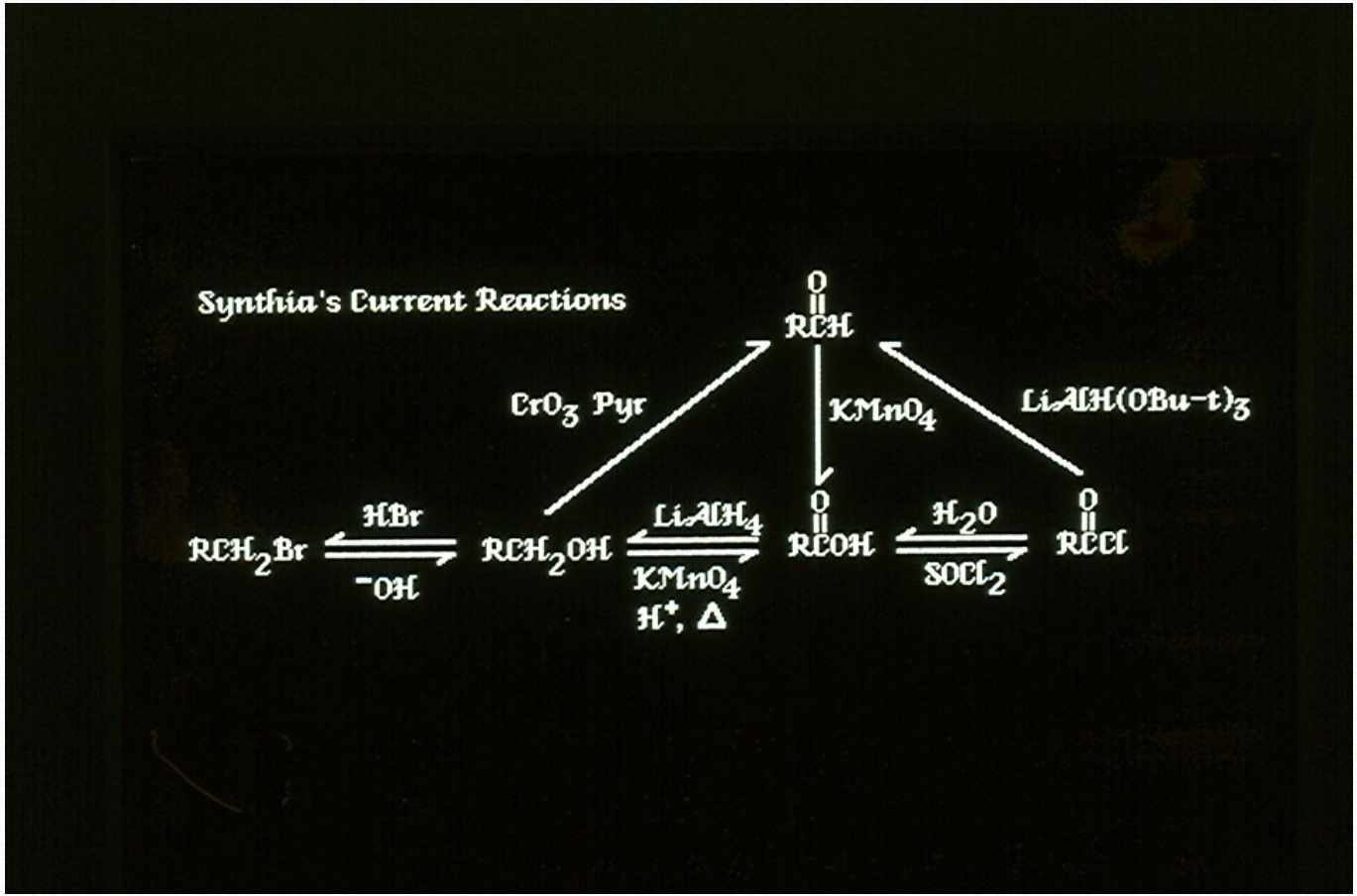
Vertex Code

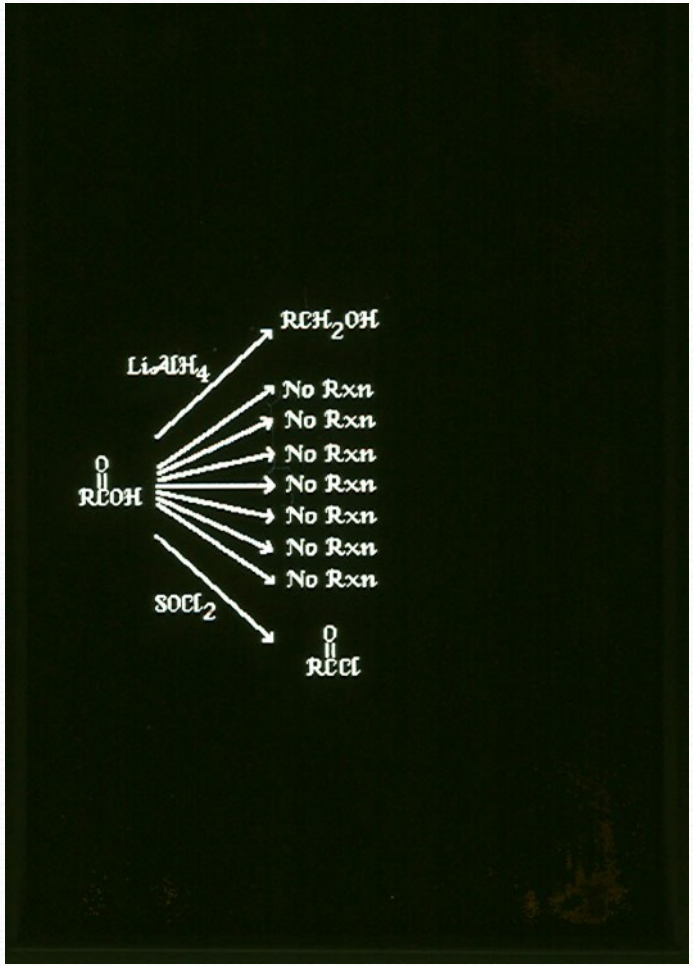
1	H	5	C
2	H	6	H
3	H	7	H
4	C	8	H

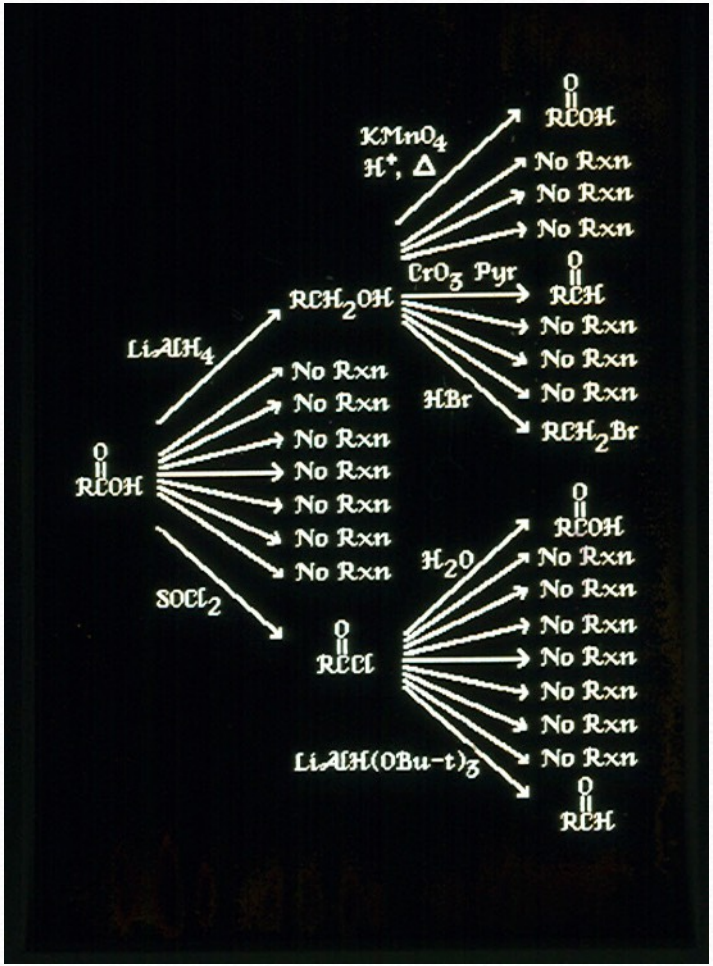
Molecule

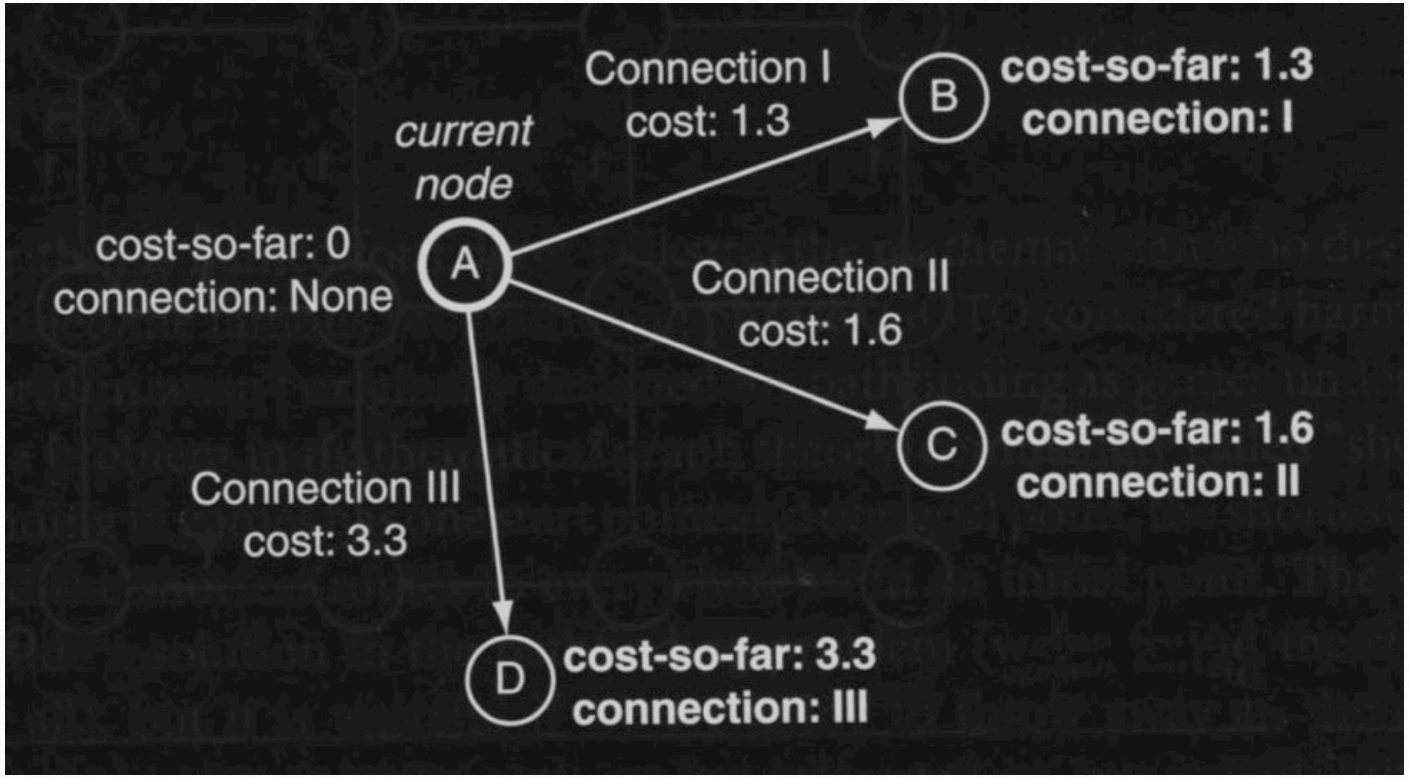



1
10
2
12
1
11
1
11
5
16
1
11
1
10
Multiply Linked List Representation
O
CH3-CH-CH2-CH2-COH
CH2
CH3

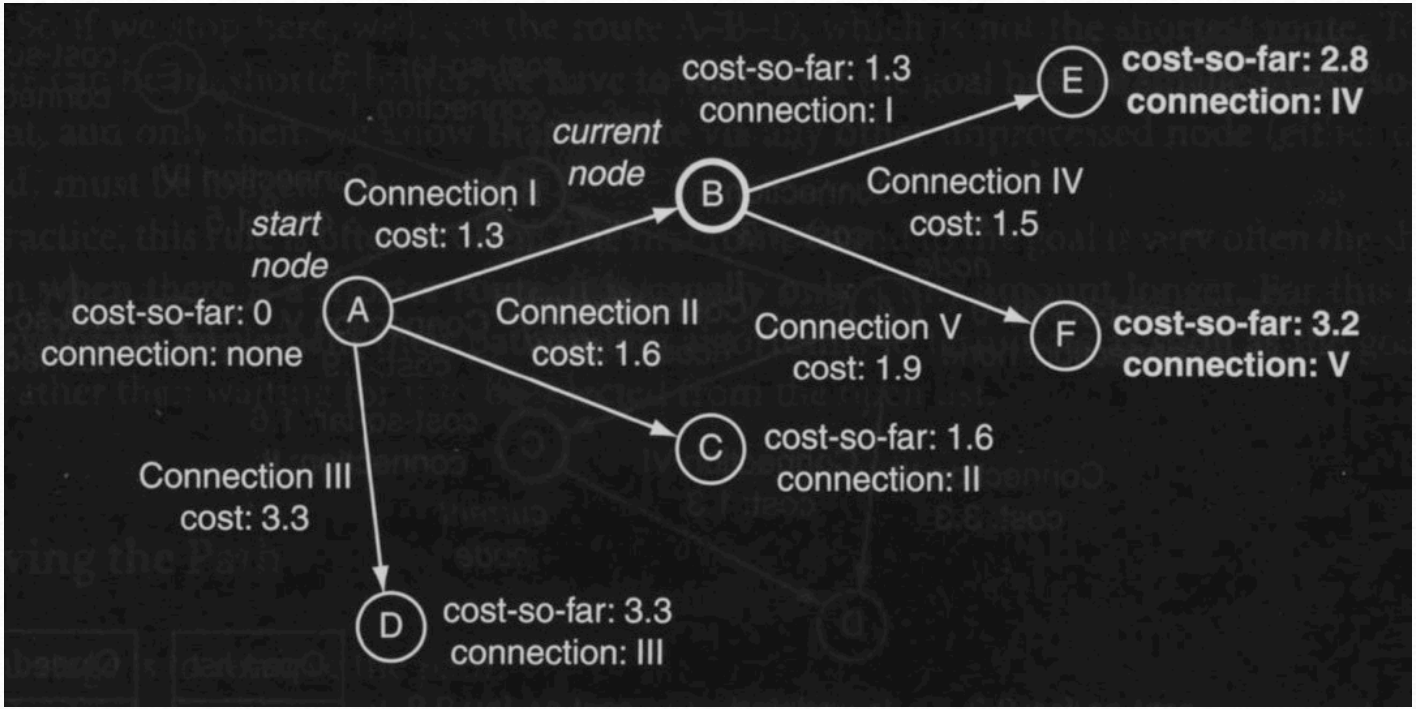

Synthia's Current Reactions
RCH
CrO3 Pyr
KMnO4
LiAlH(OBu-t)3
RCH2Br
HBr
-OH
RCH2OH
LiAlH4
KMnO4
H+, Δ
RCOH
H2O
SOCl2
RCCl


RCH2OH
LiAlH4
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
O
RCOH
SOCl2
O
RCCl

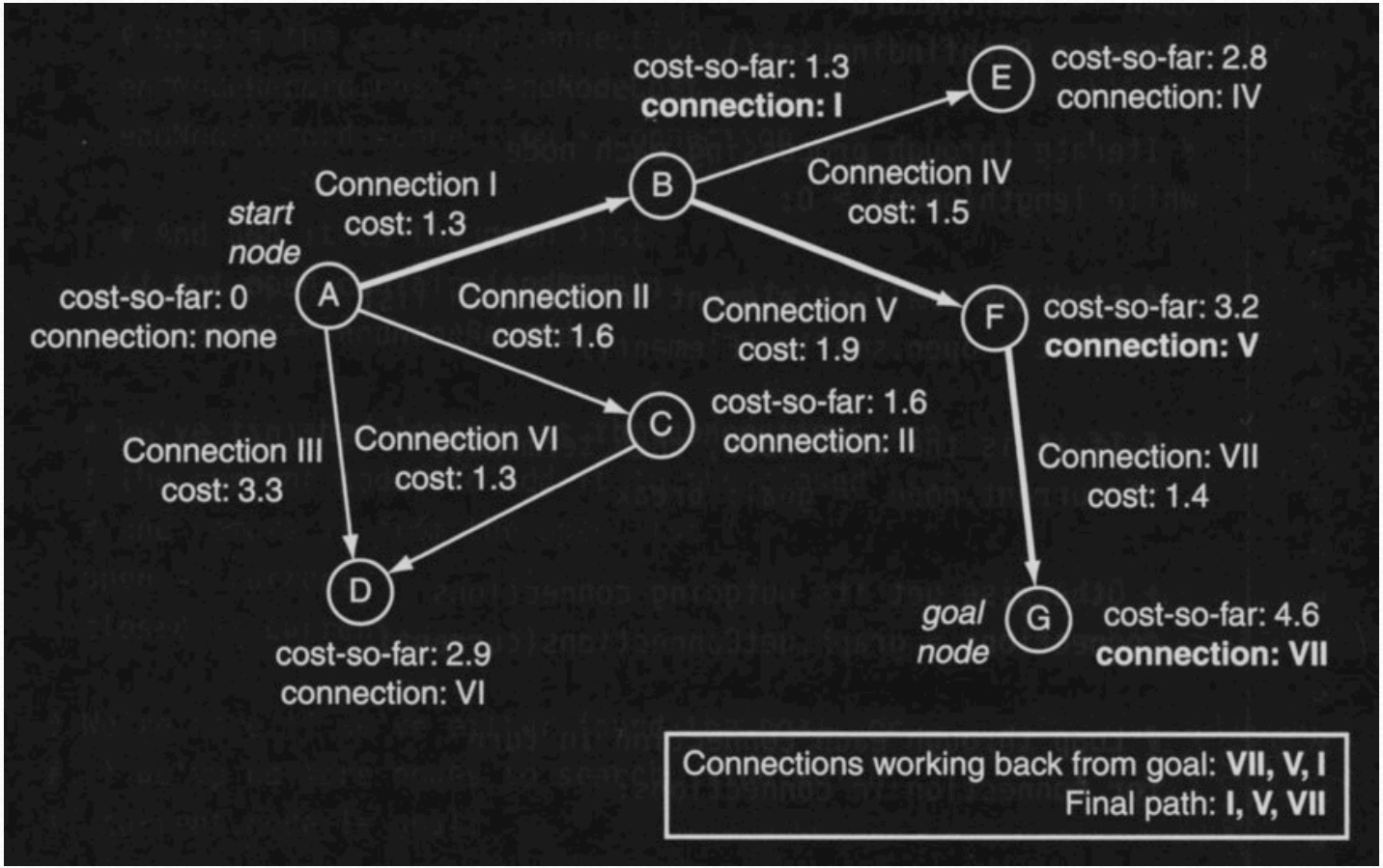

KMnO4
H+, Δ
RCOH
No Rxn
No Rxn
No Rxn
CrO3 Pyr
RCH2OH
RCH
No Rxn
No Rxn
No Rxn
LiAlH4
HBr
RCH2Br
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
RCOH
RCOH
H2O
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
No Rxn
SOCl2
RCCl
LiAlH(OBu-t)3
RCH



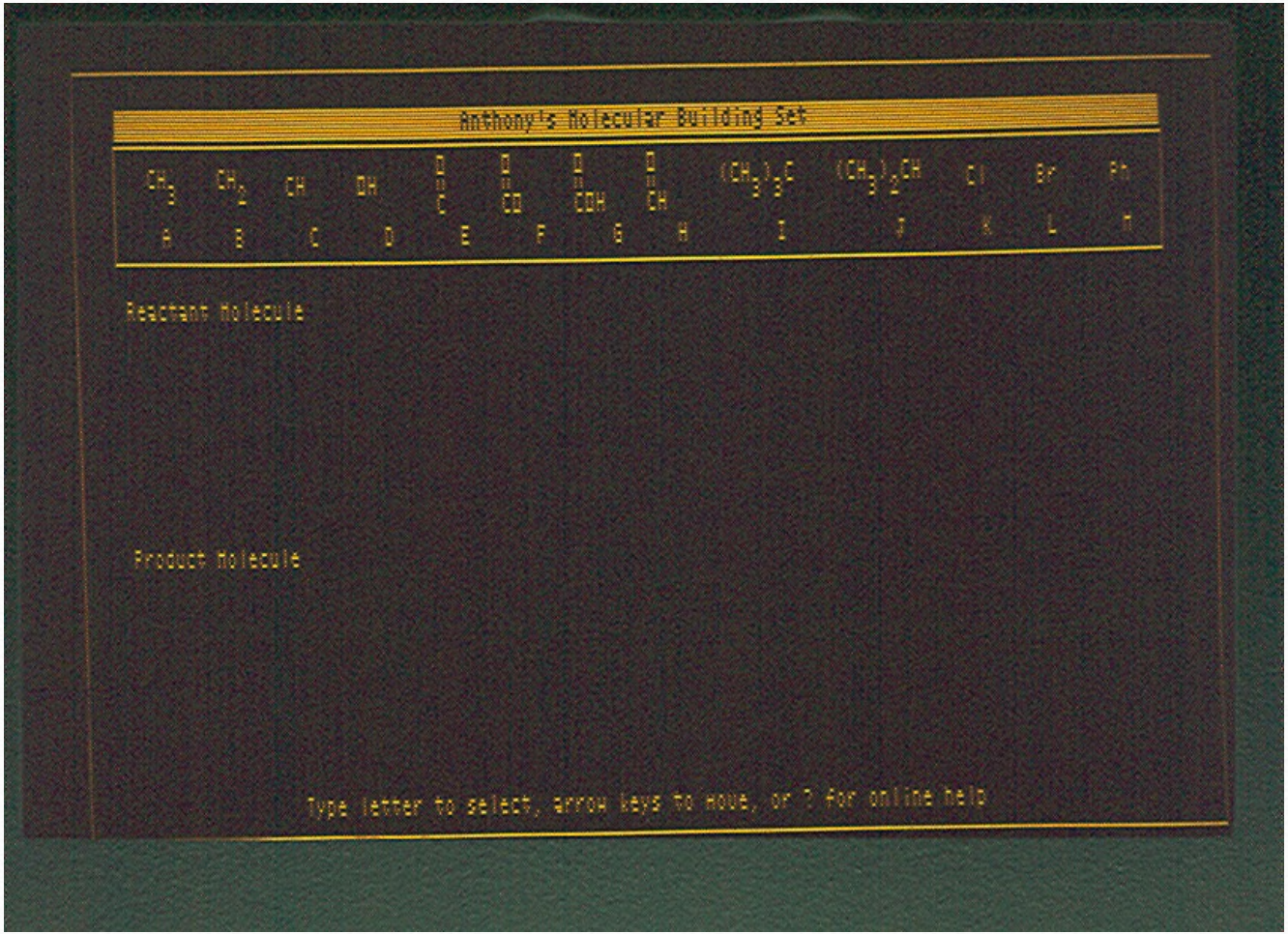
AI for Games, Millington & Funge: Figure 4.8

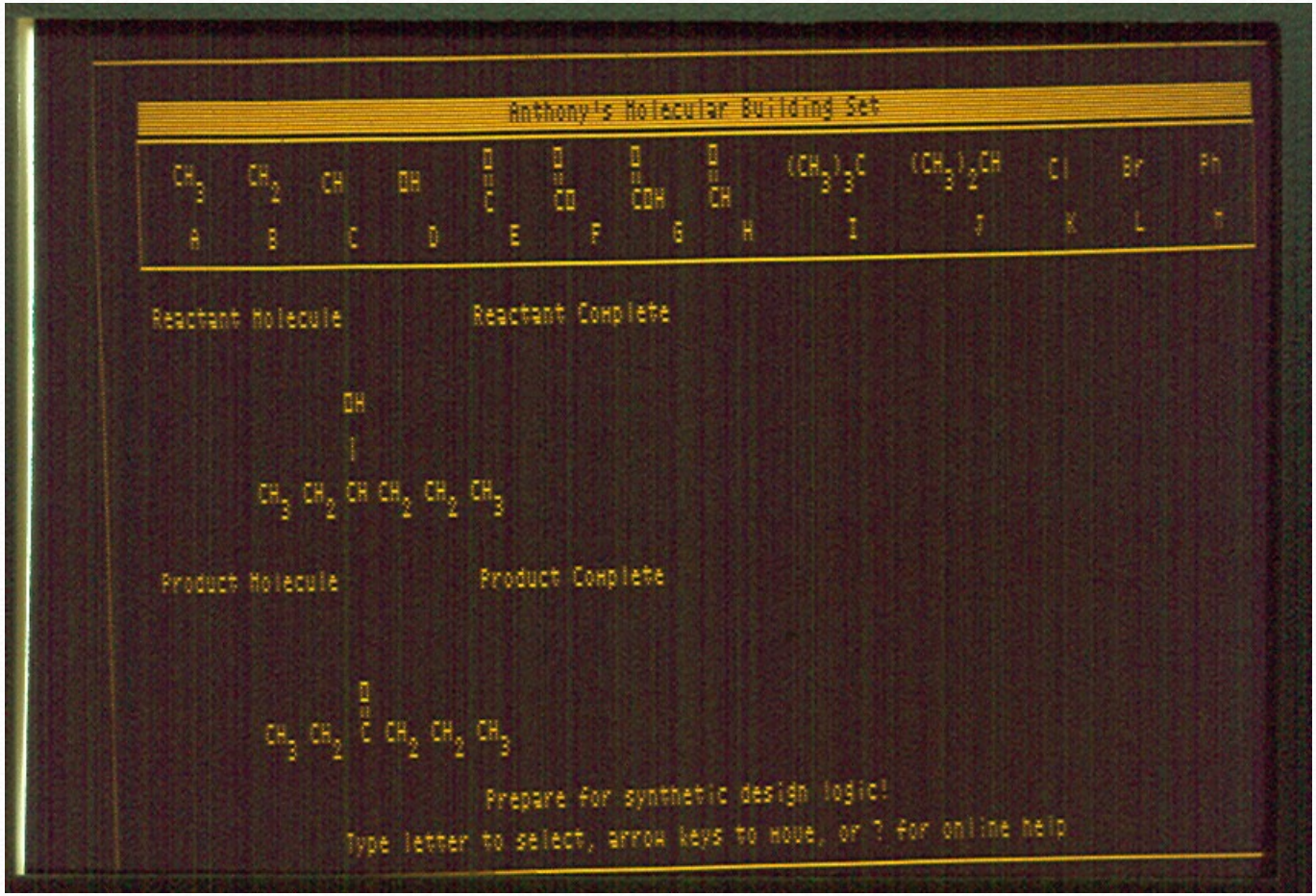


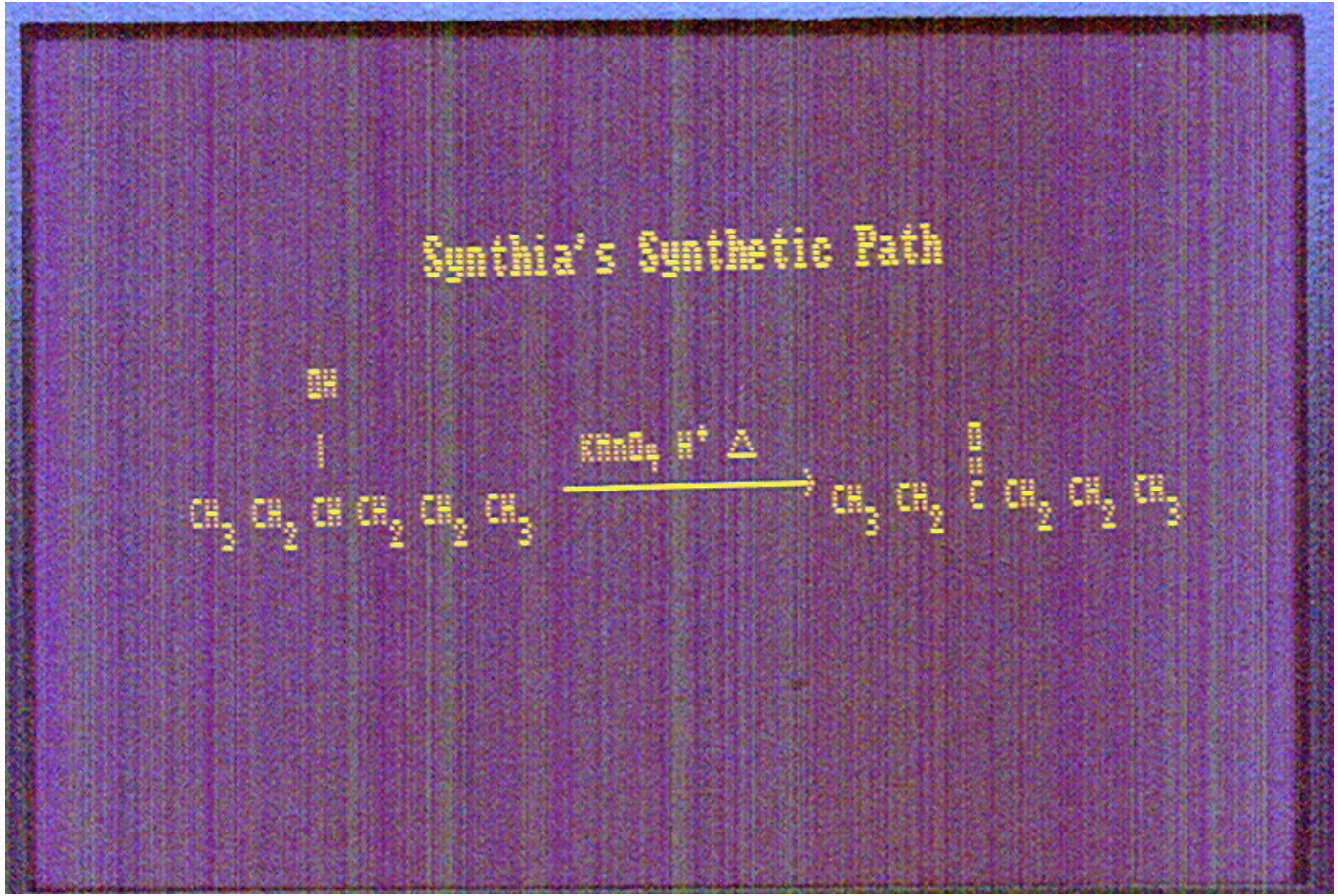
AI for Games, Millington & Funge: Figure 4.9

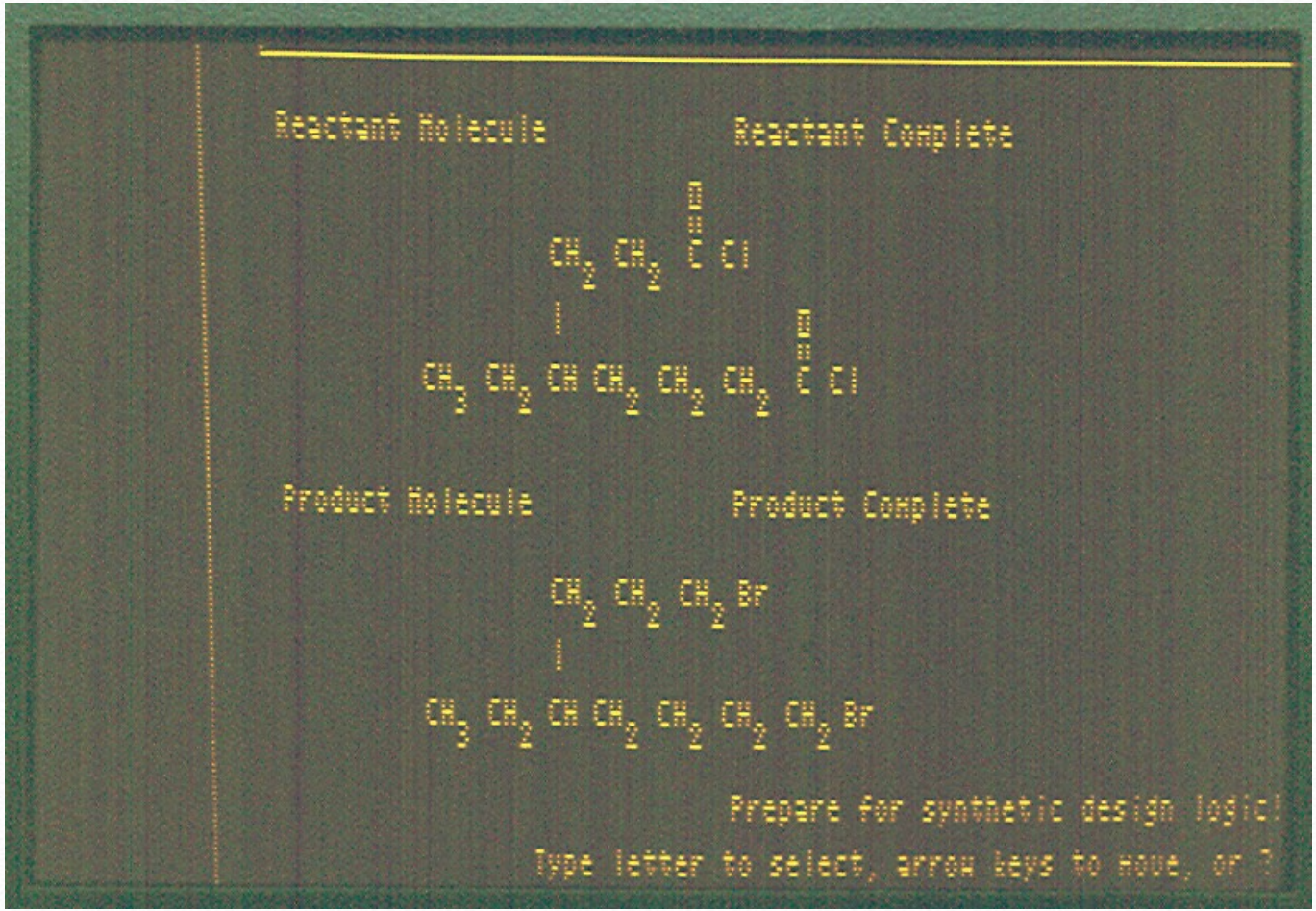


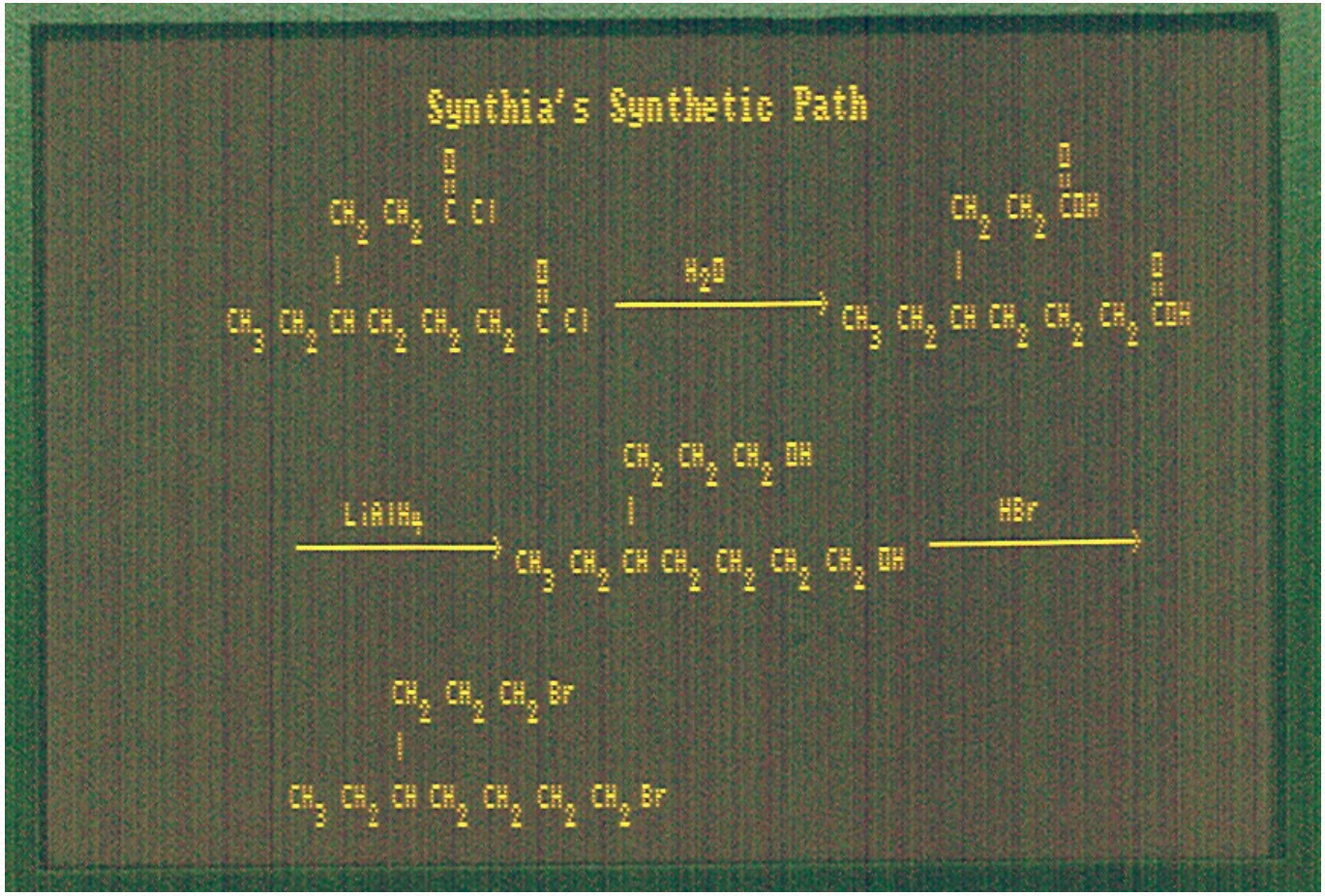
AI for Games, Millington & Funge: Figure 4.11


Anthony's Molecular Building Set
CH3 CH2 CH OH C CO COH CH (CH3)3C (CH3)2CH Cl Br Ph
A B C D E F G H I J K L M
Reactant Molecule
Product Molecule
Type letter to select, arrow keys to move, or ? for online help

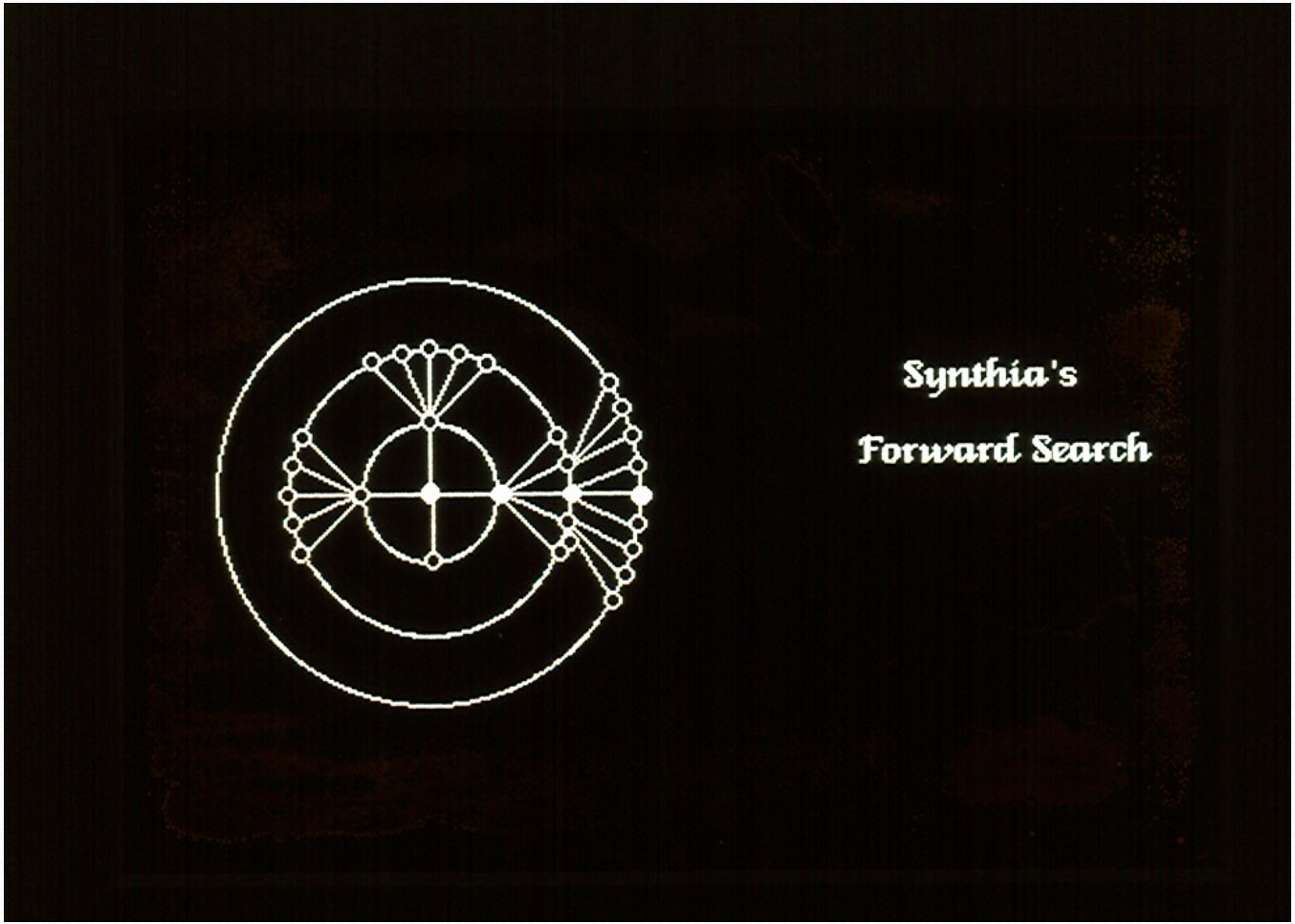

Anthony's Molecular Building Set
CH3 CH2 CH OH C=O CO COH CH (CH3)3C (CH3)2CH Cl Br Ph
A B C D E F G H I J K L M
Reactant Molecule
Reactant Complete
OH
CH3 CH2 CH CH2 CH2 CH3
Product Molecule
Product Complete
O
CH3 CH2 C CH2 CH2 CH3
Prepare for synthetic design logic!
Type letter to select, arrow keys to move, or ? for online help

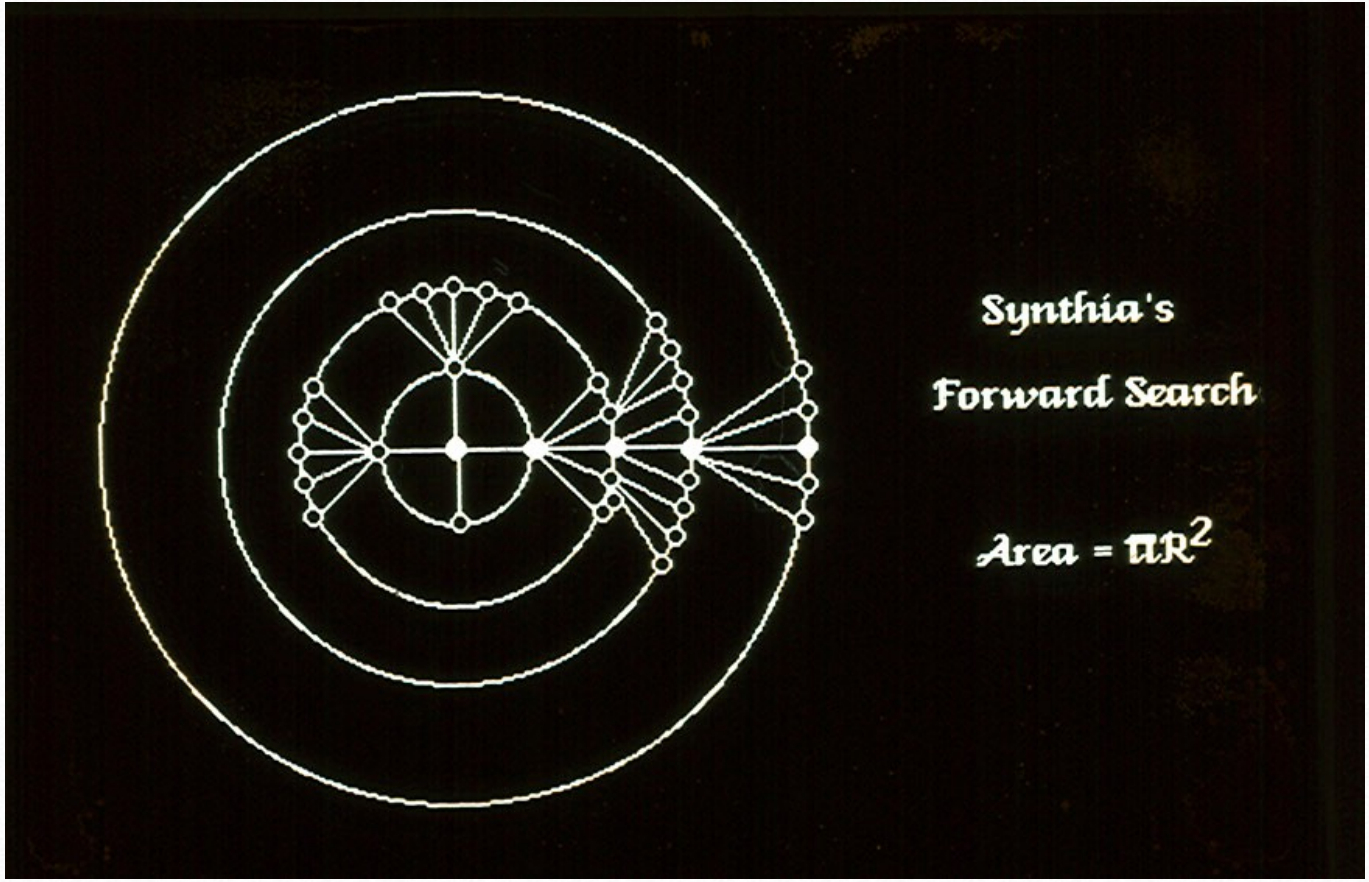

Synthia's Synthetic Path
OH
KMnO4 H+ Δ
O

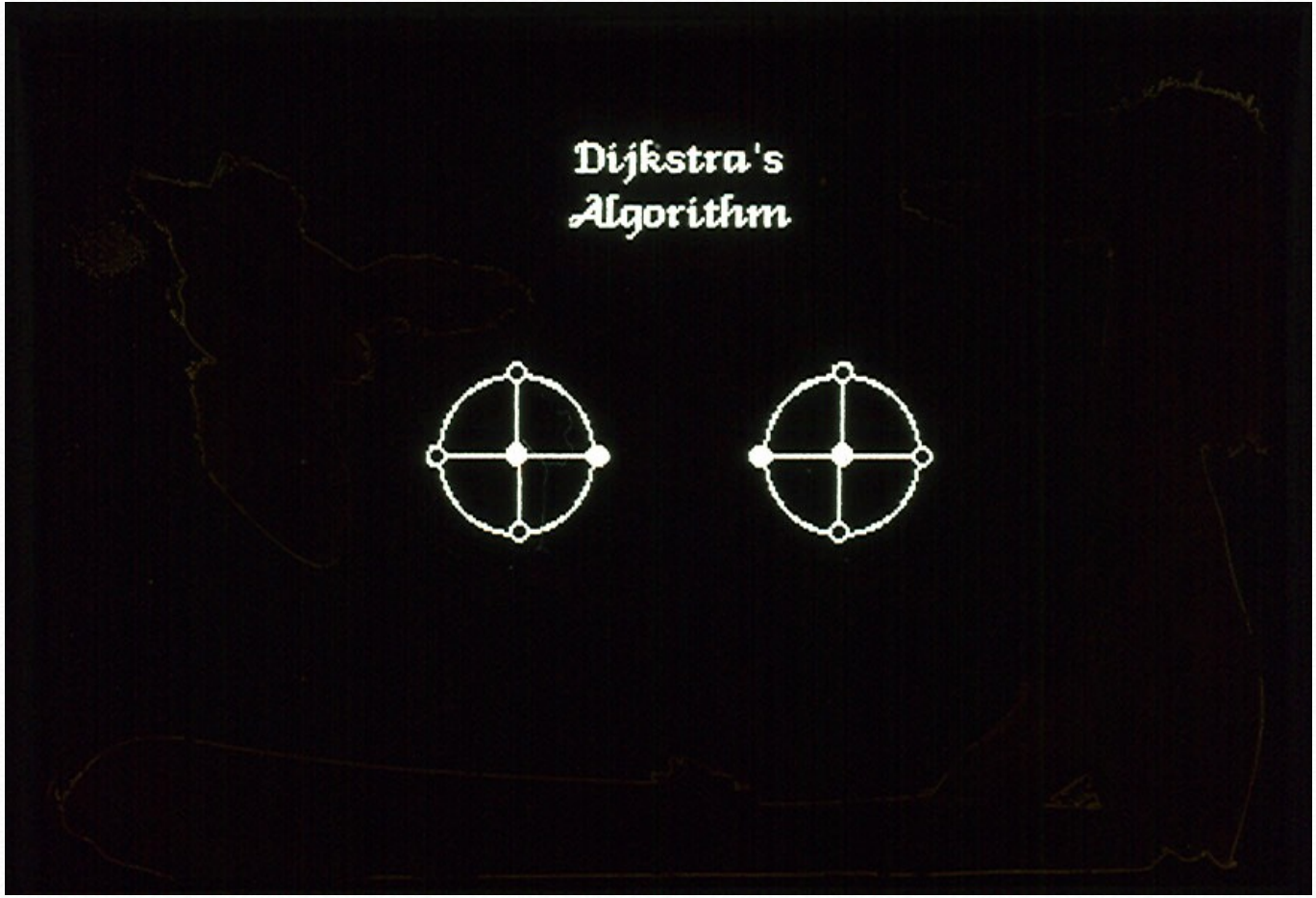

Reactant Molecule
Reactant Complete
Product Molecule
Product Complete
Prepare for synthetic design logic!
Type letter to select, arrow keys to move, or ?

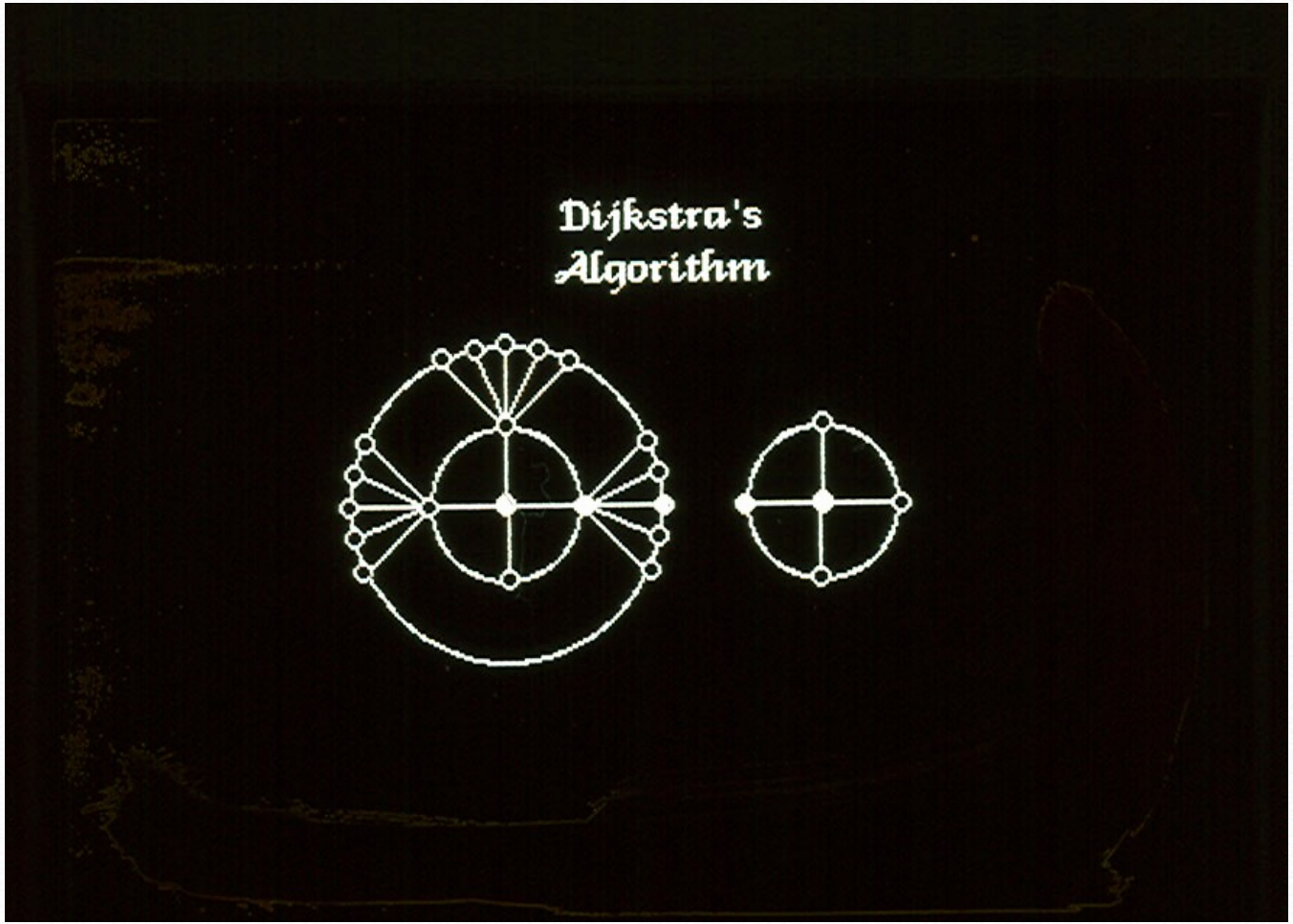

Synthia's Synthetic Path
H2O
LiAlH4
HBr

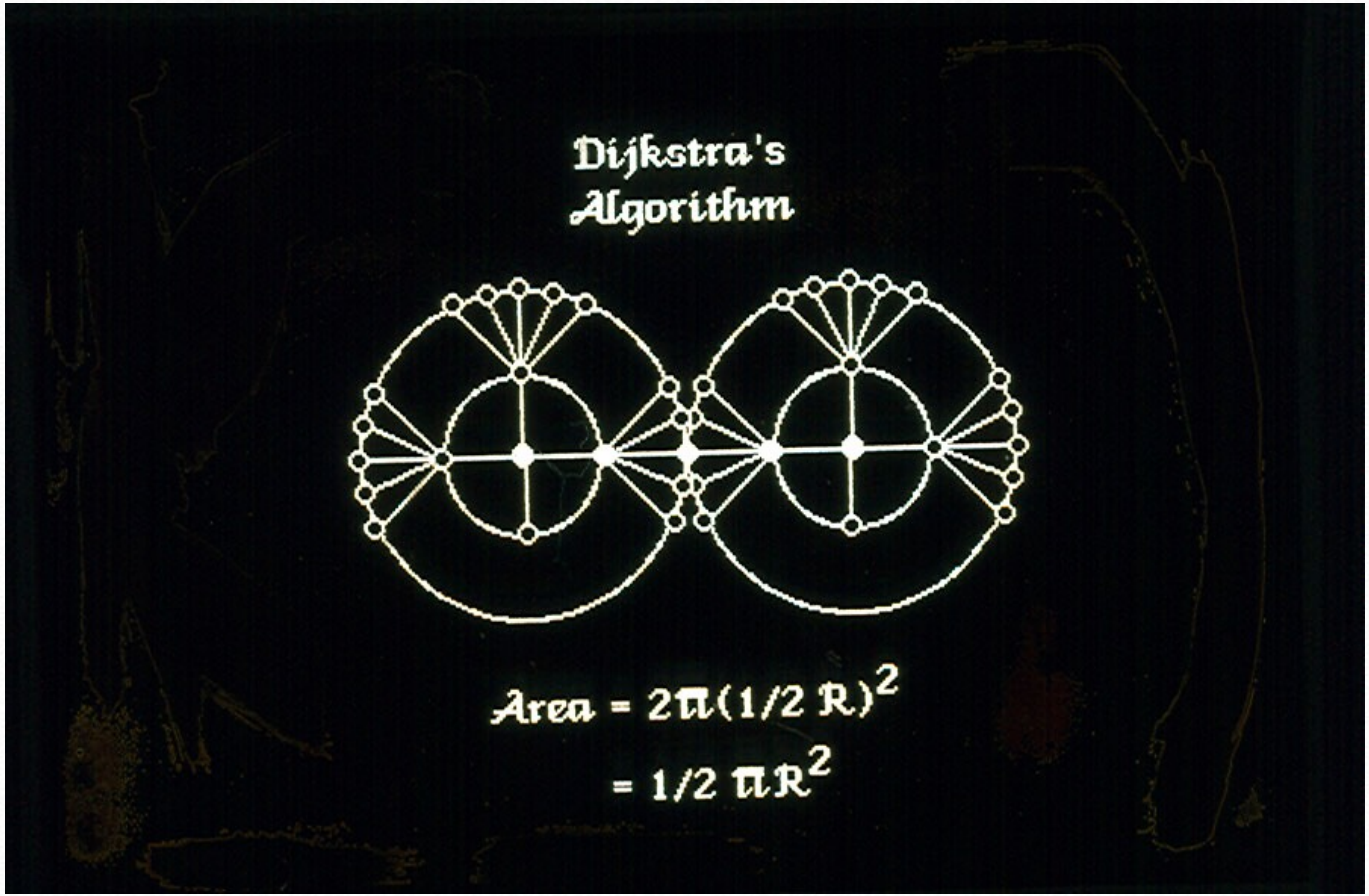

Synthia's
Forward Search

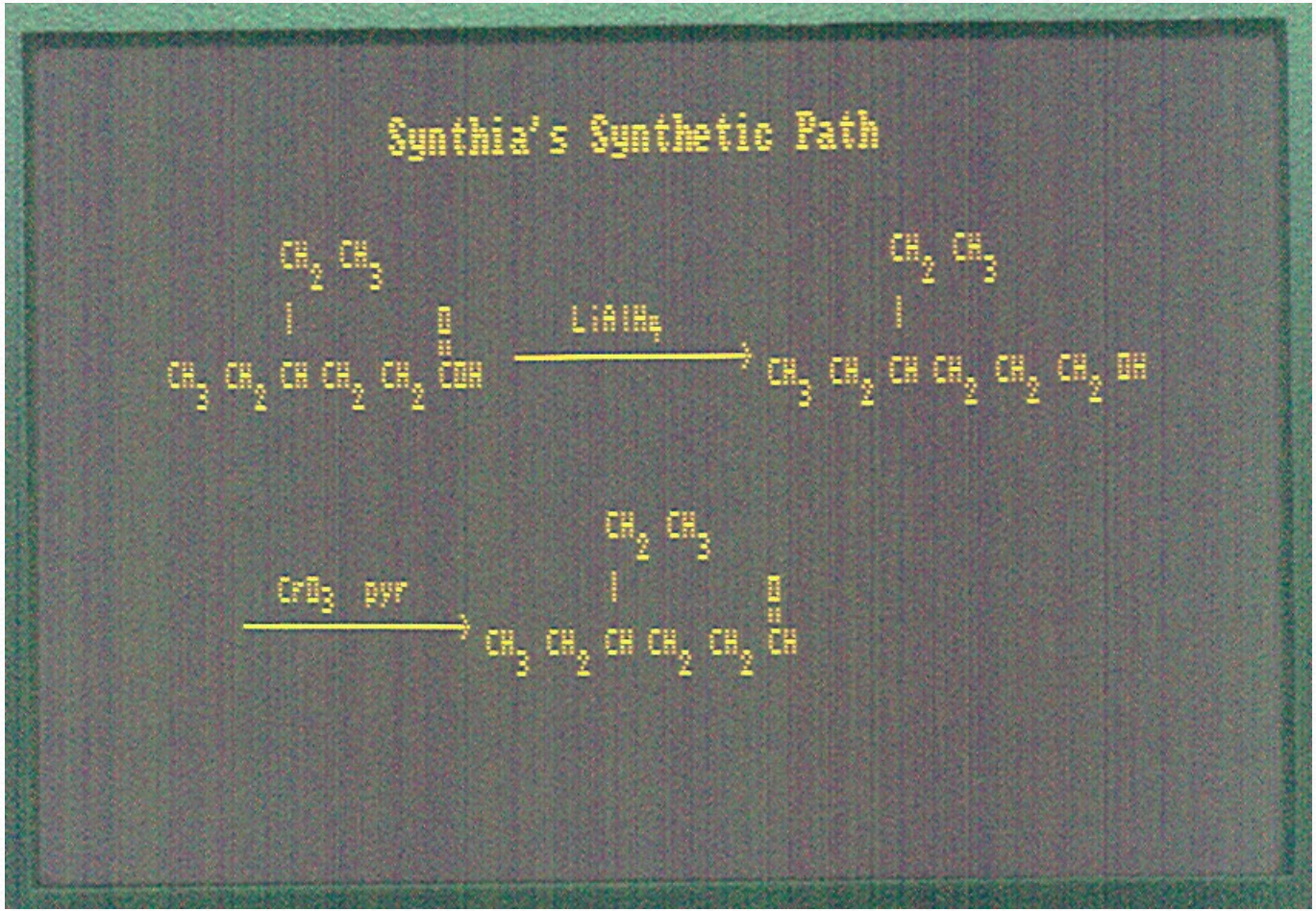

Synthia's
Forward Search

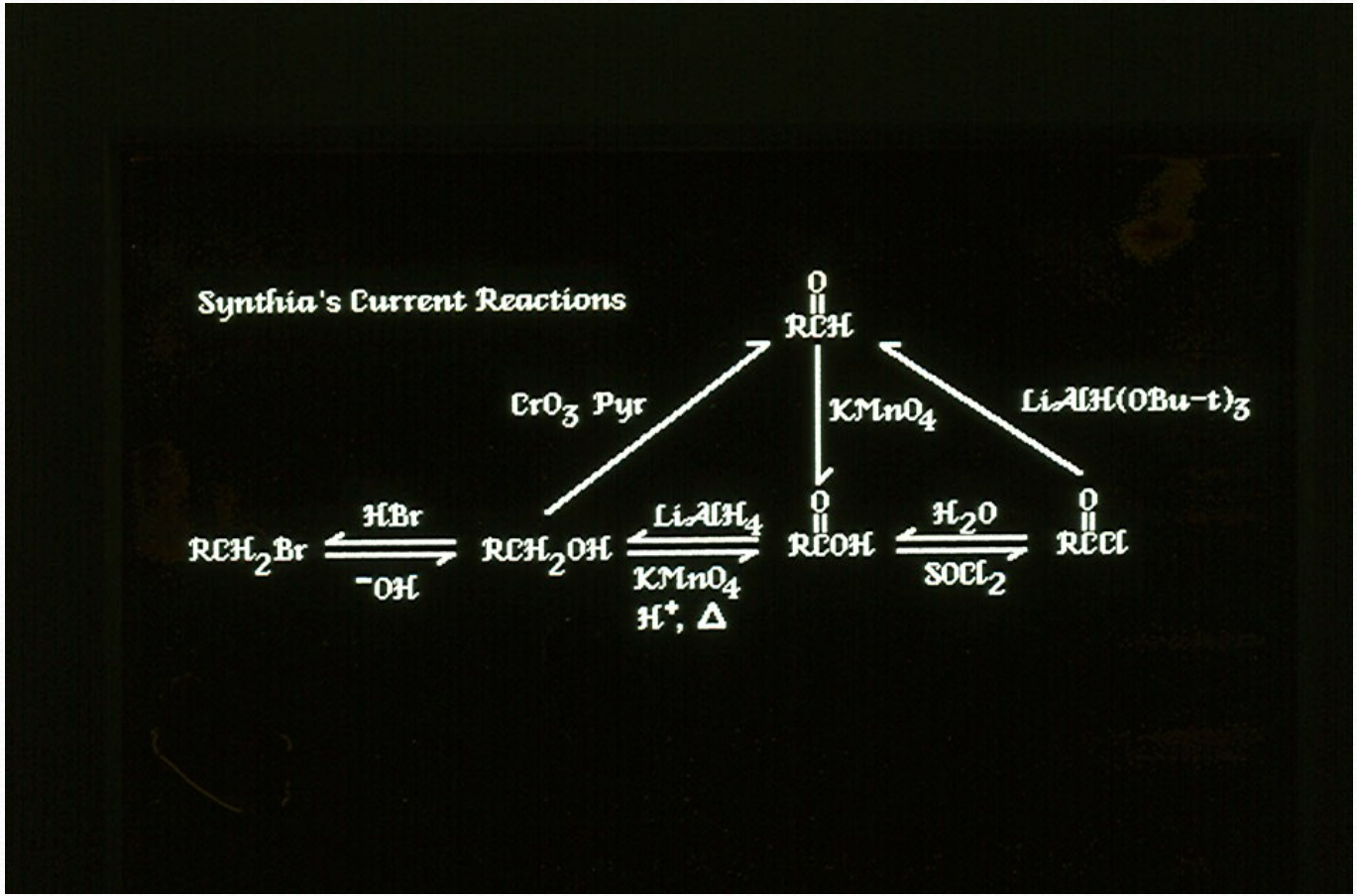

Synthia's
Forward Search
Area = πR²

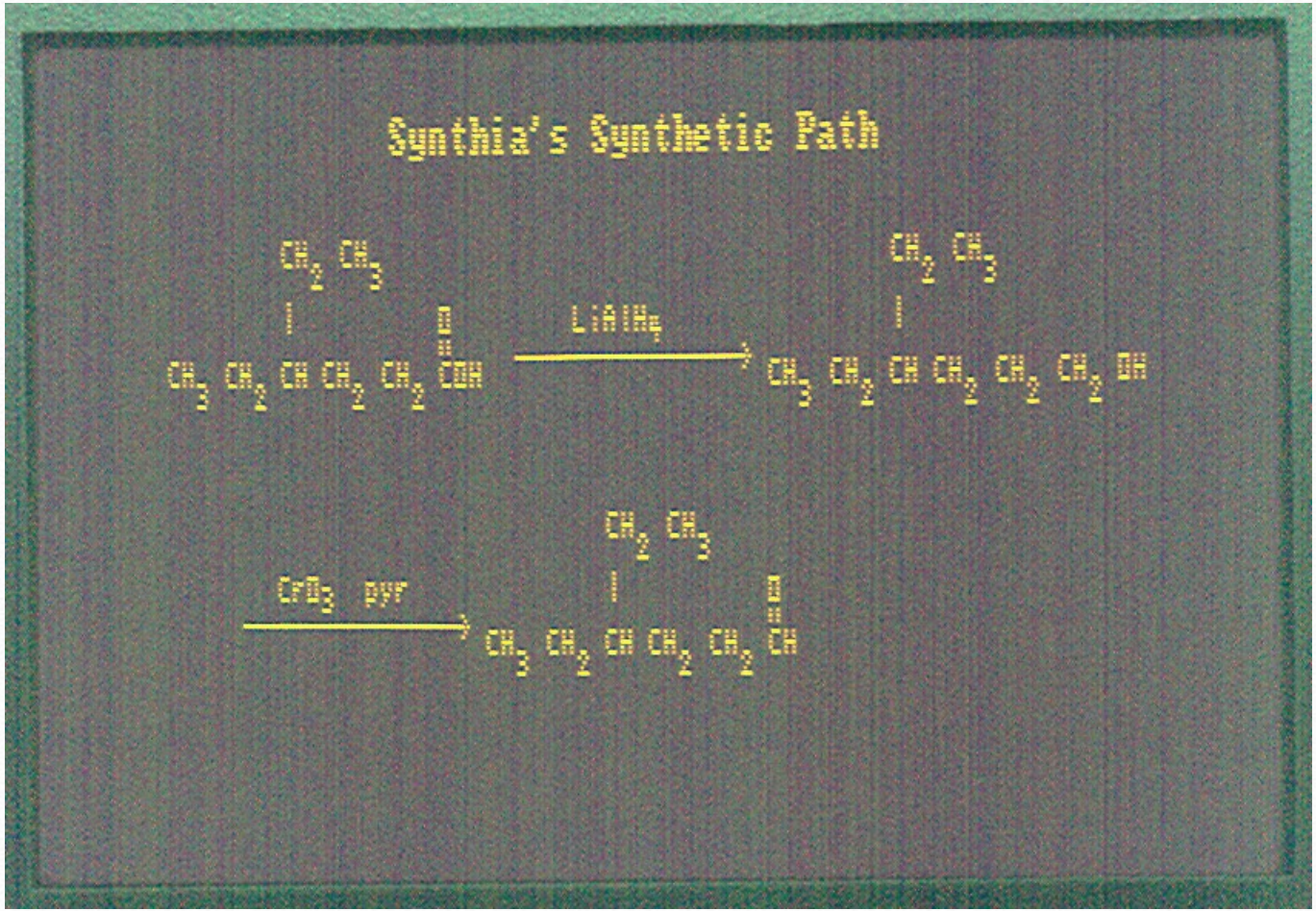

Dijkstra's
Algorithm

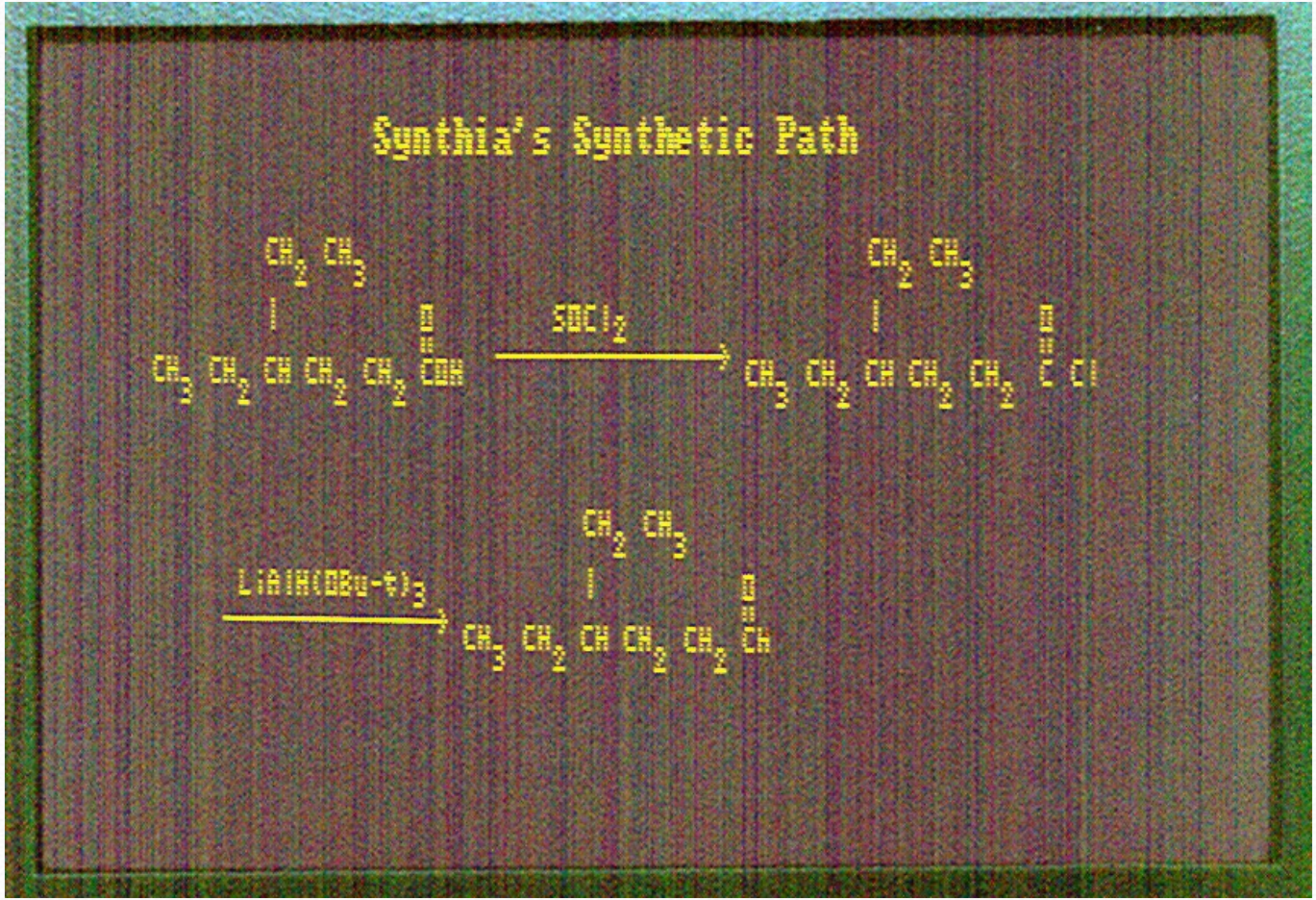

Dijkstra's
Algorithm

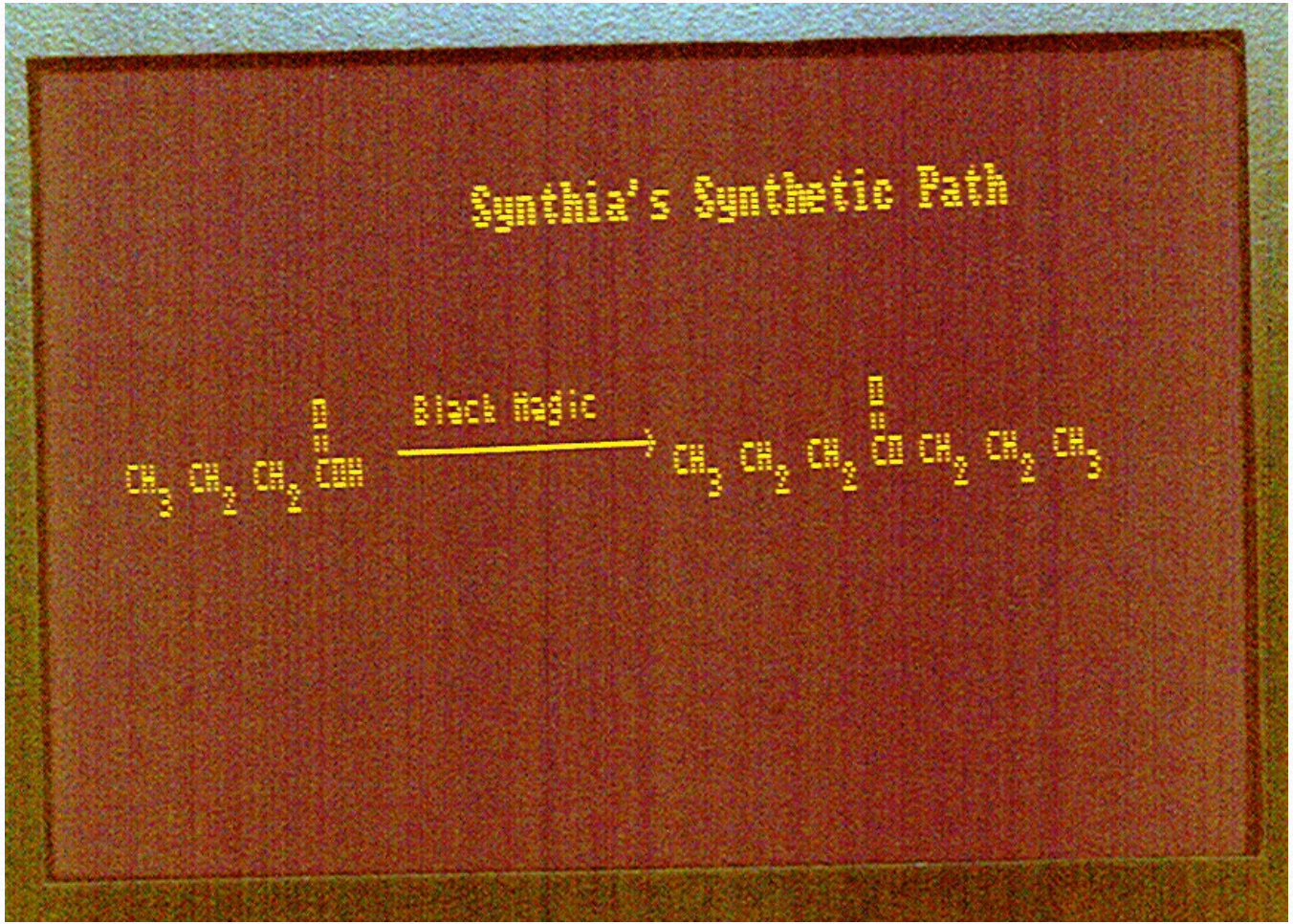

Dijkstra's
Algorithm
Area = 2π(1/2 R)²
= 1/2 πR²

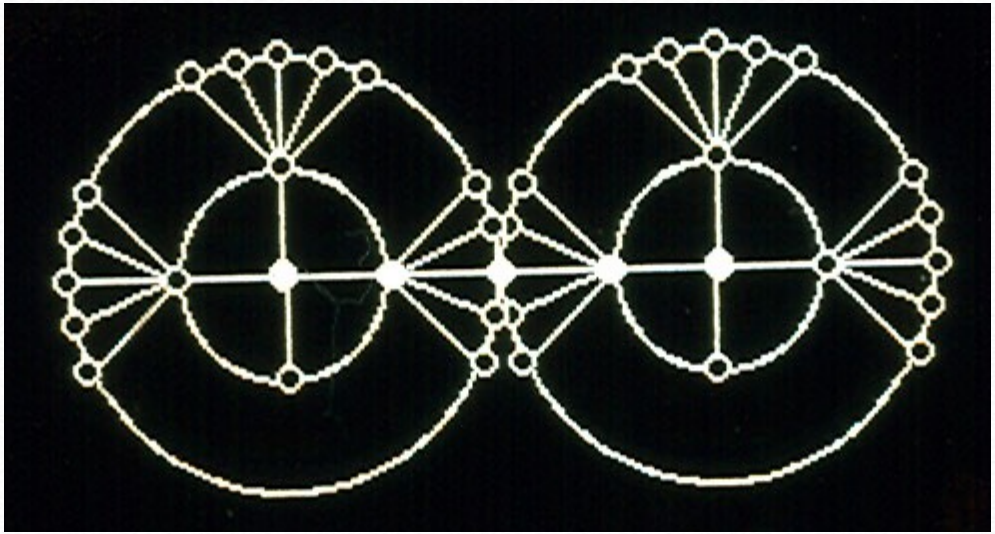

Synthia's Synthetic Path
LiAlH4
CrO3 pyr

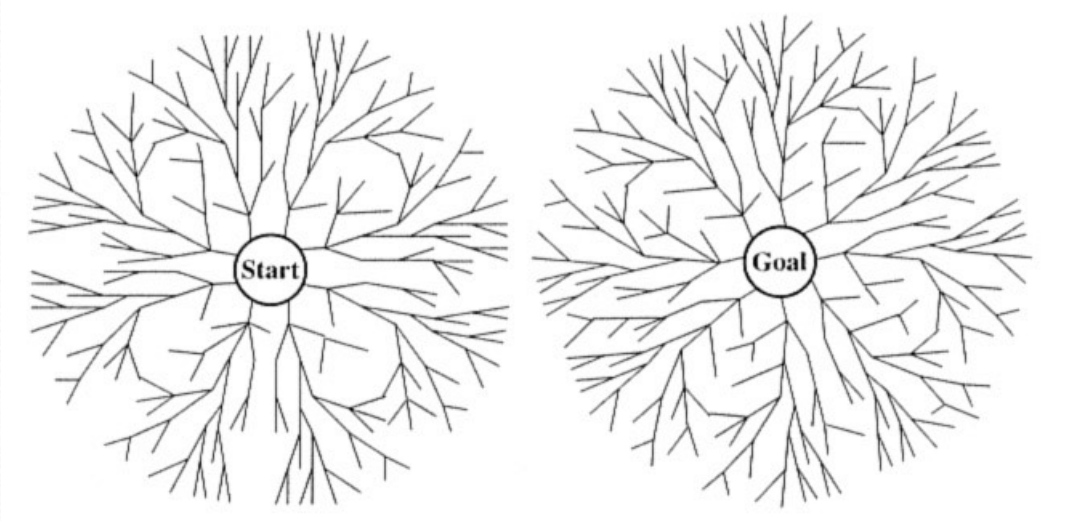

Synthia's Current Reactions
RCH
CrO3 Pyr
KMnO4
LiAlH(OBu-t)3
RCH2Br
HBr
-OH
RCH2OH
LiAlH4
KMnO4
H+, Δ
RCOH
H2O
SOCl2
RCCl


Synthia's Synthetic Path
LiAlH4
CrO3 pyr


Synthia's Synthetic Path
CH2 CH3
CH3 CH2 CH CH2 CH2 COH
SOCl2
CH2 CH3
CH3 CH2 CH CH2 CH2 C Cl
LiAlH(OBu-t)3
CH2 CH3
CH3 CH2 CH CH2 CH2 CH


Synthia's Synthetic Path
CH3 CH2 CH2 COH
Black Magic
CH3 CH2 CH2 CO CH2 CH2 CH3

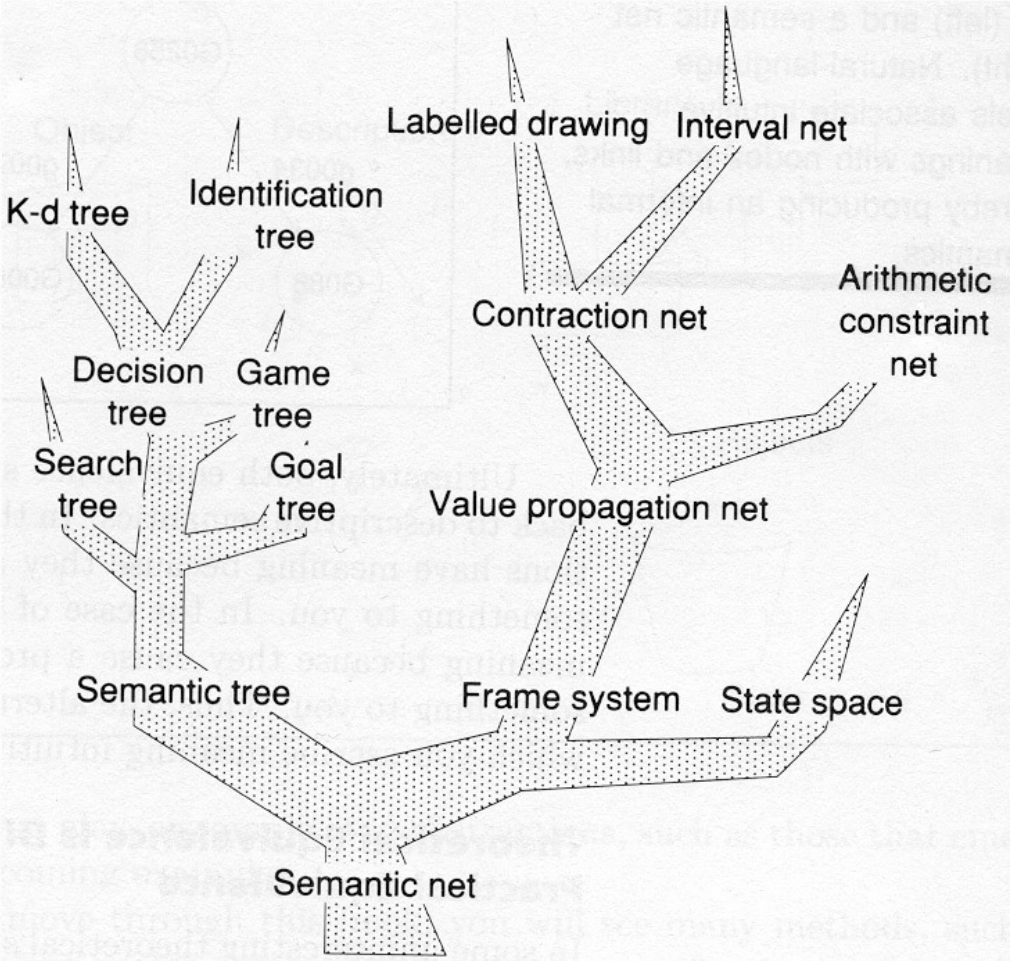




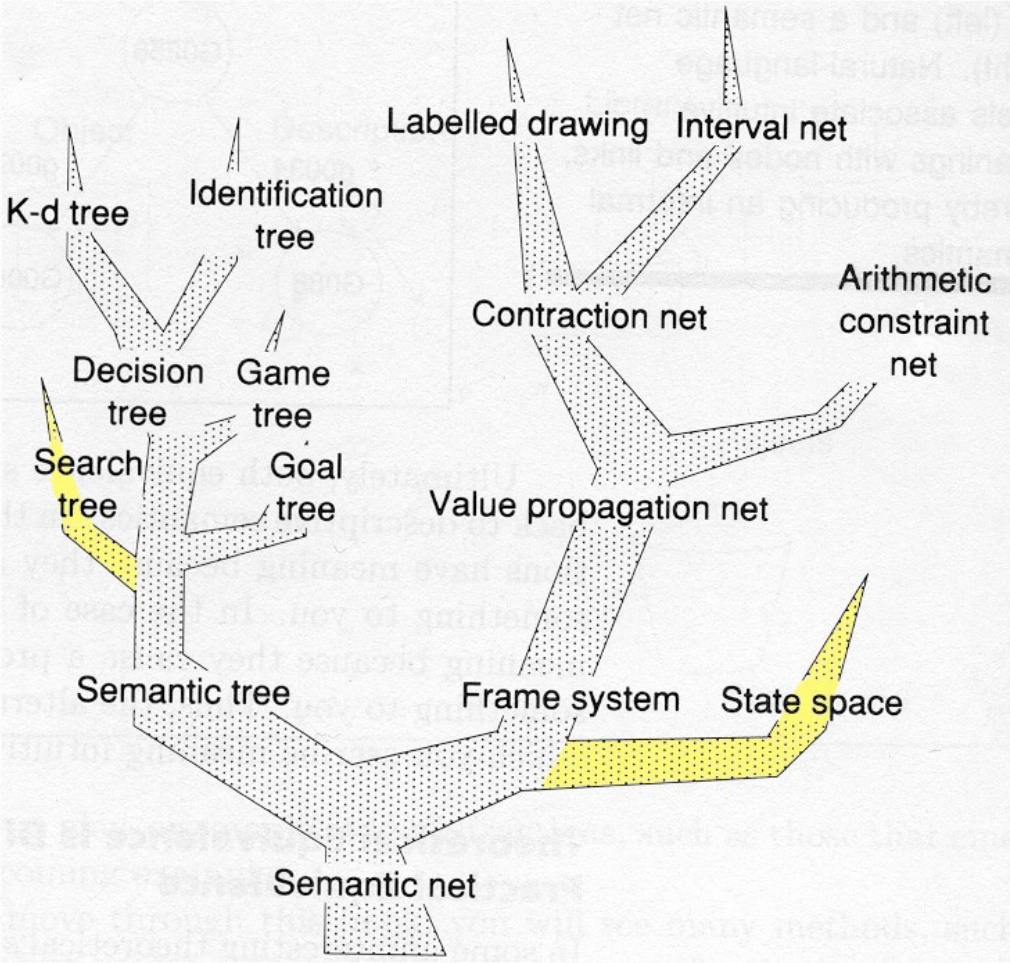
Time: $O(b^{d/2})$

Space: $O(b^{d/2})$

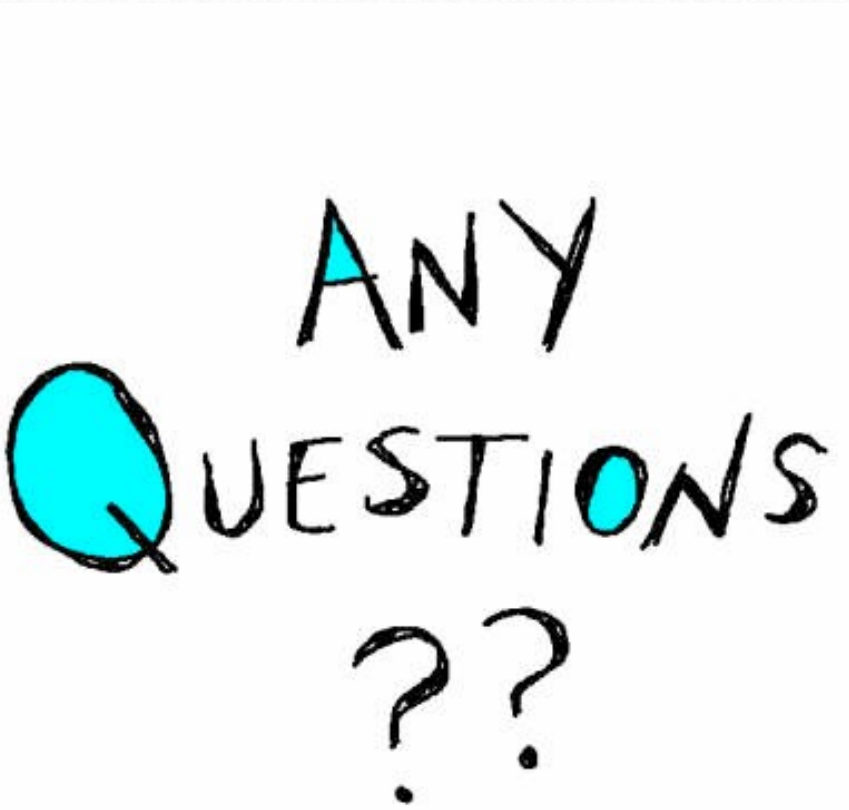
[Russell & Norvig 2003]



[Winston 1993]



[Winston 1993]


ANY
QUESTIONS
??

Study Questions

Solve the Missionaries and Cannibals problem.

What is the major difference between implementations of breadth-first and depth-first search?

Solve the Water Jug problem by completing the graph until the goal is reached.