To download the discussion slides, go to http://april.eecs.umich.edu/courses/eecs492_w10/wiki/index.php/Discussion_Slides

- Deterministic, Stochastic and Strategic Environment
  - Deterministic – Completely predictable
  - Source of Non-determinism
    - Stochasticity
    - Strategic actions of other agents
  - A strategic environment is not necessarily otherwise deterministic
  - Example: poker is both stochastic and strategic

Error occurred during initialization of VM
Could not reserve enough space for object heap
Could not create the Java virtual machine.

- Solution: Limit the size of maximum memory for VM (Max memory of 512M)
  - Step 1: setenv ANT_OPTS "-Xmx512m" on command line
  - Step 2: add memoryMaximumSize="512m" to javac block in build-java section of your build.xml file
  - Step 3: run java with -Xmx512m parameter

1. Formulate the problem
2. Find a sequence of actions (offline) that achieves the goal
   - Offline vs. Online search
3. Execute the sequence of action
   - Usually for static, deterministic environment

- States
  - State: Internal representation of an agent about the world: what the agent cares about
  - Not necessarily be correct
- Initial state
- Actions
  - Given a state, what the available actions are
- Transition model
  - Result(s,a) that returns state resulting from doing action a in state s
- Goal states
- Path cost

Vacuum world
- States: Agent location + dirt location? 8 states
- Initial State: Any
- Actions: Left, Right and Suck
- Transition model: Expected effects. Except for no effects on Left in leftmost square, Right in rightmost square and Suck in a clean square
- Goal state: All squares are clean
- Path cost: 1 (?)
Examples

- **7-queen**
  - States:
  - Initial State:
  - Actions:
  - Transition model:
  - Goal State
  - Path cost:

State Space vs. Search Space

- **State**: agent's representation of the world configuration
- **State space**: all reachable states from initial state
- **Search space**: a data structure that abstracts the state space
- **Nodes**: a data structure that represent states and related information (state, parent node, path cost...)
- **Edges**: represent actions and path costs (reflects transition model)
- **Solution**: is the path from initial to goal state
- **Optimal solution**: is the solution of the shortest path cost

Search

- **Uninformed search**: Only uses information in problem formulation
- **Informed search**: Has heuristics that guides an agent on where to look for solutions

Search Strategy

- **Order of node expansion**
  - Expansion: Given a node, creates all children of the node according to transition model

General Tree Search

```
function TreeSearch(problem)
    returns a solution, or failure
    fringe = new Queue();
    fringe.put(problem.initialState)
    loop do
        if fringe.isEmpty() then return failure
        node = fringe.get()
        if problem.isGoalState(node) then return node;
        fringe.putAll(problem.expand(node))
    end
```

- **Only the order of the queue makes the difference**
- **Avoid repeated states (Graph search)**

Graph Search

```
function TreeSearch(problem)
    returns a solution, or failure
    closed = new Set();
    closed.add(problem.initialState);
    fringe = new Queue();
    fringe.put(problem.initialState)
    loop do
        if fringe.isEmpty() then return failure
        node = fringe.get()
        if problem.isGoalState(node) then return node;
        for each child in problem.expand(node)
            if !closed.contain(child)
                fringe.put(child)
        end
    end
```

- **Avoid repeated states**
**Uninformed Search Strategies**

- **Breadth-First:**
  - FIFO queue, returning the oldest item
- **Uniform-Cost:**
  - Priority queue, returning the least-cost item
- **Depth-First:**
  - LIFO, returning the newest item
- **Depth-Limited DFS:**
  - Run DFS, cut off search at depth L
- **Iterative Deepening DFS:**
  - Run Depth-Limited DFS with L from 1 to infinity

**Example**

```
S = start, G = goal
```

**BFS Search Tree**

```
Queue = {S}
Select S
Goal(S) = true?
If not, Expand(S)
```

```
Queue = {A, D}
Select A
Goal(A) = true?
If not, Expand(A)
```

```
Queue = {D, B, D}
Select D
Goal(D) = true?
If not, expand(D)
```

```
Queue = {B, D, A, E}
Select B etc.
```
Level 3
Queue = {C, E, S, E, S, B, B, F}

Level 4
Expand queue until G is at front
Select G
Goal(G) = true

Queue = {A, D}

Queue = {B, D, D}

Queue = {C, E, D, D}

Queue = {D, F, D, D}

Queue = {D, J, D, D}
Queue = \{G, D, D\}

Iterative deepening search L=0

Iterative deepening search L=1

Iterative deepening search L=2

Iterative Deepening Search L=3

Uniform-Cost Search
Search Performance Criteria

- Completeness: Guaranteed to find a solution if it exists
- Optimality: The minimum path cost solution is found
- Time Complexity: How long it takes to find a solution
- Space Complexity: How much memory is needed

Notes:
1. BFS and Iterative Deepening only optimal when action cost is uniform;
2. UCS only optimal when action cost is nonnegative;

A*

- Informed (heuristic) search
  - Use heuristic function as a guidance on which node to expand
- Heuristic function for A*
  \[ f(n) = g(n) + h(n) \]
  where \( g(n) \) is the path cost (cost so far to reach \( n \)) and \( h(n) \) is the heuristic (estimated cost from \( n \) to goal)
  \( f(n) \) is an estimated total cost

A*

- To expand the nodes with smallest \( f(n) \)
- Admissible: Heuristic functions should never overestimate the path cost