

Planning Lab - Lesson 1

Uninformed Search

Luca Marzari and Alessandro Farinelli

University of Verona
Department of Computer Science

October 14, 2022



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>

During the lab lessons we will use Jupyter notebook files. In order to use these files you should install the following dependencies.

Detailed guide for the installation process:

<https://github.com/LM095/Planning-Lab>

- Download the *Anaconda* package manager for Python 3.7 from <https://www.anaconda.com/distribution/#download-section>
- Install Conda on your system
- Open a terminal and digit:

```
> git clone https://github.com/LM095/Planning-Lab
> cd Planning-Lab
> conda env create -f tools/planning-lab-env.yml
> conda activate planning-lab
```

To open the tutorial:

- Navigate to your local Planning-Lab folder.
- Ensure that you have activated the *planning-lab* conda environment and launch Jupyter Notebook (`> jupyter notebook`) from your folder
- Navigate with your browser to: *lesson_1/lesson_1_tutorial.ipynb*

Assignments

- Your assignments for this lesson 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

Uninformed Search: tree and graph search versions

function TREE-SEARCH(*problem*) **returns** a solution, or failure
 initialize the frontier using the initial state of *problem*
 loop do
 if the frontier is empty **then return** failure
 choose a leaf node and remove it from the frontier
 if the node contains a goal state **then return** the corresponding solution
 expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
 initialize the frontier using the initial state of *problem*
 initialize the explored set to be empty
 loop do
 if the frontier is empty **then return** failure
 choose a leaf node and remove it from the frontier
 if the node contains a goal state **then return** the corresponding solution
 add the node to the explored set
 expand the chosen node, adding the resulting nodes to the frontier
 only if not in the frontier or explored set

Breadth-First Search (BFS): graph search version

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

node \leftarrow a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

frontier \leftarrow a FIFO queue with *node* as the only element

explored \leftarrow an empty set

loop do

if EMPTY?(*frontier*) **then return** failure

node \leftarrow POP(*frontier*) /* chooses the shallowest node in *frontier* */

 add *node*.STATE to *explored*

for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**

child \leftarrow 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 \leftarrow INSERT(*child*, *frontier*)

Iterative Deepening Search (IDS): tree search version

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? \leftarrow false

for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**

child \leftarrow CHILD-NODE(*problem*, *node*, *action*)

result \leftarrow RECURSIVE-DLS(*child*, *problem*, *limit* - 1)

if *result* = *cutoff* **then** *cutoff_occurred?* \leftarrow 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** ∞ **do**

result \leftarrow DEPTH-LIMITED-SEARCH(*problem*, *depth*)

if *result* \neq cutoff **then return** *result*