AI Lab - Lesson 1

Uninformed Search

Davide Corsi
Alessandro Farinelli

University of Verona
Department of Computer Science

October 21^{st} 2021

UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

The OpenAI Gym Framework

What is it

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball

What is it for

- An open-source collection of environments that can be used for benchmarks
- A standardized set of tools to define and to work with environments

Where to find it

https://gym.openai.com

Installation Process

- Download the *Conda* package manager for Python 3.7 from `https://docs.conda.io/en/latest/miniconda.html`
- Install Conda on your system
- Set-Up the conda environment with the configuration file at `https://github.com/d-corsi/AI-Lab/tree/master/tools/ai-lab-environment.yml`

Detailed guide for the Installation Process:

`https://github.com/d-corsi/AI-Lab`

Tutorial

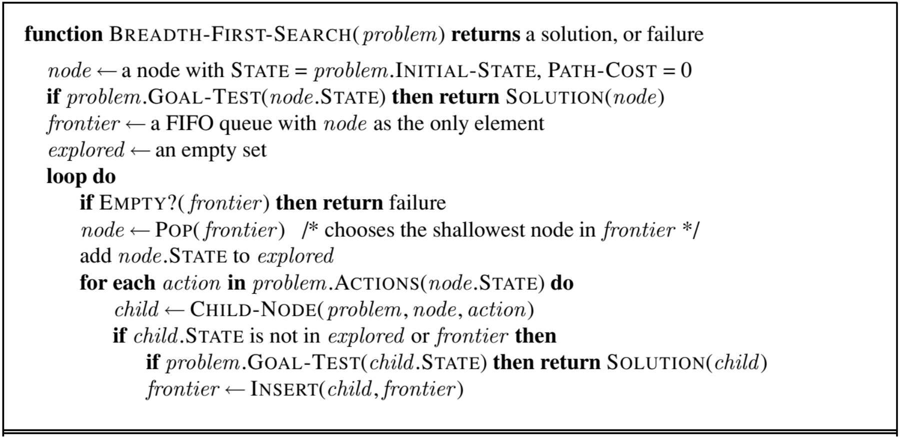
To open the tutorial:

- Navigate to your local ai-lab folder.
- Activate ai-lab conda environment and launch Jupyter Notebook.
- Navigate with your browser to: *lesson_1/lesson_1_tutorial.ipynb*

Assignments

- Your assignments for this session are at: *lesson_1/lesson_1_problem.ipynb*. You will be required to implement some uninformed search algorithms
- In the following you can find pseudocodes for such algorithms

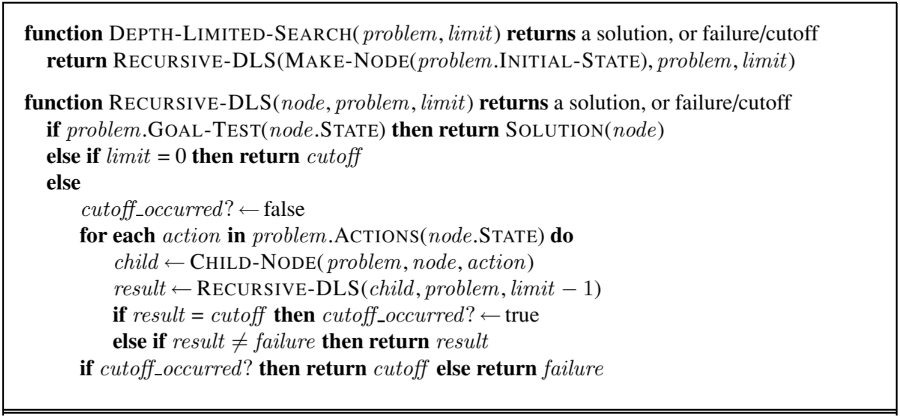
Breadth-First Search (BFS)



```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

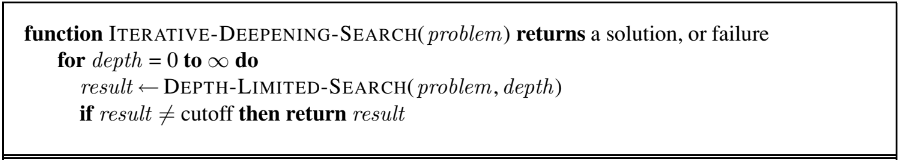
Note: this is a graph search version

Iterative Deepening Search (IDS)



```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if $limit = 0$ then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, $limit - 1$)
          if $result = cutoff$ then cutoff_occurred? ← true
          else if $result \neq failure$ then return result
      if cutoff_occurred? then return cutoff else return failure
```



```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
   for $depth = 0$ to $\infty$ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
      if $result \neq$ cutoff then return result
```

Note: this is a tree search version