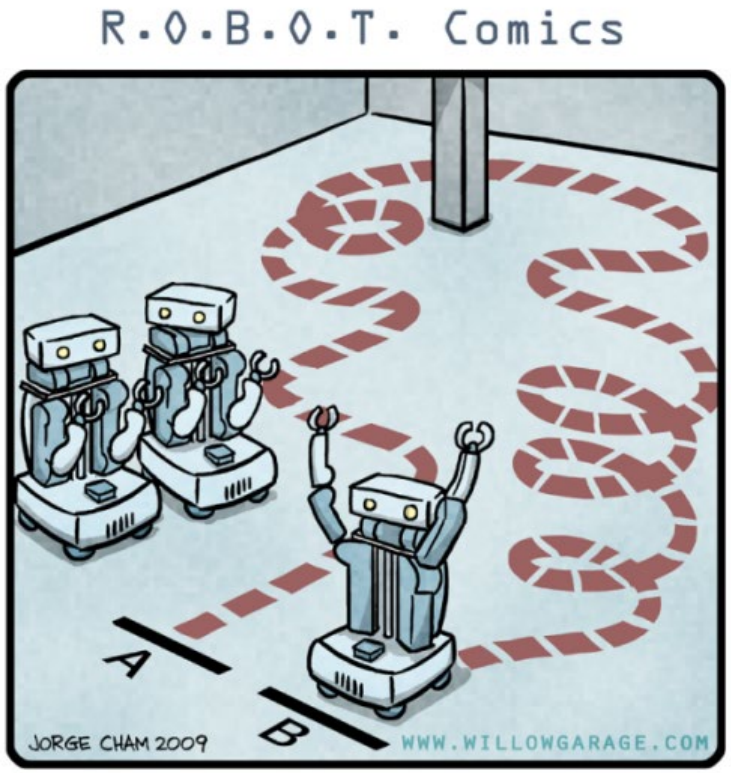


Robot Intelligence: Planning

CS 4649/7649 – Asst. Prof. Matthew Gombolay

Slides adapted from:
6.034 Tomas Lozano Perez
16.410 Brian Williams
Russell and Norvig AIMA

Assignments

- Due Today, 8/19
  - Read Ch. 3 in Russel & Norvig
- Due Monday, 8/24
  - Read Ch. 4 in Russel & Norvig
- Due Wednesday, 8/26
  - Pset1 (released on Canvas by 3:15 pm on Wednesday, 8/19)

Research Opportunity

Outline:

Problem Solving as State Space Search

- Problem Formulation (Modeling)
  - Problem solving as state space search
- Formal Representation
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

Search is the basic building block

Operations/Task Planning:

- Activity Planning
- Diagnosis
- Repair
- Scheduling
- Resource Allocation

Mobility/Motion Planning:

- Path Planning
- Localization
- Map Building
- Control Trajectory Design

Example: Outpost Logistics Planning

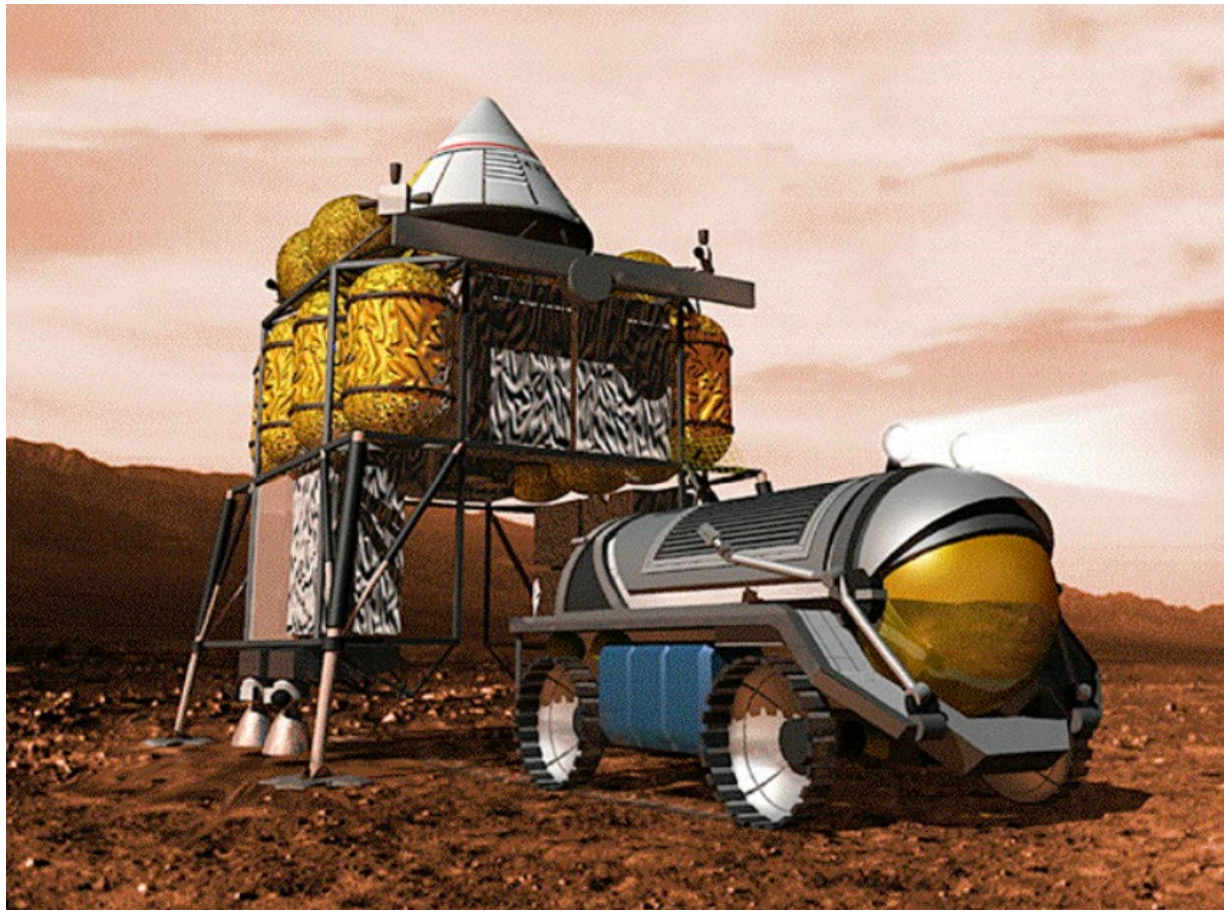
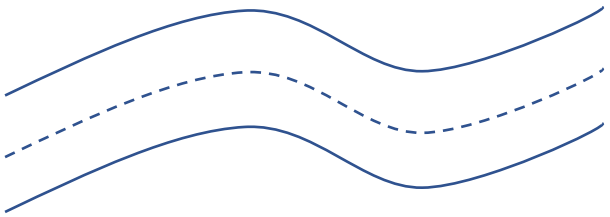


Image produced for NASA by John Frassanito and Associates

Example: Outpost Logistics Planning

- Can the astronaut get its supplies safely across a Martian crevasse?

Astronaut
Goose
Grain
Fox



Rover

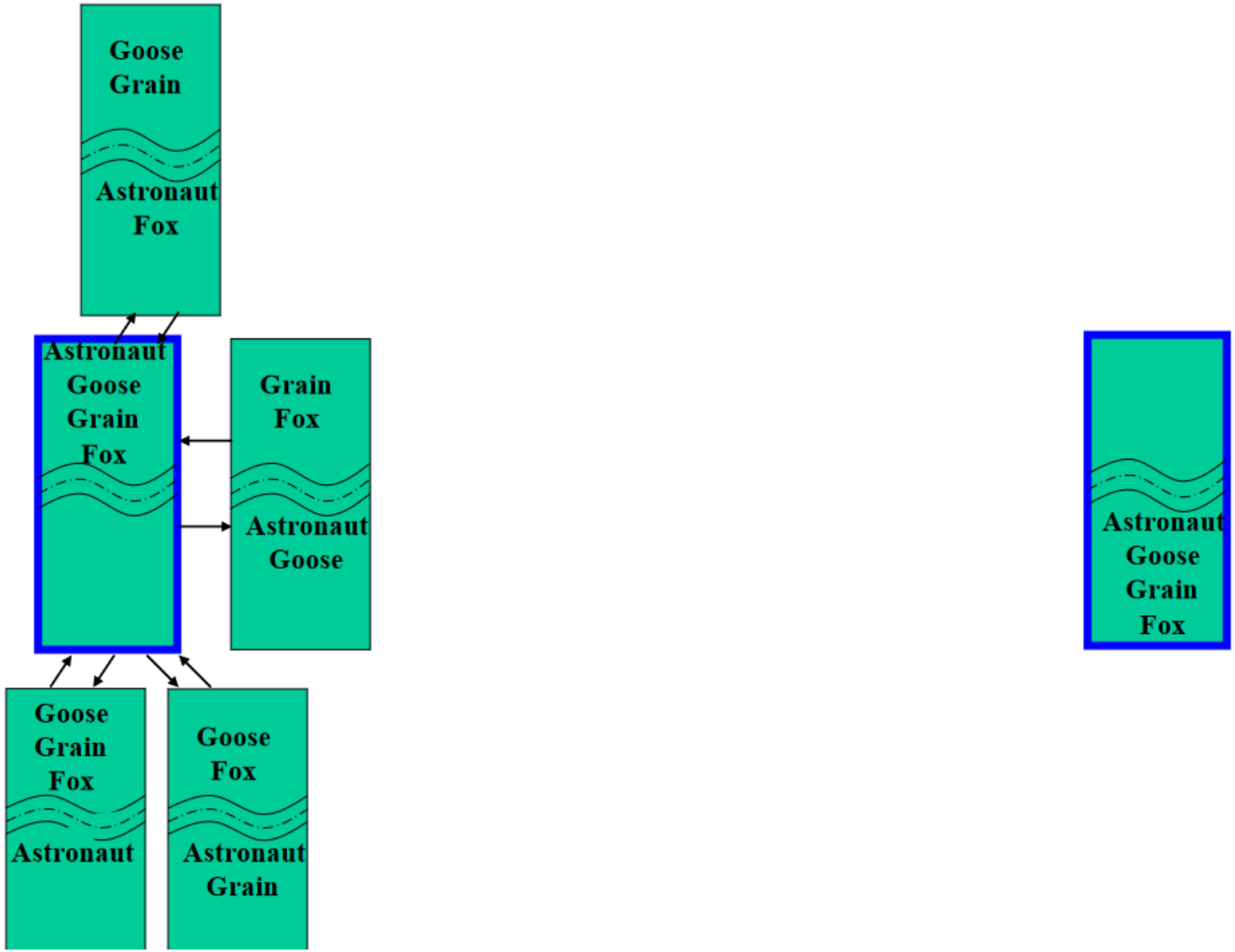
- Astronaut + 1 item allowed in the rover
- Goose alone eats Grain
- Fox alone eats Goose

- Early AI: What are the universal problem solving methods?

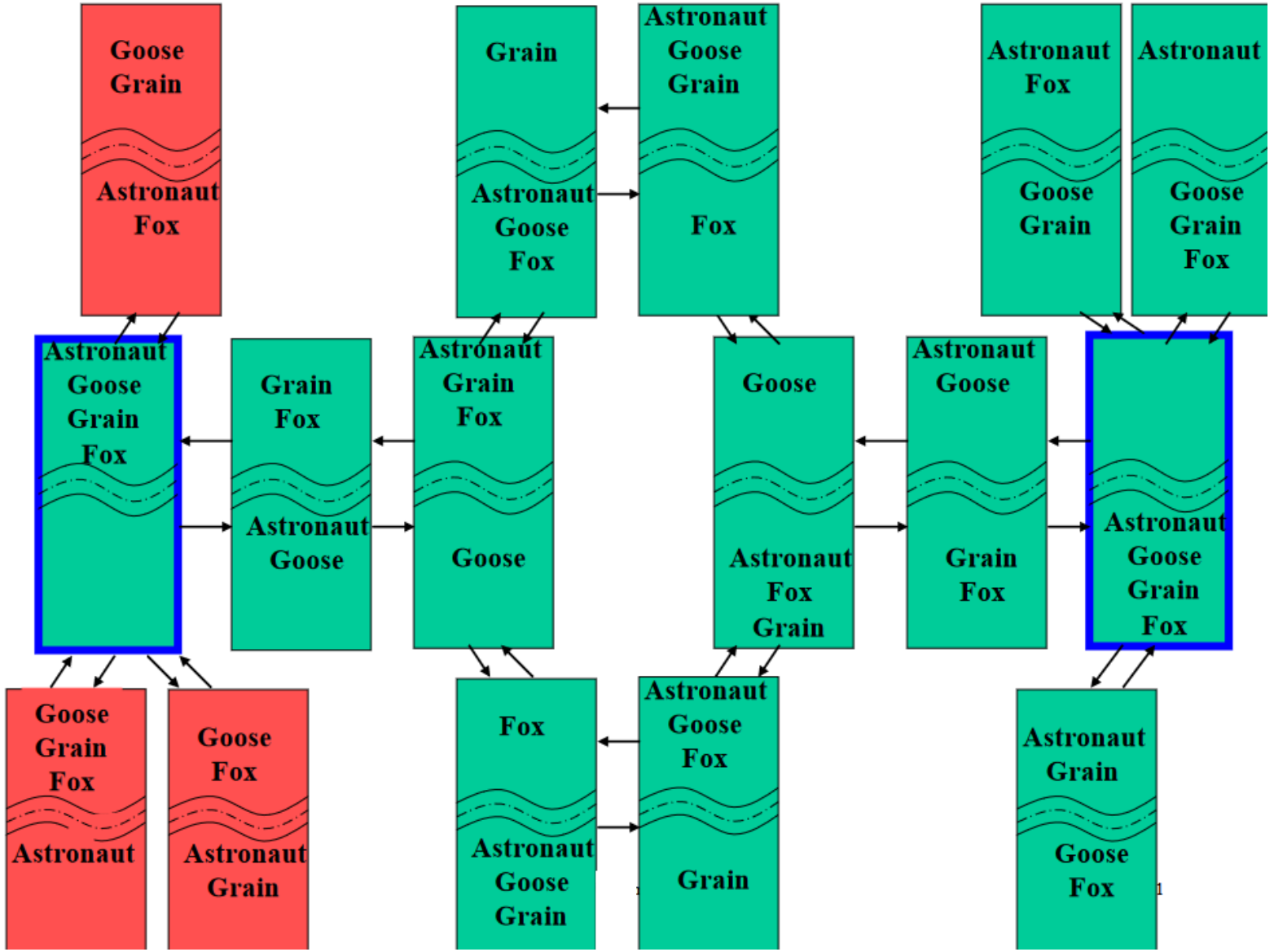
Simple → Trivial

Problem Solving as State Space Search

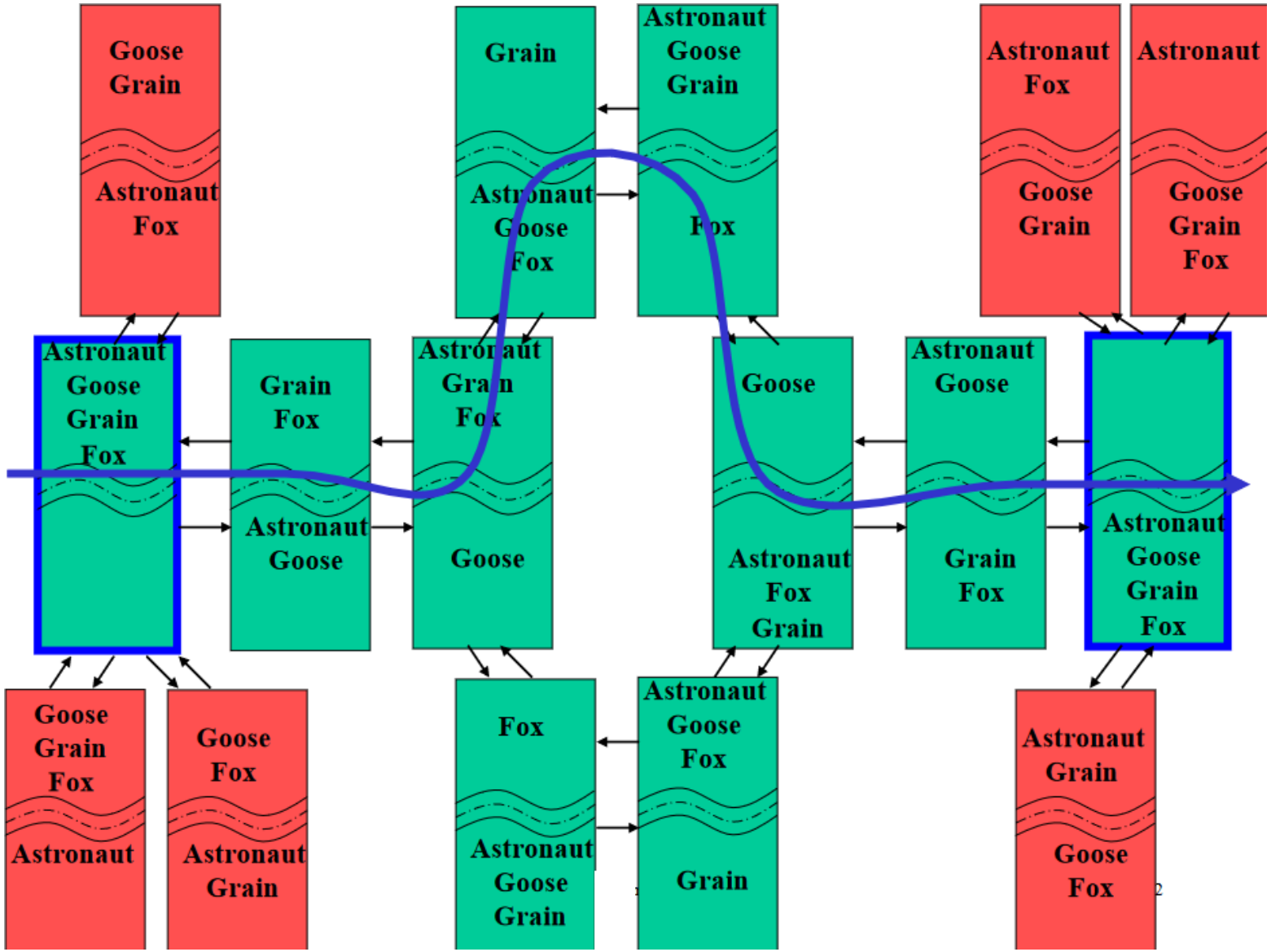
- Formulate Goal
  - State
    - Astronaut, Fox, Goose, & Grain below crevasse
- Formulate Problem
  - States
    - Astronaut, Fox, Goose, & Grain above or below the crevasse
  - Operators
    - Move: Astronaut drives rover and 1 or 0 items to the other side of the crevasse
  - Initial State
    - Astronaut, Fox, Goose, & Grain above crevasse
- Generate Solution
  - Sequence of Operators (or States)
    - Move(goose,astronaut), Move(astronaut), …



Credit: Brian Williams



Credit: Brian Williams



Credit: Brian Williams

Formulation Example: 8-Puzzle

Start

Goal

- States: integer location for each tile AND …???
- Operators: move empty square up, down, left, right
- Initial and Goal States: as shown above

Languages for Express States and Operators for Complex Tasks

Image Source: NASA

Example: STRIPS Planning Language

Stanford **R**esearch **I**nstitute **P**roblem **S**olver

Initial State:

(and (hose a)
(clamp b)
(hydroxide-unit c)
(on-table a)
(on-table b)
(on-table c)
(clear a)
(clear b)
(clear c)
(empty arm))

Goal (Partial) State:

(and (connected a b)
(connected b c))

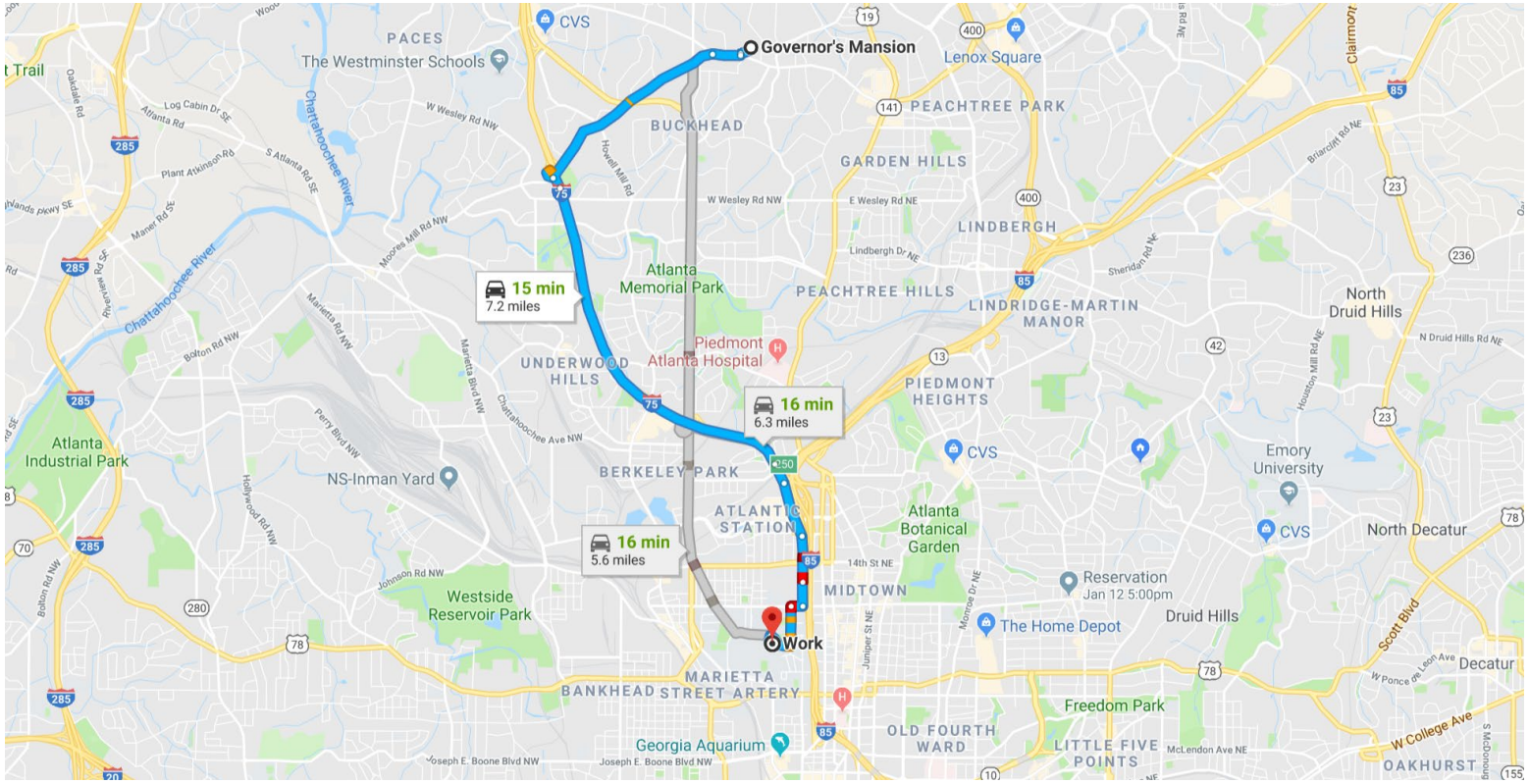
Operators:

Precondition: (and (clear hose)
(on-table hose)
(empty arm))

Pickup hose

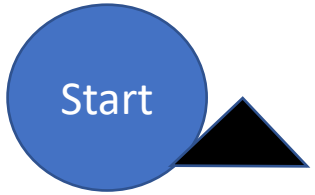
Effect: (and (not (clear hose))
(not (on-table hose))
(not (empty arm))
(holding arm hose))

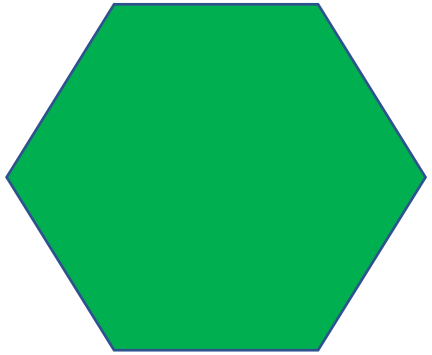
Problem: Find a route from home to GaTech



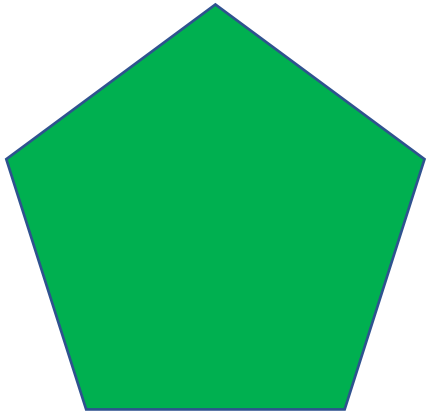
States? Operators? Initial and Goal State?

How do we map path planning to state space search?

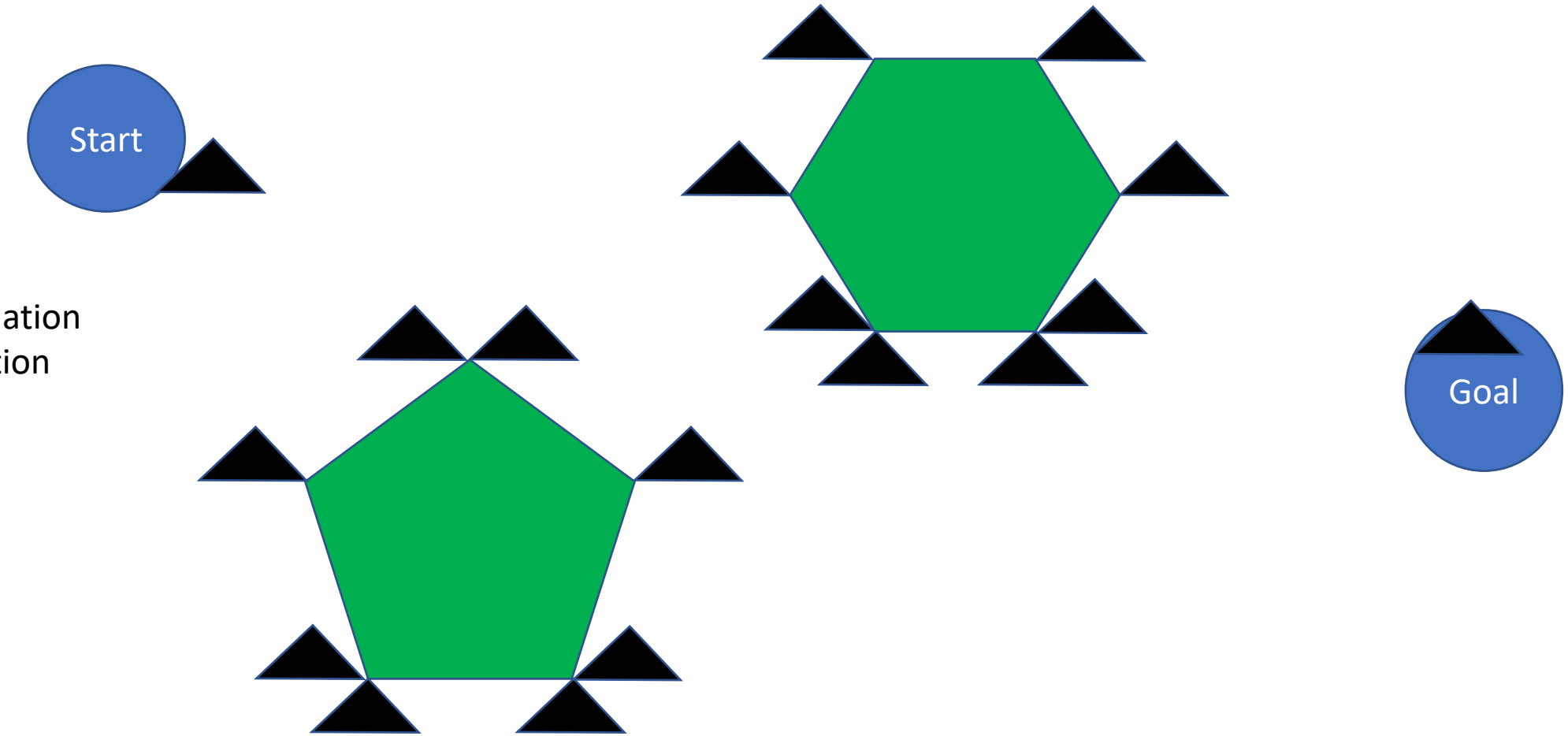




Assume translation
but no rotation

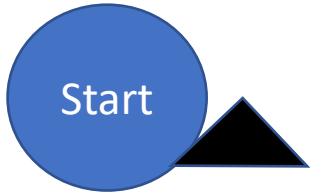


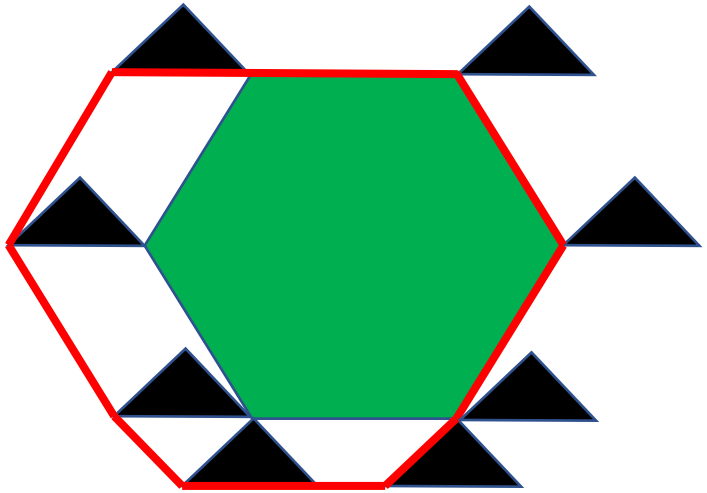
How do we map path planning to state space search?



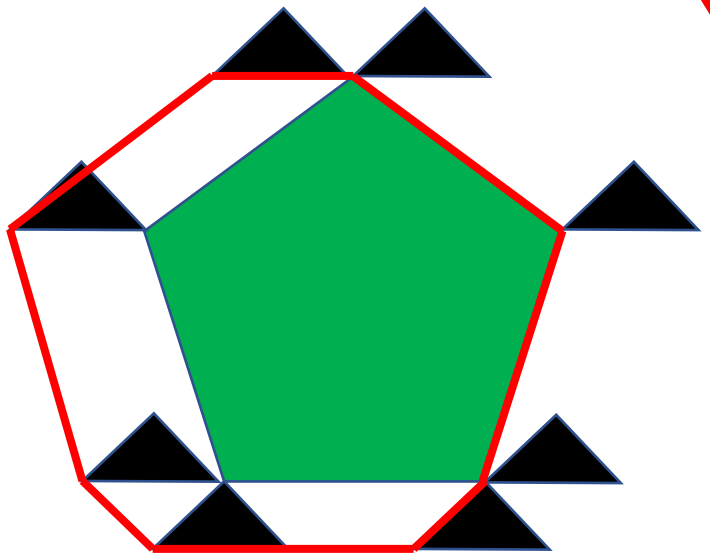
Assume translation but no rotation

Create Configuration Space



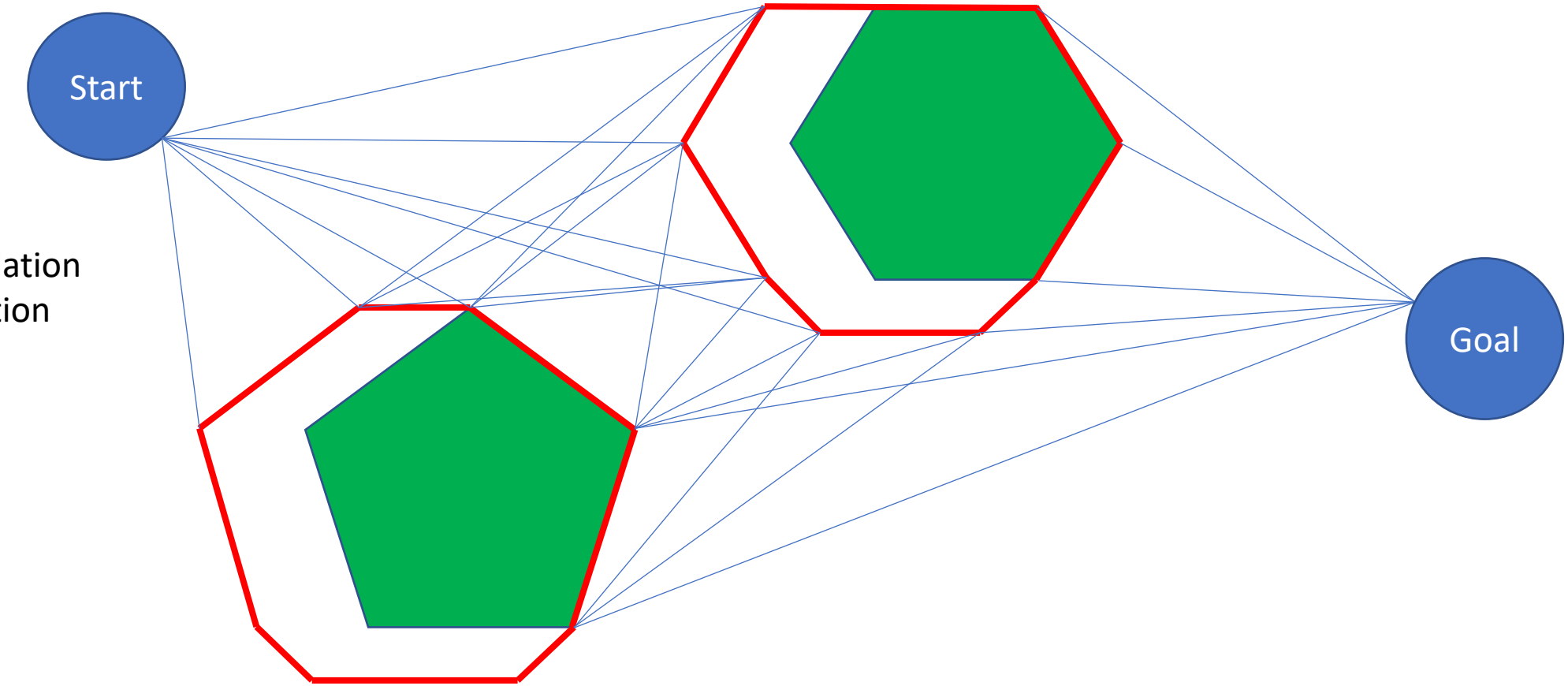


Assume translation
but no rotation



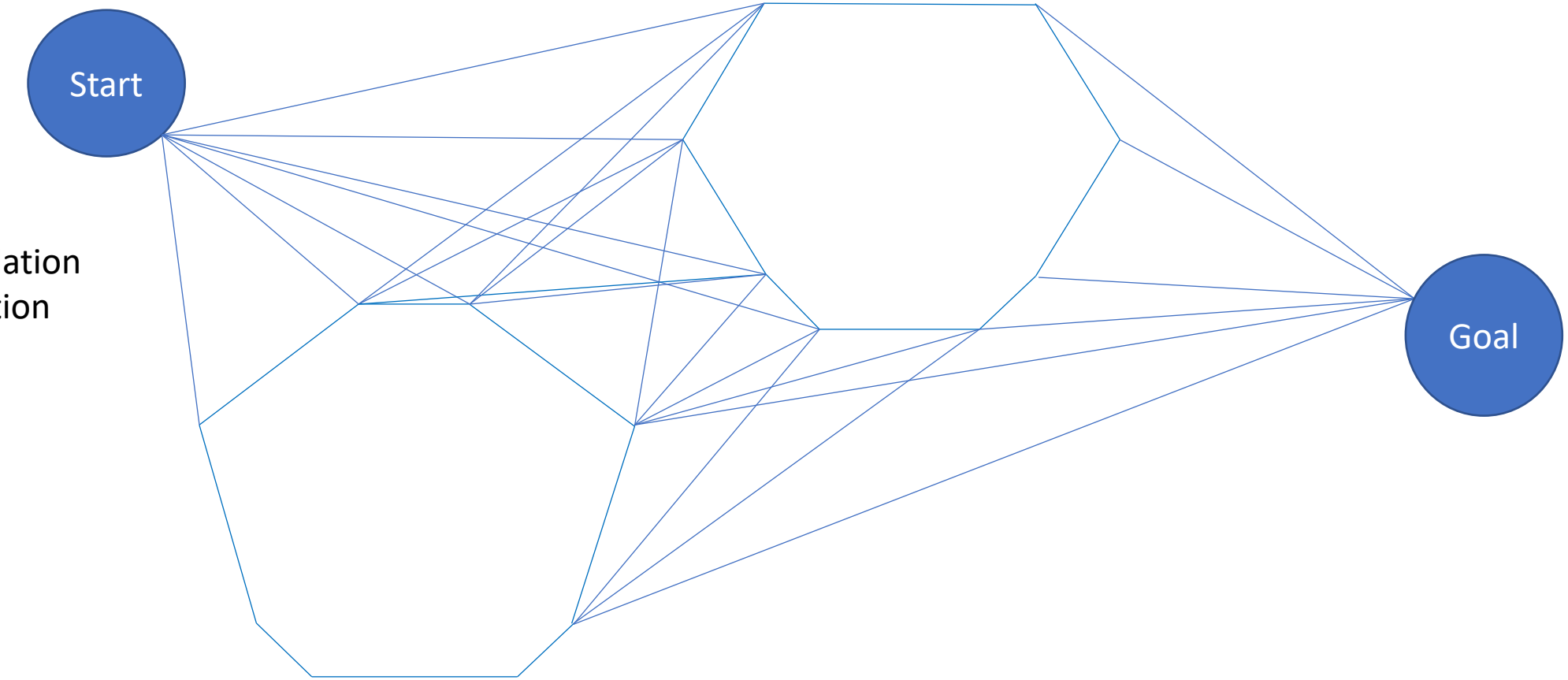
Map From Continuous Problem to Graph Search: Create Visibility Graph

Assume translation but no rotation



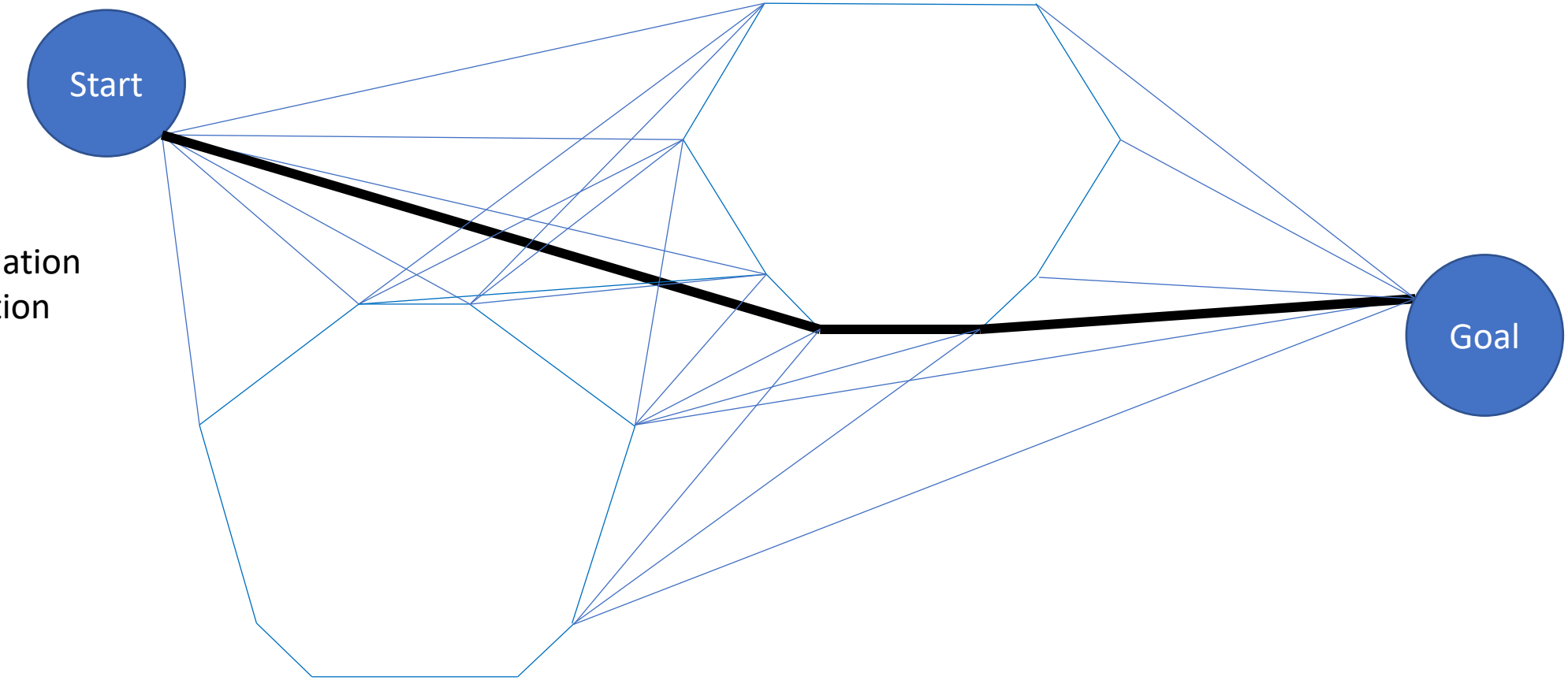
Map From Continuous Problem to Graph Search: Create Visibility Graph

Assume translation but no rotation



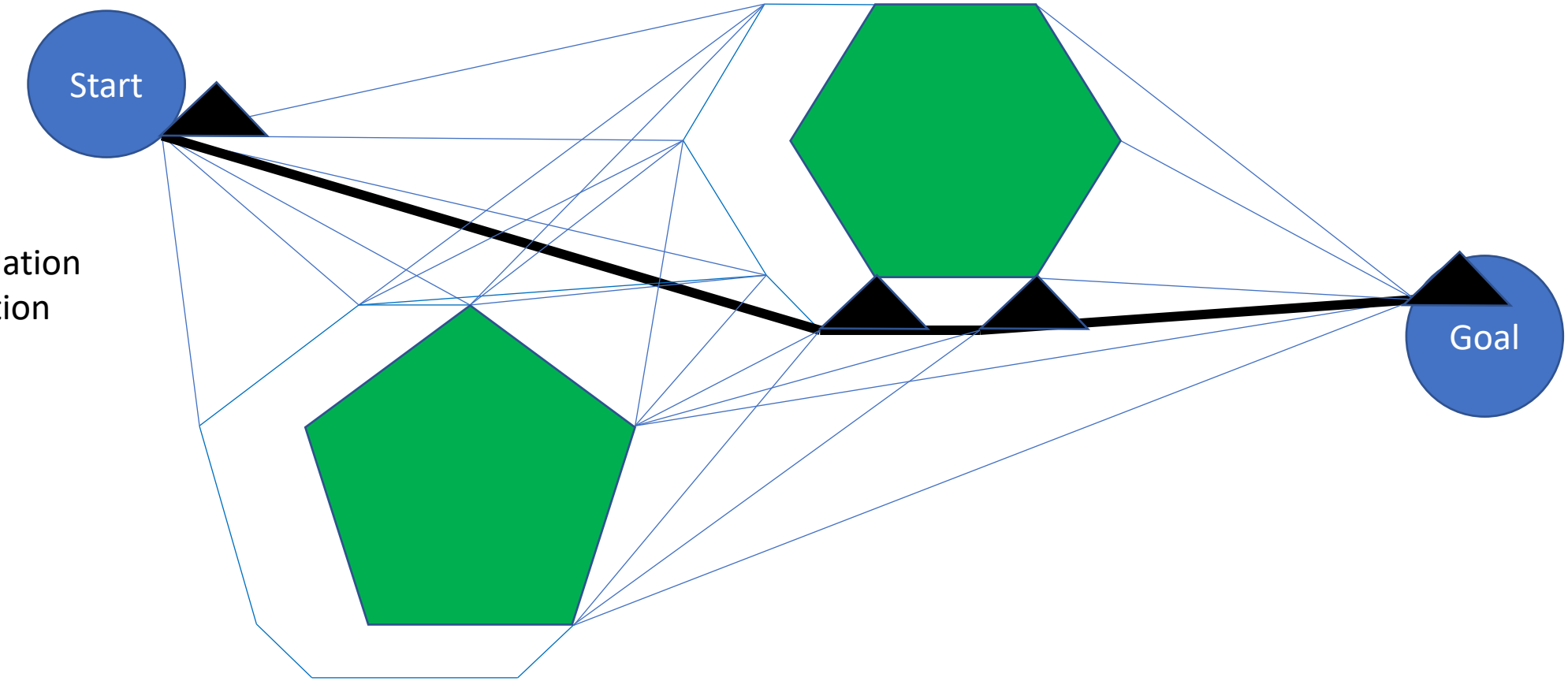
Finding Shortest Path (e.g., A*)

Assume translation
but no rotation



Finding Shortest Path (e.g., A*)

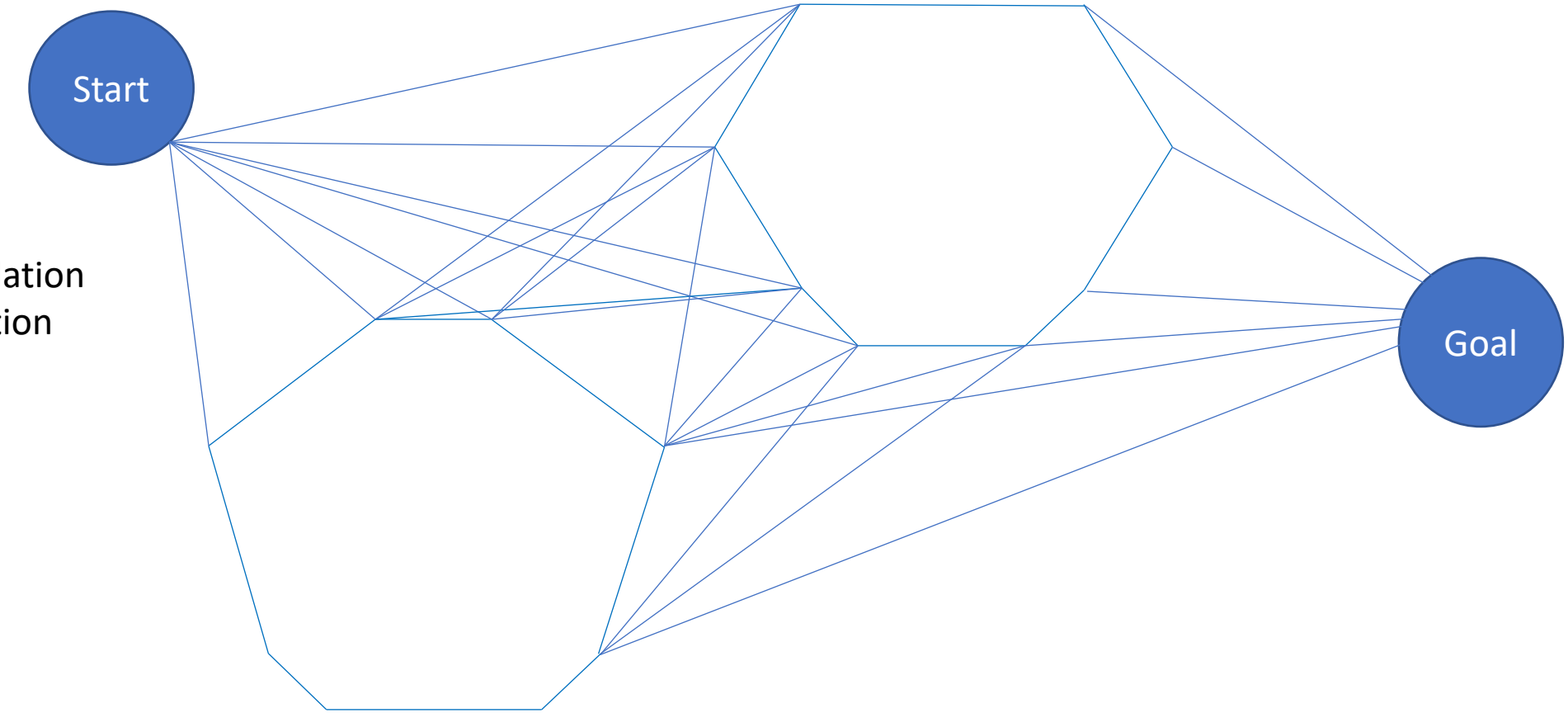
Assume translation
but no rotation



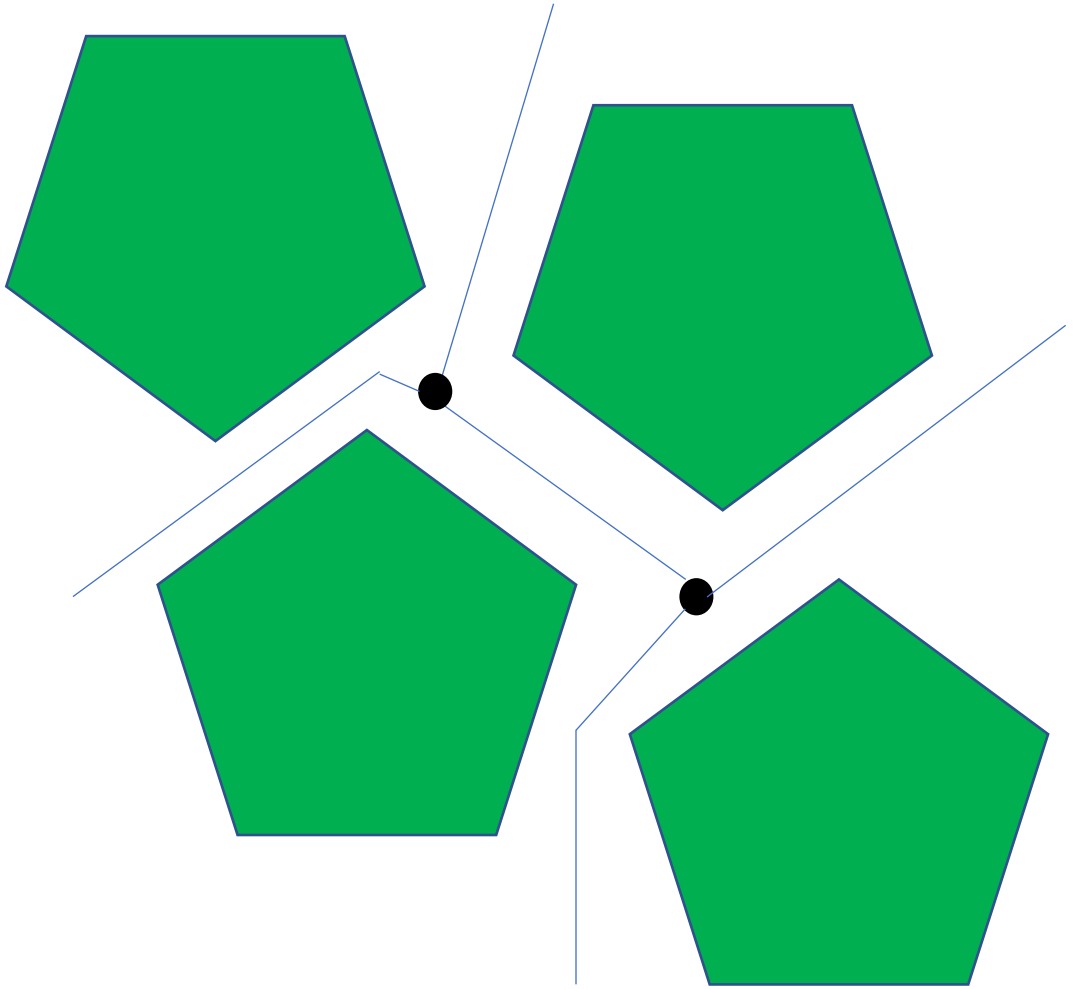
A Visibility Graph is a Kind of Roadmap

- What are the strengths / weaknesses of roadmaps?
- What are some other types of roadmaps?

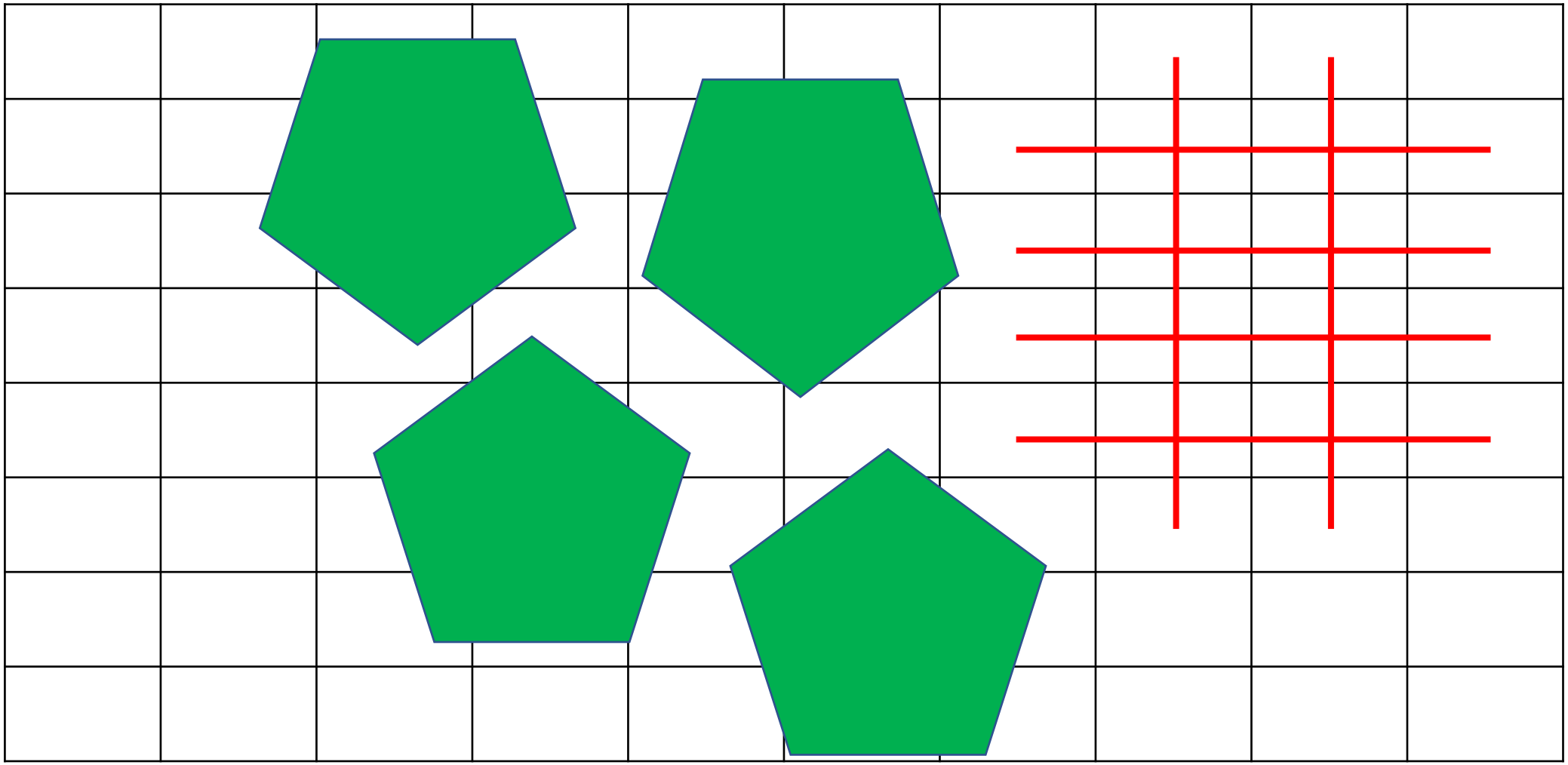
Assume translation but no rotation



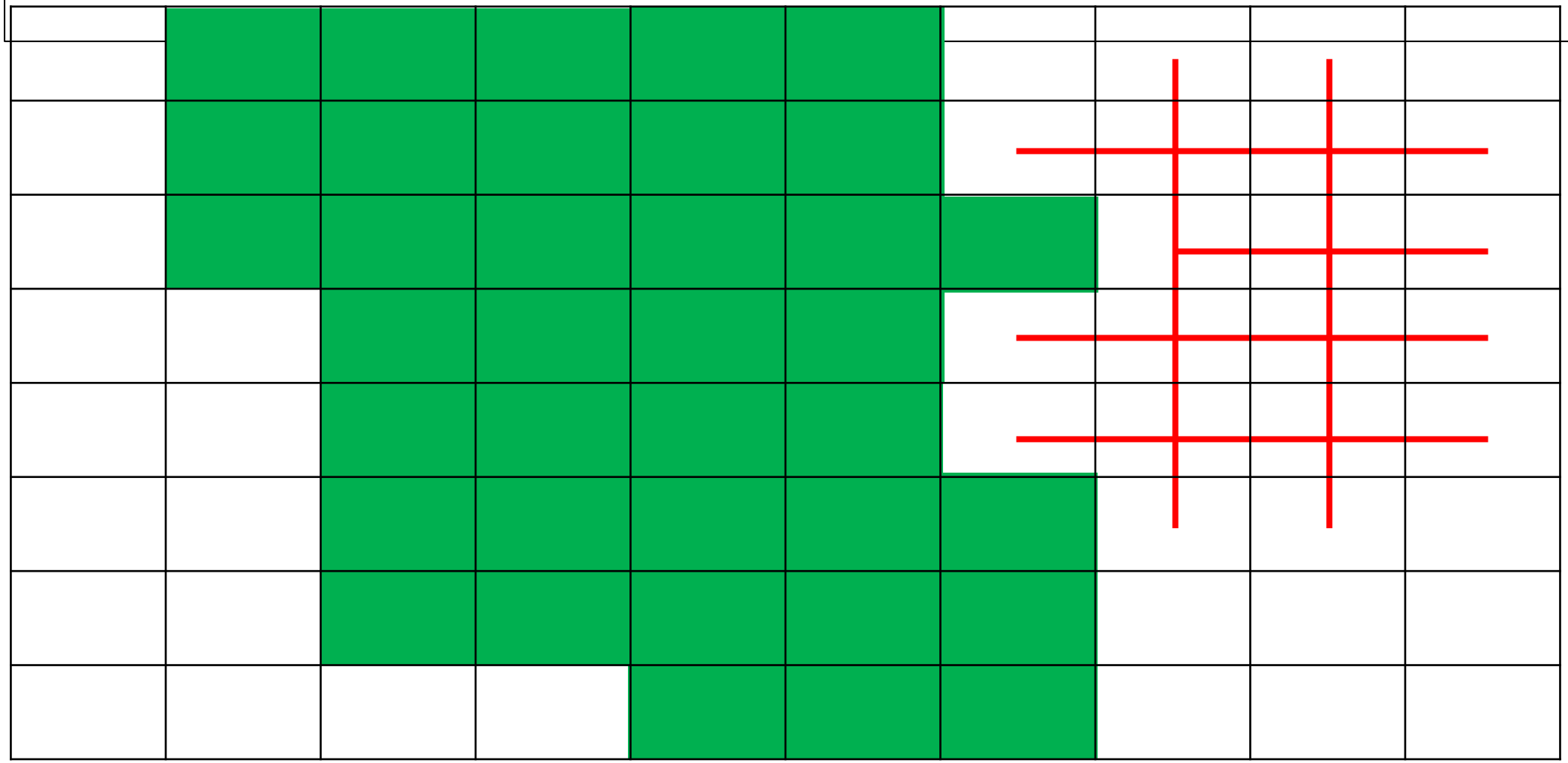
Voronoi Diagrams



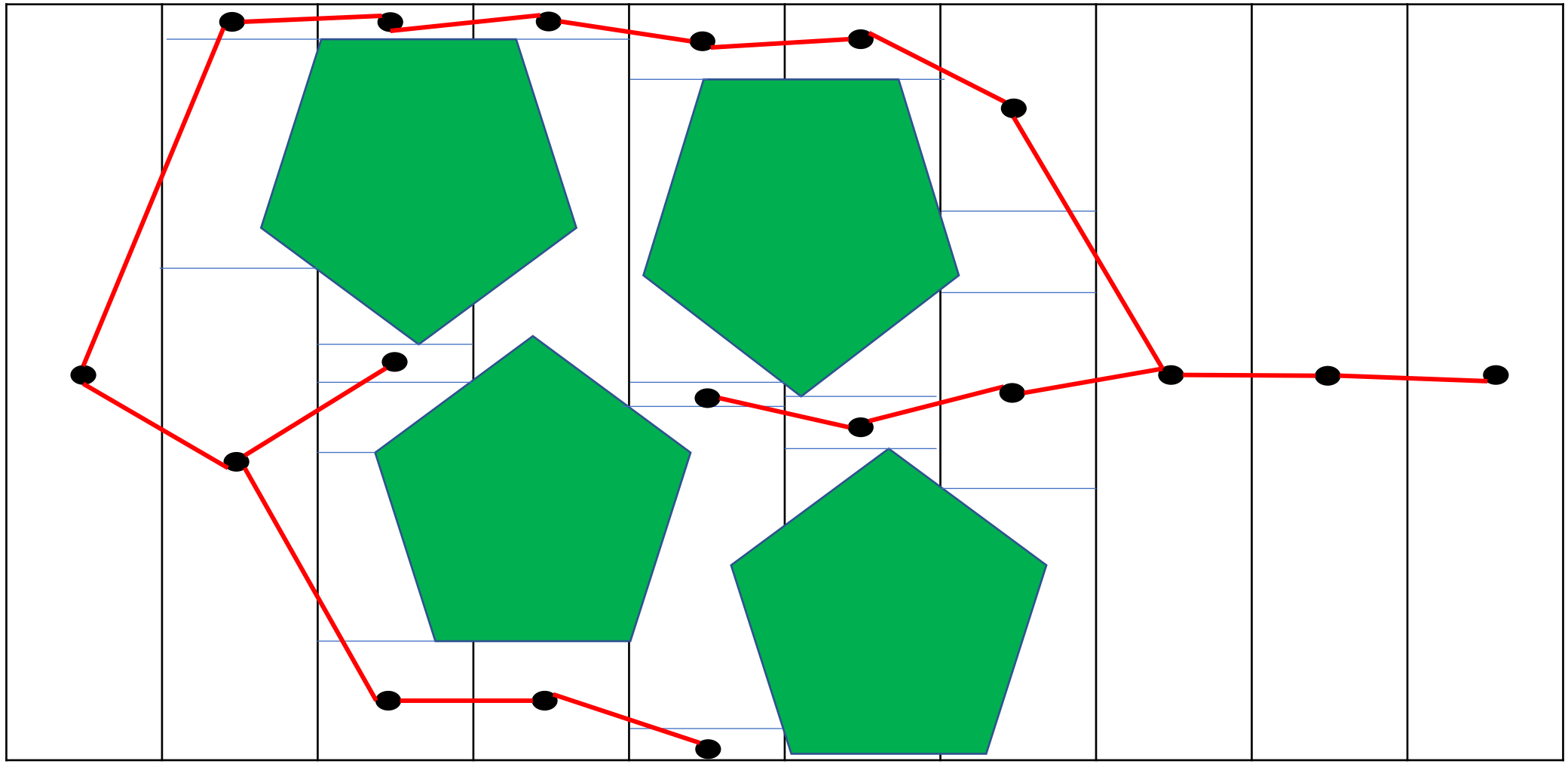
Roadmaps: Approximate Fixed Cell



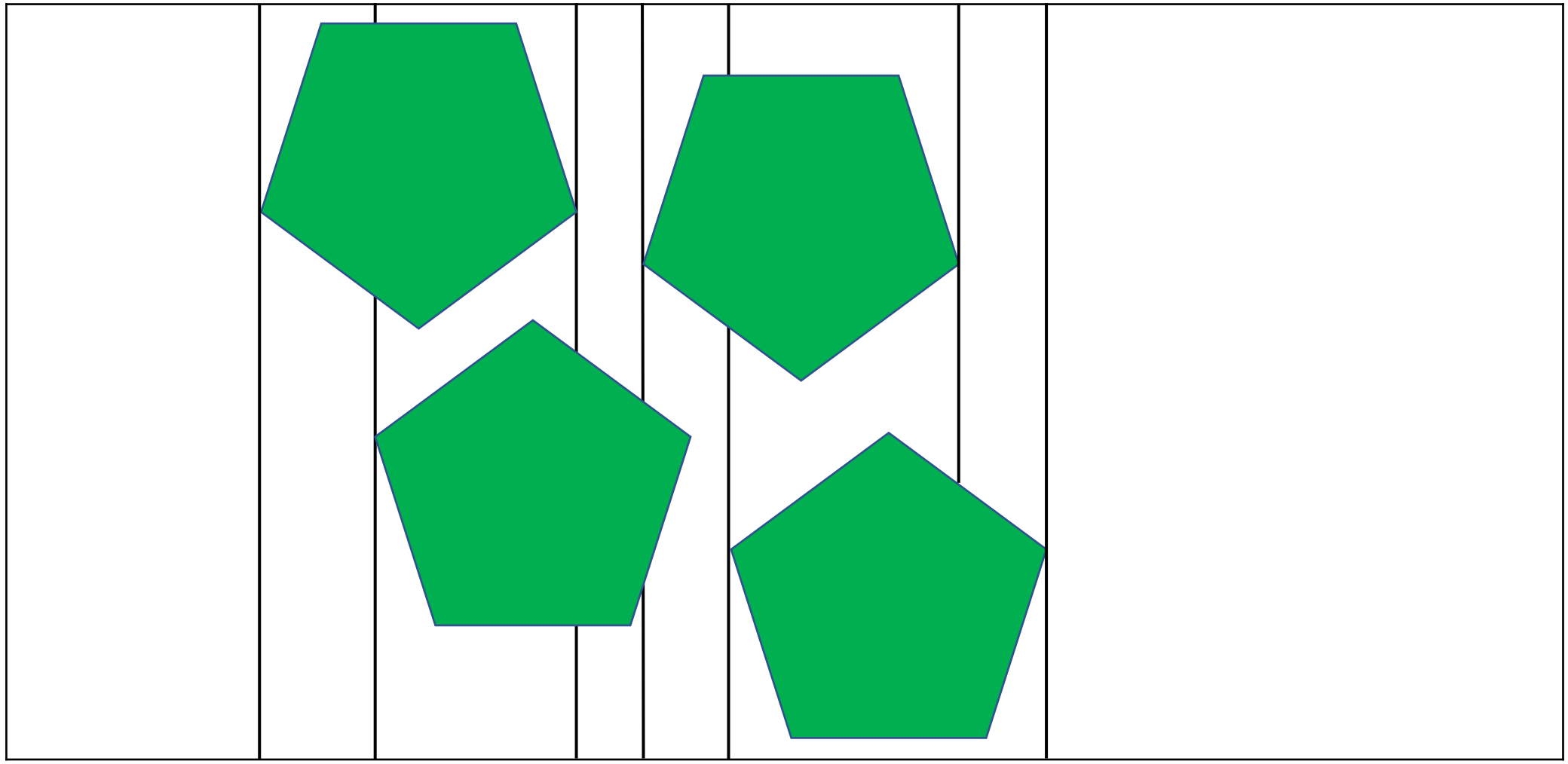
Roadmaps: Approximate Fixed Cell



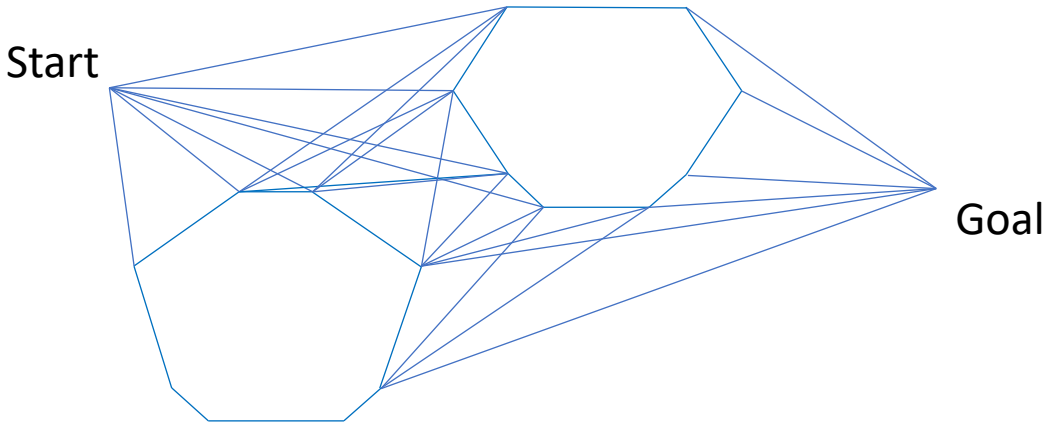
Roadmaps: Approximate Variable Cell

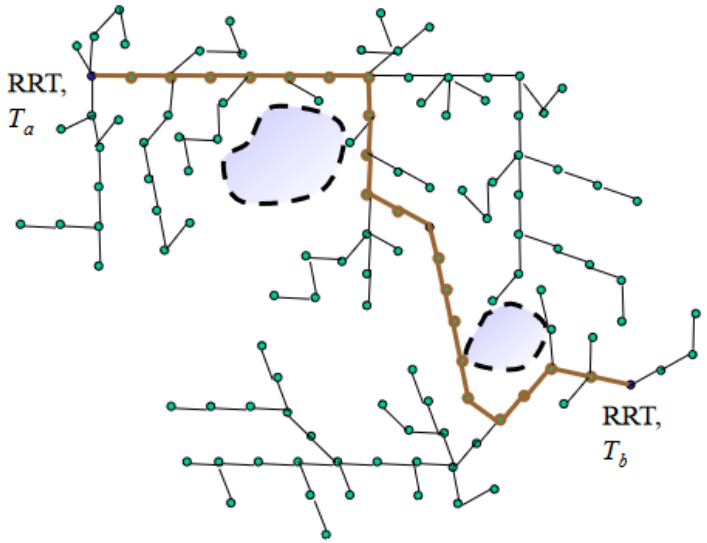


Exact Cell Decomposition



How do we handle large state spaces?

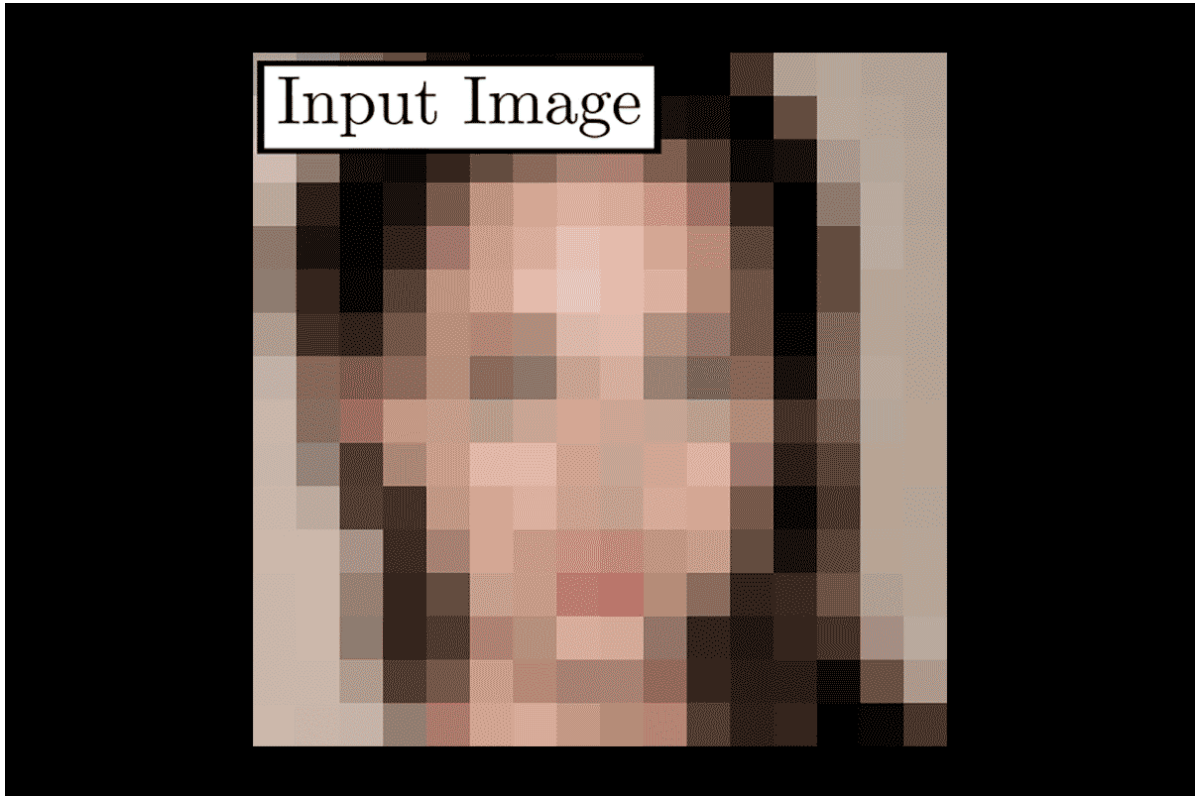




Rapidly Exploring **R**andom **T**rees

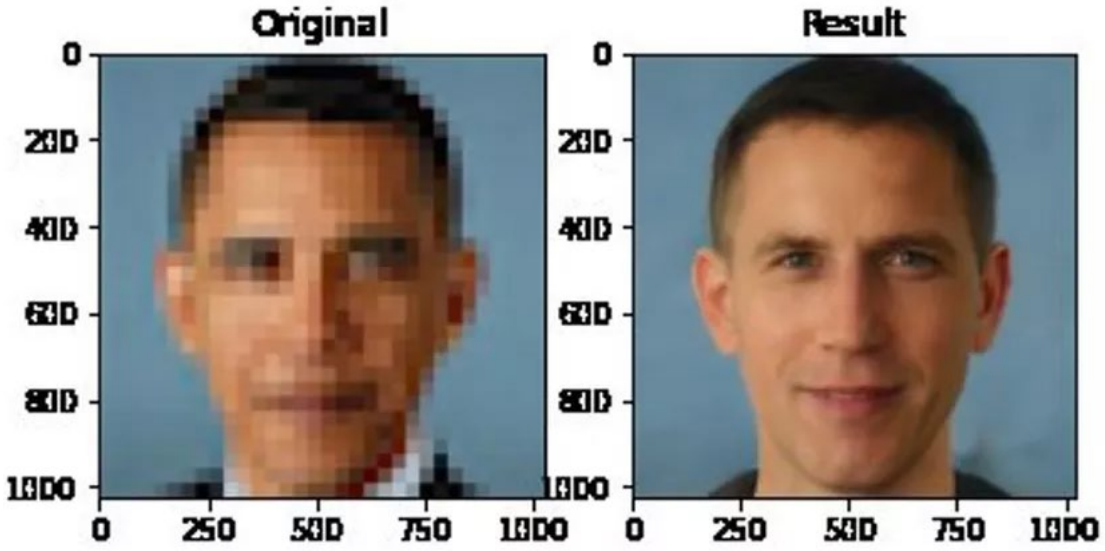
Mid-lecture Break

Mid-lecture Break



Menon, S., Damian, A., Hu, S., Ravi, N. and Rudin, C., 2020. PULSE: Self-Supervised Photo Upsampling via Latent Space Exploration of Generative Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2437-2445).

Mid-lecture Break



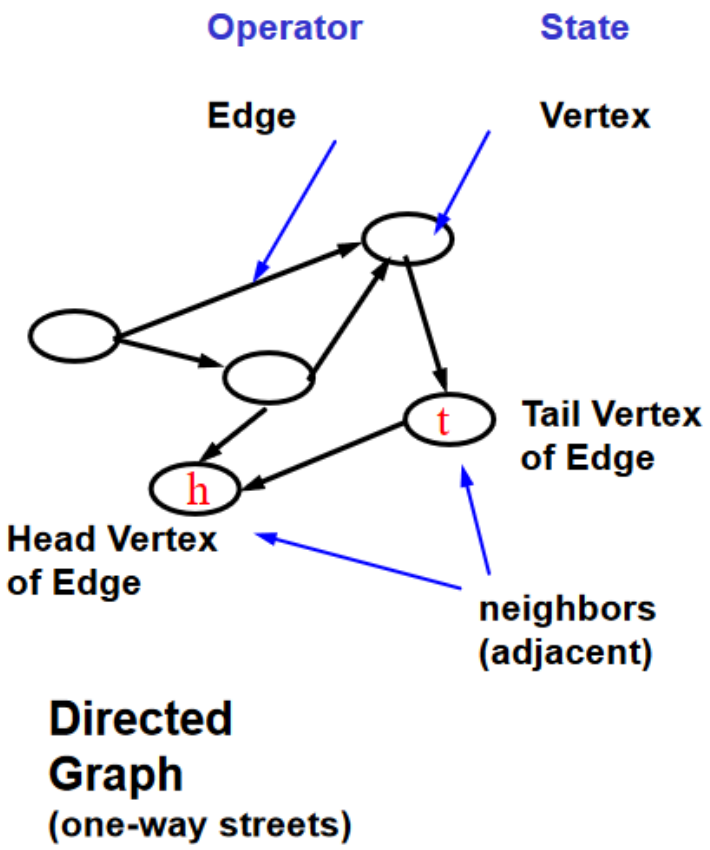
Mid-lecture Break

Outline:

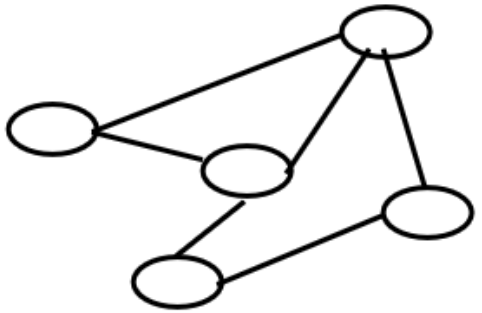
Problem Solving as State Space Search

- Problem Formulation (Modeling)
  - Problem solving as state space search
- Formal Representation
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

Problem Formulation: A Graph

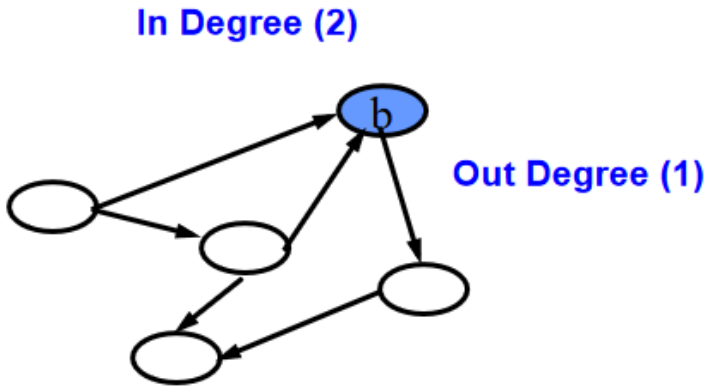


Directed Graph
(one-way streets)

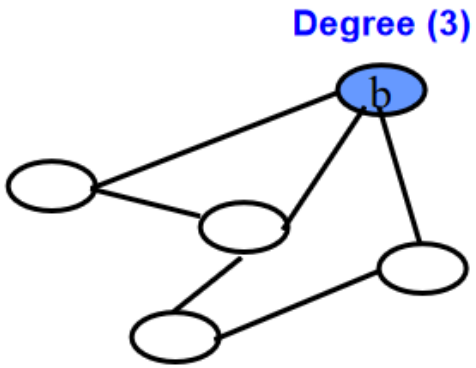


Undirected Graph
(two-way streets)

Problem Formulation: A Graph



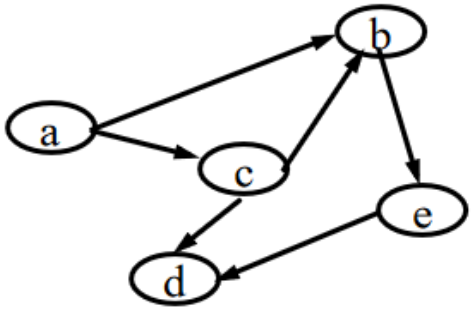
Directed Graph
(one-way streets)



Undirected Graph
(two-way streets)

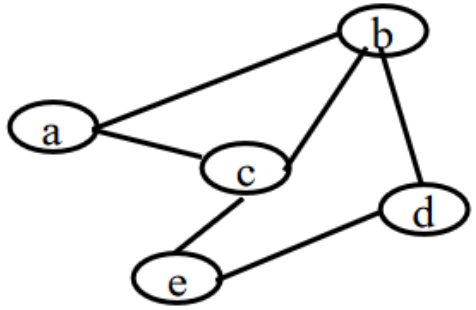
Problem Formulation: A Graph

Strongly connected graph
Directed path between all vertices.



Connected graph
Path between all vertices.

Complete graph
All vertices are adjacent.



Sub graph
Subset of vertices
edges between vertices in Subset

Directed Graph
(one-way streets)

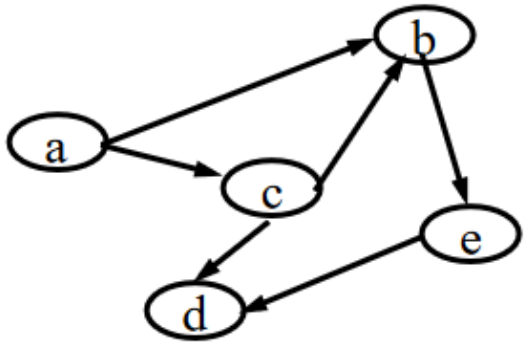
Clique
A complete subgraph
(All vertices are adjacent).

Undirected Graph
(two-way streets)

Specifying a Graph

Notation:

<a,b,…,n>	an ordered list of elements, a, b, …
{a,b,…,n}	an unordered set of distinguished elements

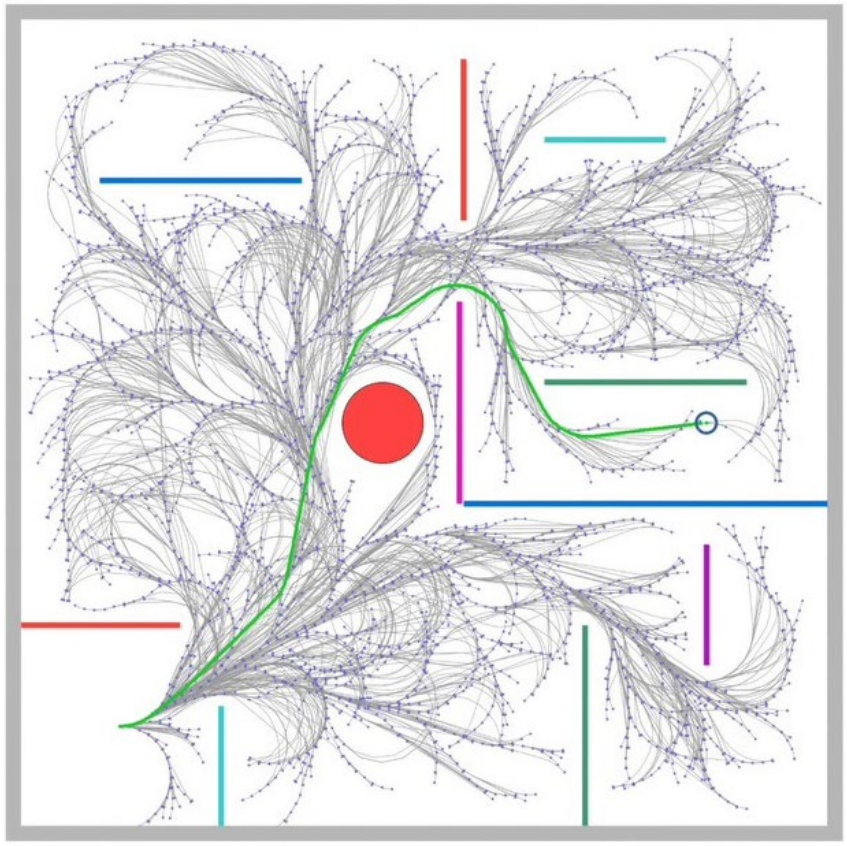


Vertices V = {a,b,c,d,e}

Edges E = {<a,b>, <a,c>, <b,e>, <c,b>, <c,d>, <e,d>}

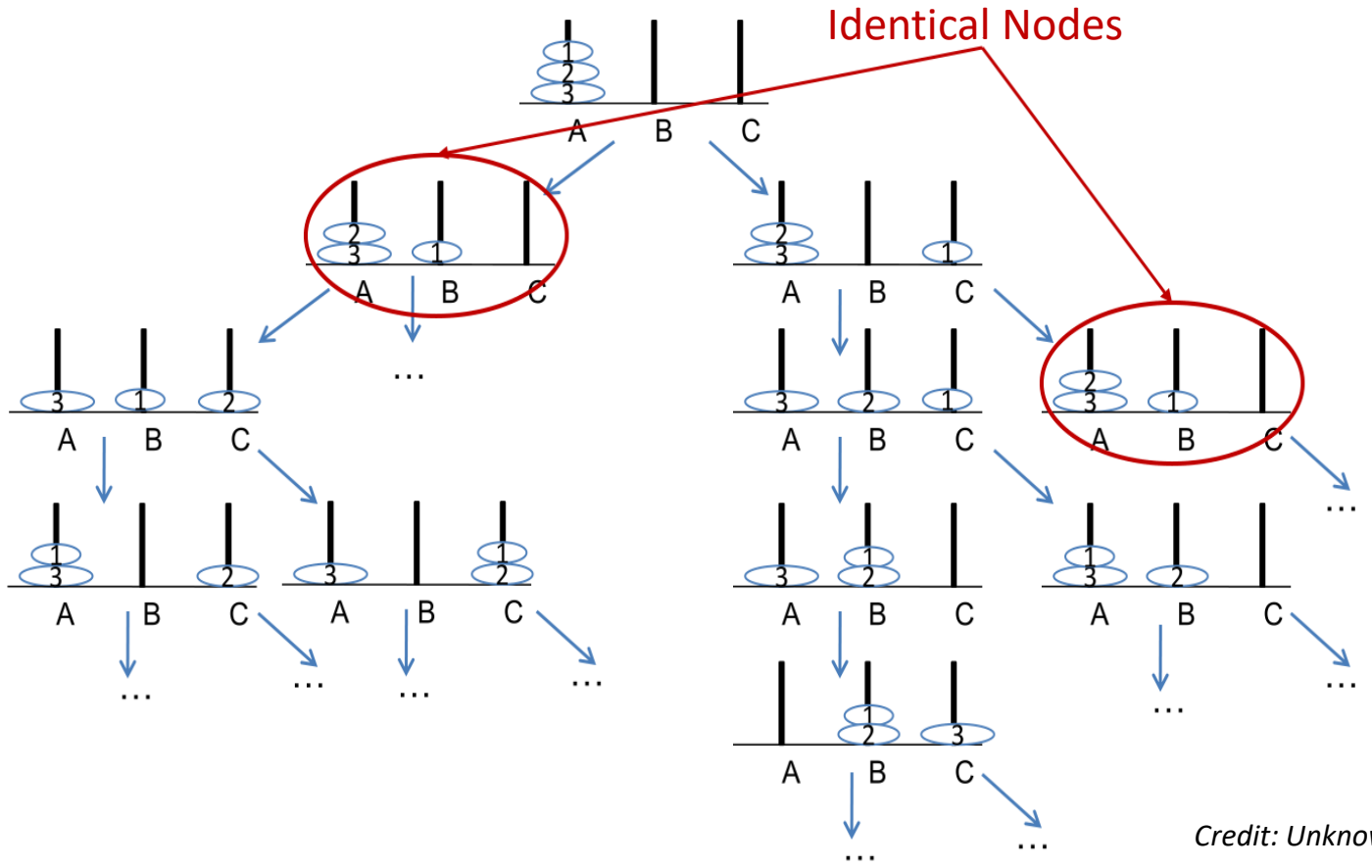
Examples of Graphs

Roadmap



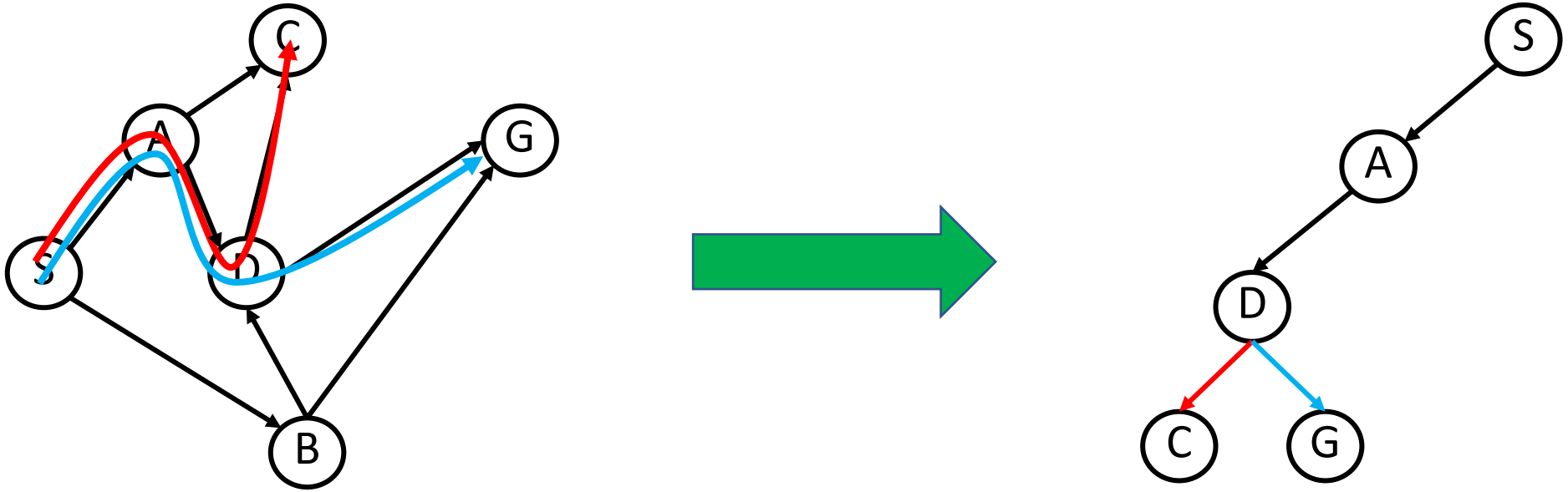
Credit: Olzhas Adiyatov and H. Atakan Varol

Planning Actions
(Graph of possible states of the world)

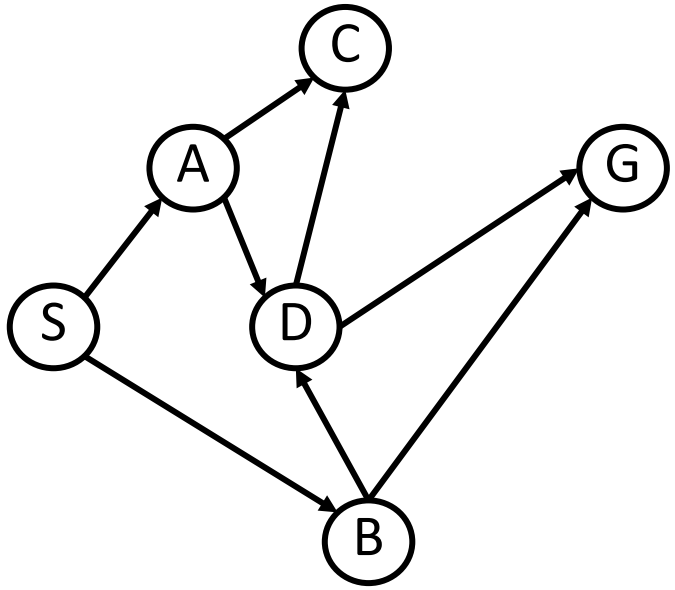


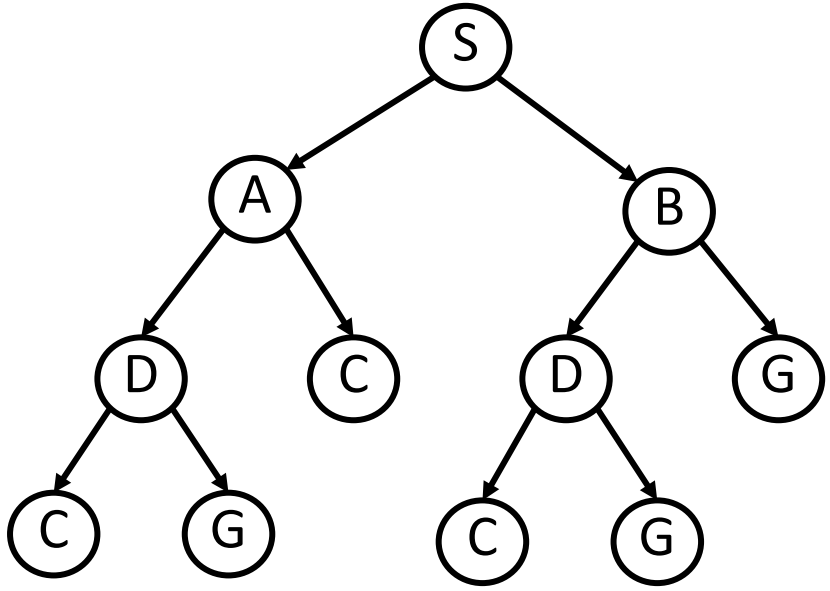
Credit: Unknown

A Problem Solver Searches Through All Simple Paths



Search Tree Denotes All Simple Paths

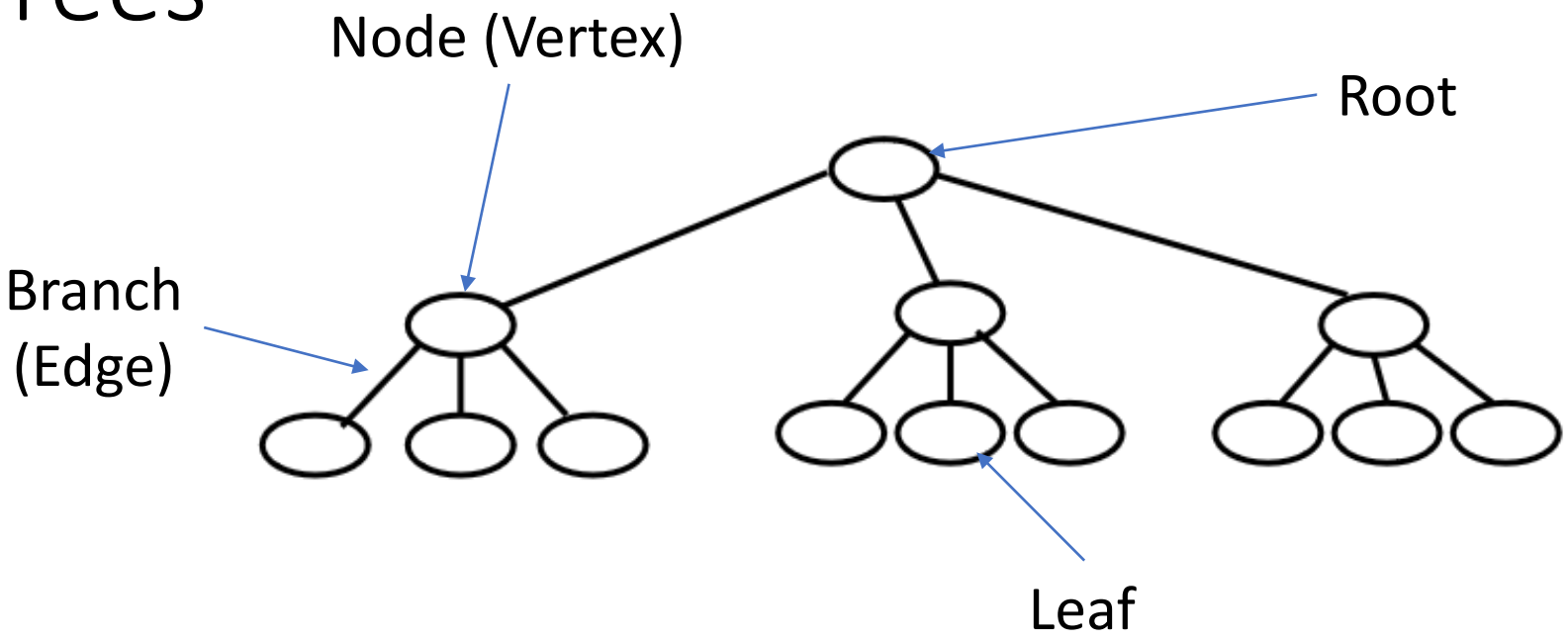




Enumeration is:

- Complete
- Sound

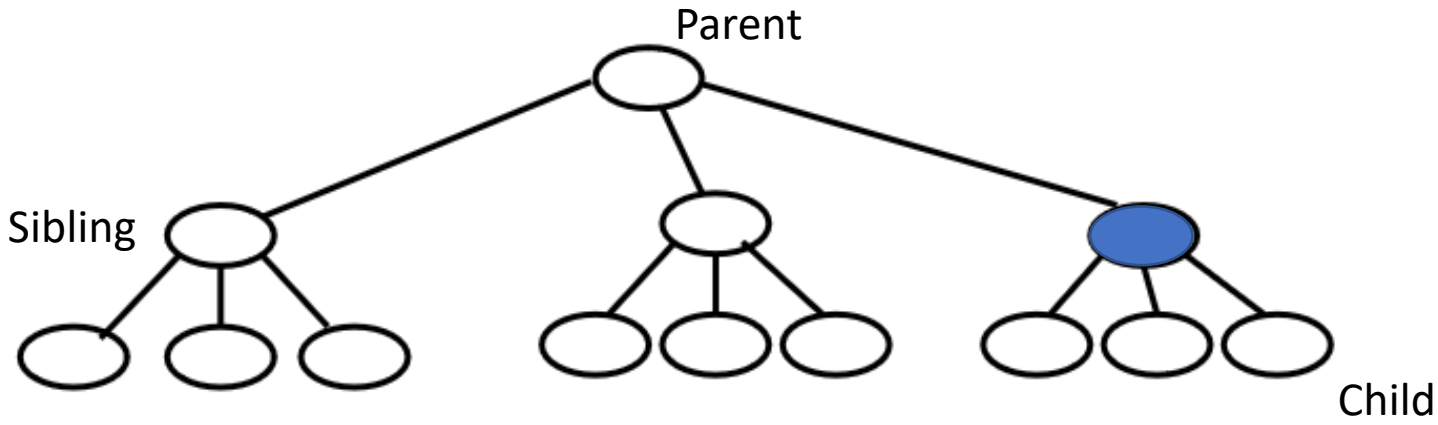
Search Trees



A tree T is a directed graph such that:

- There exists exactly one undirected path between any pair of vertices
- *In degree* of each vertex is 1

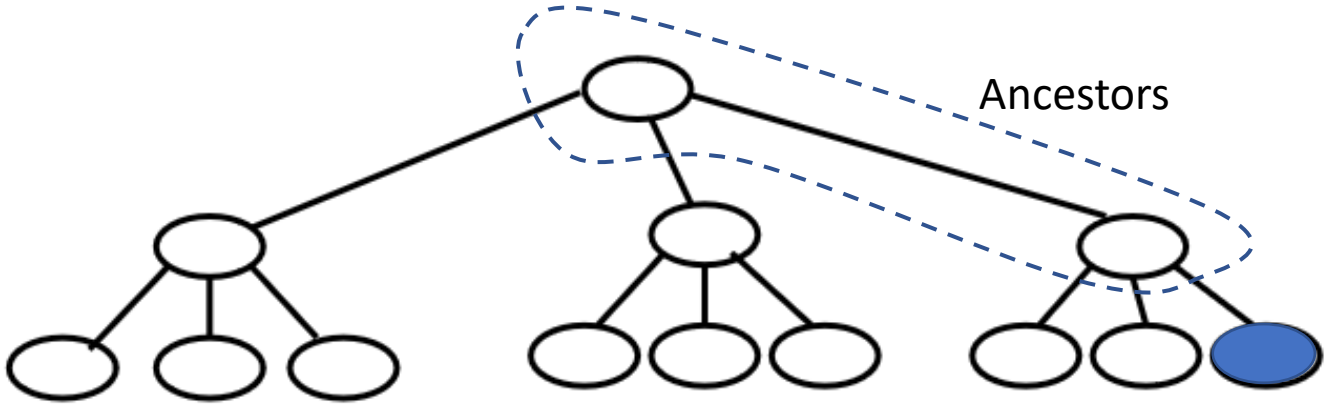
Search Trees



A tree T is a directed graph such that:

- There exists exactly one undirected path between any pair of vertices
- *In degree* of each vertex is 1

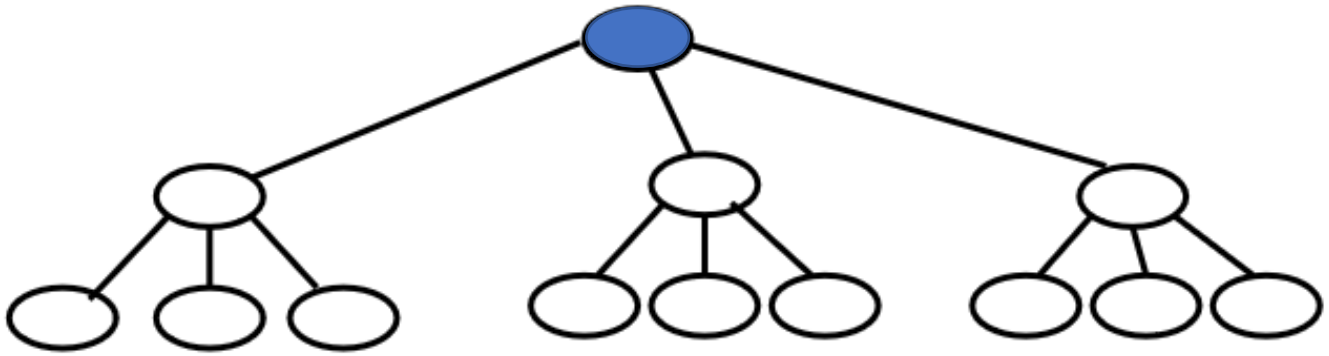
Search Trees



A tree T is a directed graph such that:

- There exists exactly one undirected path between any pair of vertices
- *In degree* of each vertex is 1

Search Trees



Descendants

A tree T is a directed graph such that:

- There exists exactly one undirected path between any pair of vertices
- *In degree* of each vertex is 1

Outline:

Problem Solving as State Space Search

- Problem Formulation (Modeling)
  - Problem solving as state space search
- Formal Representation
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

Taxonomy of Search

Class	Types	Description
Blind (uninformed)	Depth-First Breadth-First Iterative-Deepening	Systematic exploration of the whole tree until the goal is found
Optimal (informed)	A* Branch & Bound	Use path “length” measure to find the shortest path
Heuristic (informed)	Hill-Climbing Best-First Beam	Use a heuristic measure of goodness of a node

Simple Search Algorithm

Let Q be a list of partial paths,

S be the start node and

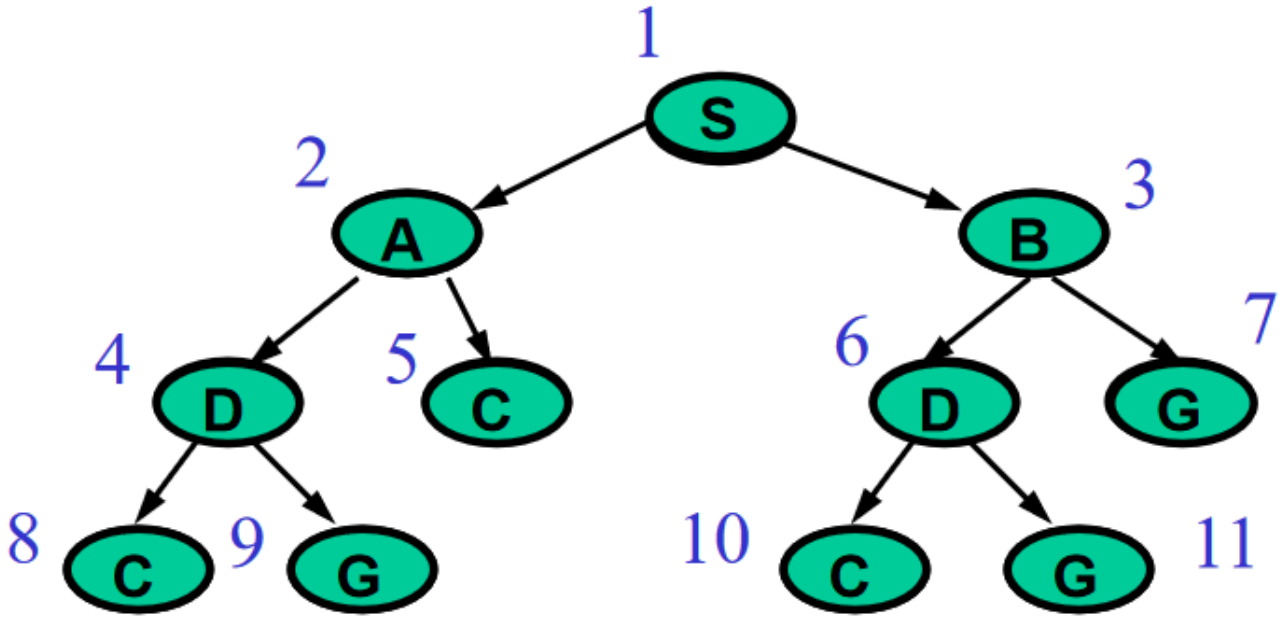
G be the goal node.

1. Initialize Q with a partial path <S>
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N // Goal reach!
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and create a one-step extension of N to each child
   c) Add all extended paths to Q // How we add to Q defines search type
   d) GOTO step 2

Breadth-First Search

Idea: After visiting a node:

- Visit each child
- Visit each child's sibling before the child's children



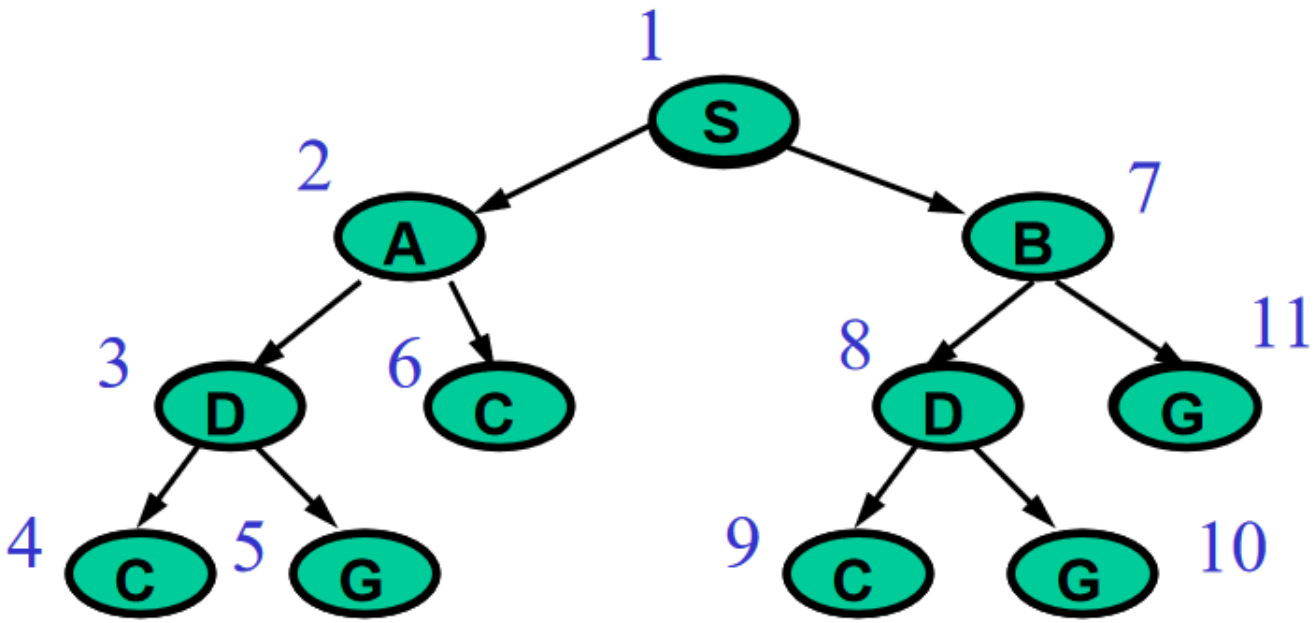
What kind of queue is this?

→ First-in-first-out (FIFO)

Depth-First Search

Idea: After visiting a node:

- Visit each child
- Visit each child's children before the child's sibling

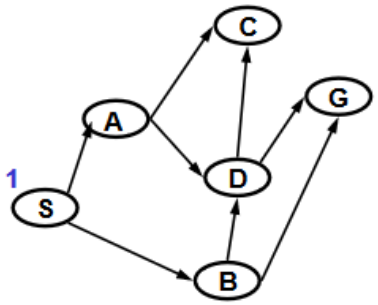


What kind of queue is this?

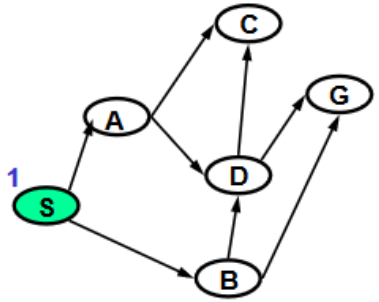
→ Last-in-first-out (LIFO)

DFS Example

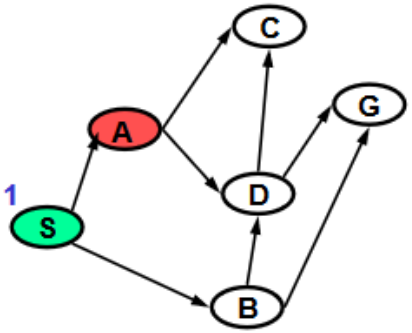
	Q
1	(S)
2	
3	
4	
5	
6	



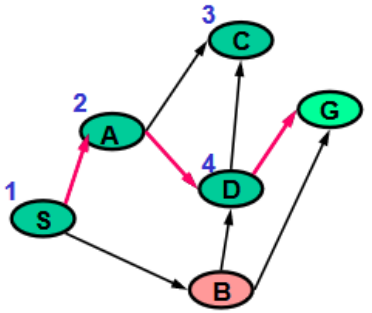
	Q
1	(S)
2	
3	
4	
5	



	Q
1	(S)
2	(A S)
3	
4	
5	

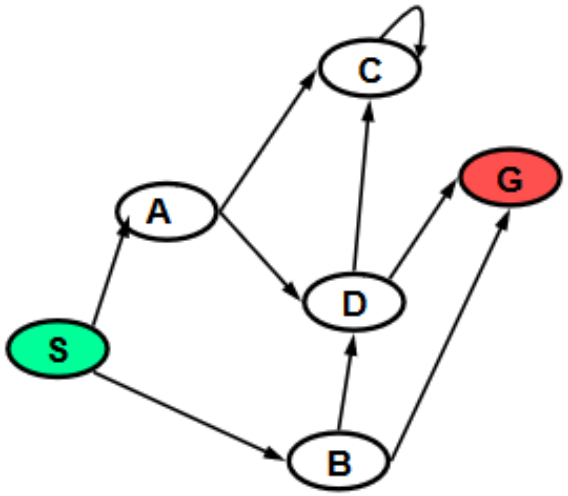


	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)
6	(G D A S)(B S)



Issue

With DFS, will we ever reach G?



Is effort wasted even when you don’t have cycles?

Solution: Visited List

Idea:

- Keep track of nodes already visited
- Do not place expanded path on Q if head is a visited node

Does this maintain correctness?

- Any goal reachable from a node that was visited a second time would be reachable from that node the first time

Does this always improve efficiency?

- Visits only a subset of the original path

Simple Search Algorithm (Handling Cycles)

Let Q be a list of partial paths,
S be the start node and
G be the goal node.

1. Initialize Q with a partial path <S>; **set Visited = {}**
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N //Goal reach!
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) **not in Visited** and create a one-step extension of N to each child
   c) Add all extended paths to Q
   d) Add Children of head(N) to Visited
   e) GOTO step 2

Elements of Algorithmic Design

Algorithmic Description:

- Pseudocode vs. Implementation

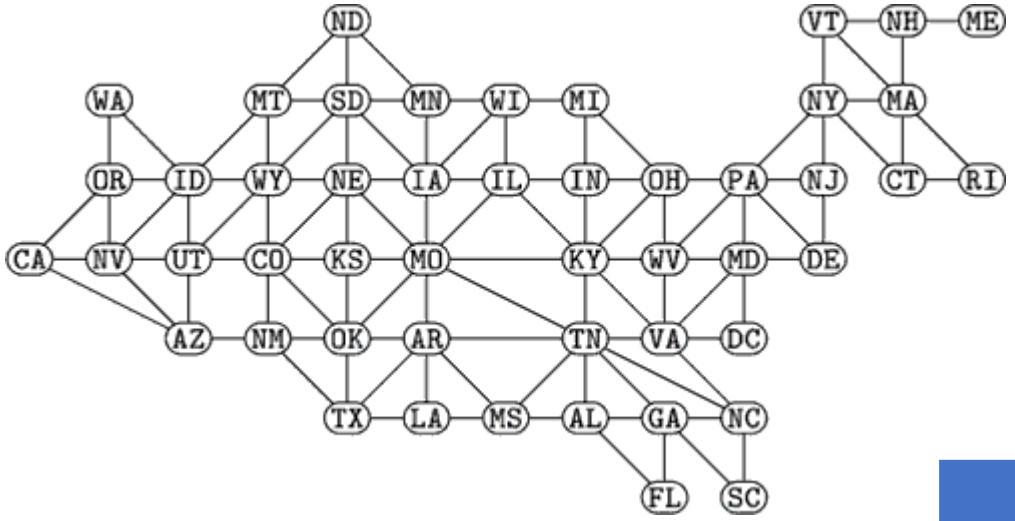
Algorithm Analysis:

- Time complexity
  - How long does it take to find a solution?
- Space complexity
  - How much memory does it need to perform search?
- Soundness
  - When a solution is returned, is it guaranteed to be correct?
- Completeness
  - Is the algorithm guaranteed to find a solution when one exists?

Takeaways

- Most decision-making tasks can be formulated as a state space search
- State space search is formalized using graphs, simple paths, search trees, and pseudo code
- DFS and BFS are framed are generic search strategies
- Cycle detection is required to achieve efficiency
  (and completeness in DFS)

PSet 1



	Visited List	Time (s)	# Paths Popped from Queue	Max Queue Size	Returned Path's Length/Cost
DFS	No				
	Yes				
BFS	No				
	Yes				
IDS	No				
	Yes				