CMPE-633. Topics in Robotics and Control

Lecture 5

Abubakr Muhammad

Discrete Planning

- Planning (robotics) or Problem Solving (AI) ?
- We will study
  - Discrete configuration spaces or state spaces
  - Modeling planning problems as graph search algorithms (feasible planning)
  - Optimal planning (dynamic programming)
- No geometric models or differential equations
- Key towards a unified approach towards planning problems
- Reference: Planning Algorithms by LaValle Ch 2

State Space Models

- A distinct situation is a state, say x
- Set of all possible states is State Space X
- World is transformed through actions (controls)
- Actions are chosen by a planner
- Each action u, when applied to state x, produces a new state x', via State transition function f

$$x' = f(x,u)$$

Discrete Feasible Planning

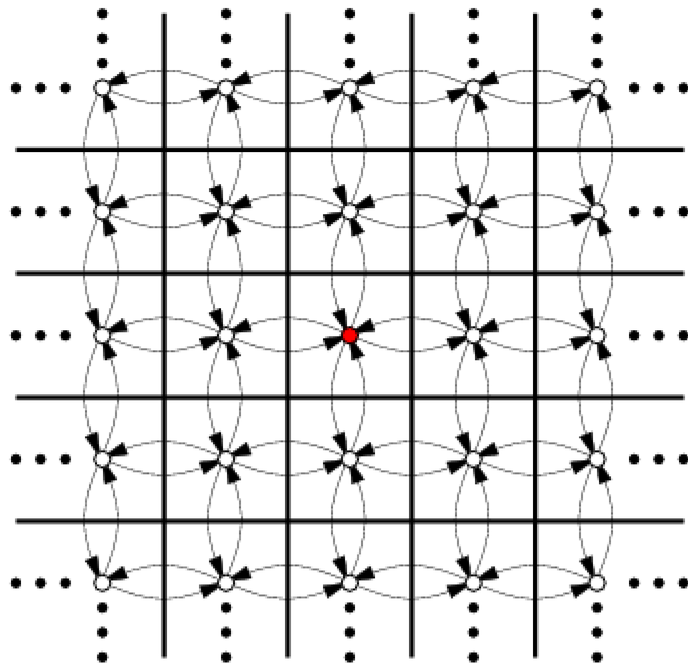
Formulation 2..1 (Discrete Feasible Planning)

1. A nonempty *state space* X, which is a finite or countably infinite set of *states*.
2. For each state $x \in X$, a finite *action space* $U(x)$.
3. A *state transition function* f that produces a state $f(x, u) \in X$ for every $x \in X$ and $u \in U(x)$.
4. An *initial state* $x_I \in X$.
5. A *goal set* $X_G \subset X$.

- U(x) : set of all actions that can be applied from state x.
- Choose controls to stear state to the goal.

Example: A labyrinth

Example: An infinite tiled floor

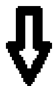


Example 2: Puzzle games

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

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

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



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

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

Sounds familiar?

- When state space is finite
  - Finite state machines (Mealy/Moore machines)
  - Deterministic Finite automata (DFA)

How to solve? Search algorithms

- Graph search algorithms
- Requirement: systematic (keep track of visited states; visit every reachable state)

General forward search

```
FORWARD_SEARCH
1   Q.Insert(x_I) and mark x_I as visited
2   while Q not empty do
3       x ← Q.GetFirst()
4       if x ∈ X_G
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' not visited
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

Backward Search

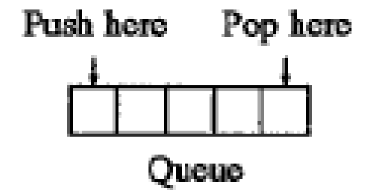
```
BACKWARD_SEARCH
1   Q.Insert(x_G) and mark x_G as visited
2   while Q not empty do
3       x' ← Q.GetFirst()
4       if x = x_I
5           return SUCCESS
6       forall u^{-1} ∈ U^{-1}(x)
7           x ← f^{-1}(x', u^{-1})
8           if x not visited
9               Mark x as visited
10              Q.Insert(x)
11          else
12              Resolve duplicate x
13  return FAILURE
```

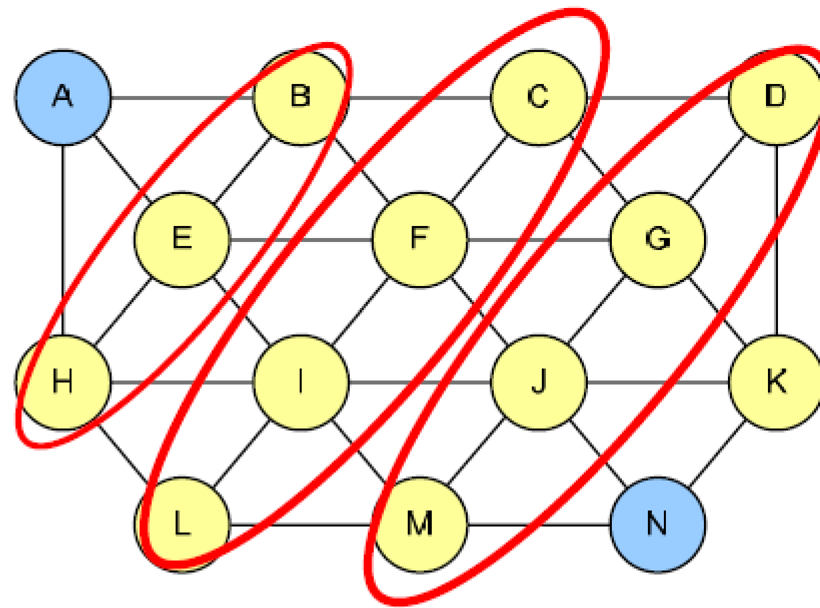
Figure 2.6: A general template for backward search.

General forward search

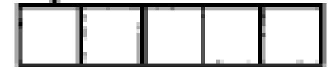
- States
  - Unvisited
  - Dead
  - Alive
- Set of alive states kept in a priority queue
- Implementation details (omitted)
  - How to determine whether x is in X_G (representation) [Line 4]
  - How to verify whether x has been visited? (lookup tables? Hashing?) [Line 8]
  - How to sort the queue?

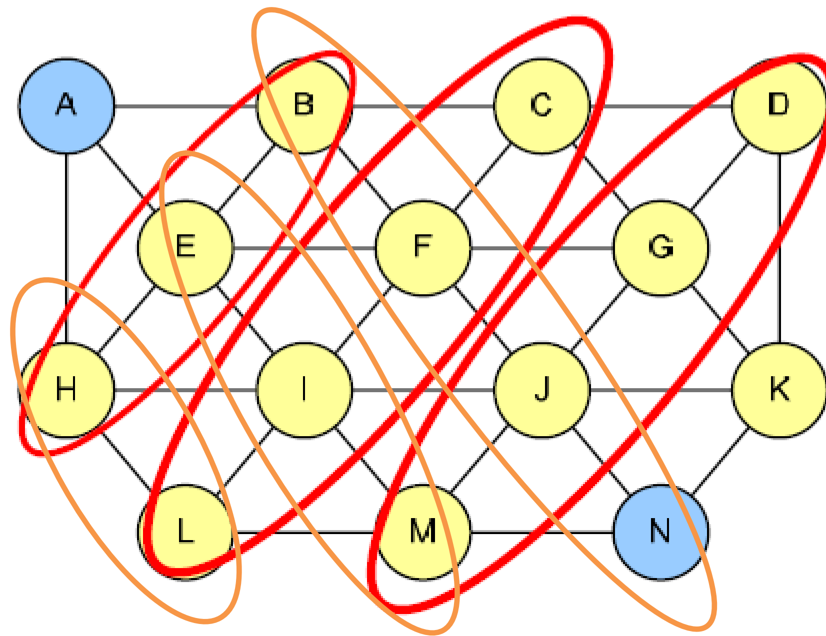
Breadth First Search





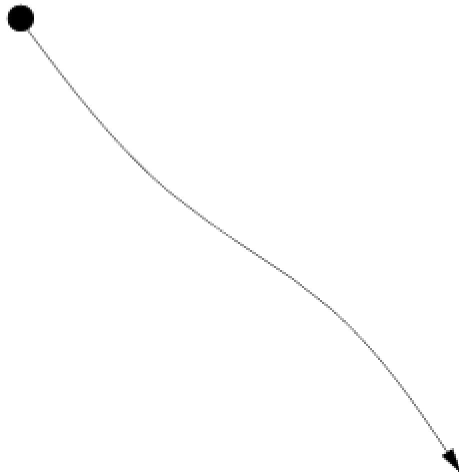
Depth First Search

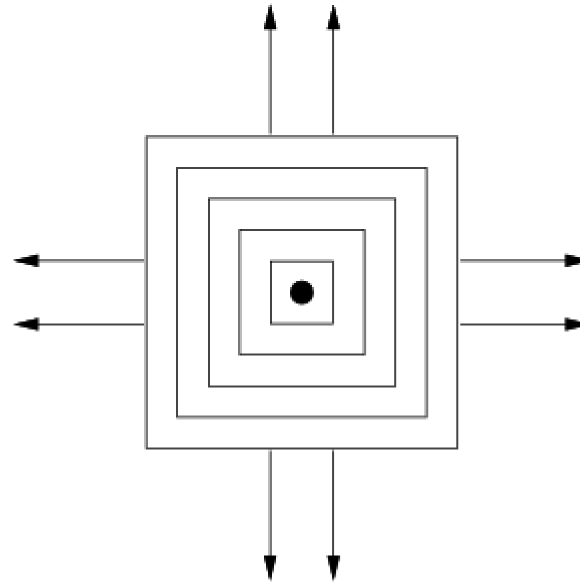




Systematic searches

Requirement: systematic
- keep track of visited states
- visit every reachable state)





BFS is systematic
DFS is systematic for finite graphs
For finite graphs all searches are systematic

Dijkstra's algorithm

- Optimal feasible paths
- Cost associated to each edge $l(e) = l(x,u)$
- Total cost of plan = sum of edge costs
- Cost to come $C: X \rightarrow [0,\infty]$
- Optimal cost to come $C^*(x)$

Dijkstra's algorithm

- Algorithm
  - Initially $C^*(x_I) = 0$
  - Generate new state $x' = f(x,u)$
  - For each new state $C(x') = C^*(x) + l(x,u)$
  - If x' already exists then update $C^*(x)$
  - Reorder queue according to costs