



Search in Python

Using AIMA Code

Today's topics

- AIMA Python code
- What it does
- How to use it
- Worked example: water jug program



Install AIMA Python ?

- [Aimacode](#) is a GitHub repo of python code linked to the AIMA book
- It's not available for pip installing 😞
 - Per [Peter Norvig](#)'s recommendation
- Workarounds
 - Clone repo on computer & follow [readme](#) instructions
 - Add directory path to your [PYTHONPATH](#) env variable
 - Use it with [Binder](#)
- We've put code we need in our 471 [code-and-data](#) repo

Two Water Jugs Problem

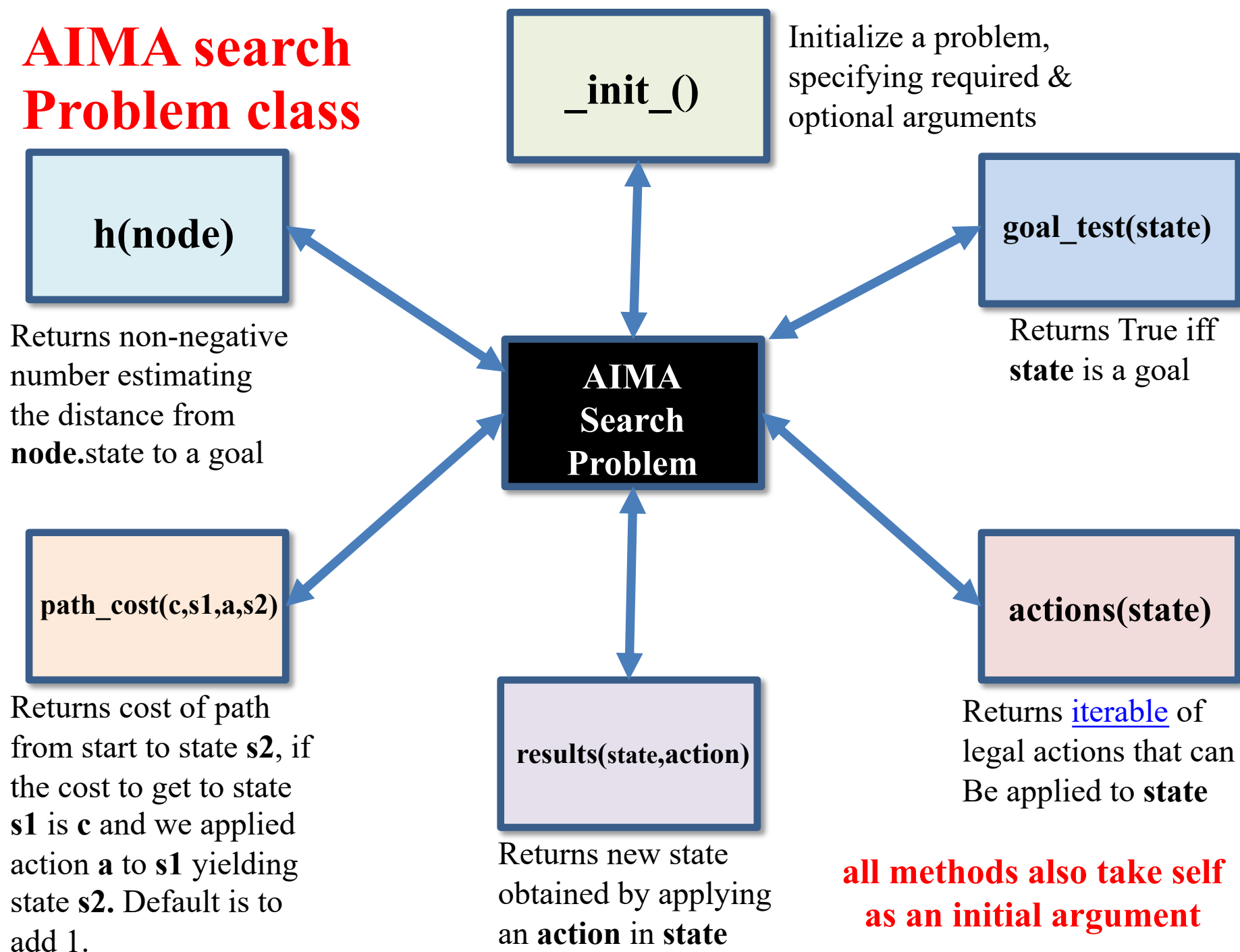


- Given two water jugs, $J1$ and $J2$, with capacities $C1$ and $C2$ and initial amounts $W1$ and $W2$, find actions to end up with amounts $W1'$ and $W2'$ in the jugs
- Example problem:
 - We have a 5-gallon and 2-gallon jug
 - Initially both are full
 - We want to end up with exactly one gallon in $J2$ and don't care how much is in $J1$

AIMA's search.py

- Defines a *Problem* class for a search problem
- Has functions to do various kinds of search given an instance of a *Problem*, e.g., BFS, DFS, & more
- *InstrumentedProblem* subclasses *Problem* and is used with *compare_searchers* for evaluation
- To use for WJP:
 1. Decide how to represent it (i.e., state, actions, goal);
 2. Define *WJP* as a subclass of *Problem*; and
 3. Provide methods to (a) create a WJP instance, (b) compute state successors, and (c) test for a goal

AIMA search Problem class



Example: Water Jug Problem



Given full 5-gal. jug
and empty 2-gal. jug,
fill 2-gal jug with one
gallon

- State = (x,y) , where x is water in jug 1; y is water in jug 2
- Initial State = $(5,0)$
- Goal State = $(-1,1)$, where -1 means any amount

Action table

Name	Cond.	Transition	Effect
dump1	$x > 0$	$(x,y) \rightarrow (0,y)$	Empty Jug 1
dump2	$y > 0$	$(x,y) \rightarrow (x,0)$	Empty Jug 2
pour_1_2	$x > 0$ & $y < C2$	$(x,y) \rightarrow (x-D, y+D)$ $D = \min(x, C2-y)$	Pour from Jug 1 to Jug 2
pour_2_1	$y > 0$ & $X < C1$	$(x,y) \rightarrow (x+D, y-D)$ $D = \min(y, C1-x)$	Pour from Jug 2 to Jug 1

Our WJ problem class



```
class WJ(Problem):
```

```
    def __init__(self, capacities=(5,2), initial=(5,0), goal=(0,1)):
```

```
        self.capacities = capacities
```

```
        self.initial = initial
```

```
        self.goal = goal
```

```
    def goal_test(self, state): # returns True iff state is a goal state
```

```
        g = self.goal          # -1 is a don't care
```

```
        return (state[0] == g[0] or g[0] == -1 ) and
```

```
                (state[1] == g[1] or g[1] == -1)
```

```
    def __repr__(self): # returns string representing the object
```

```
        return f"WJ({self.capacities},{self.initial},{self.goal})"
```

Note: [f-string](#)

Returns possible actions in state

```
def actions(self, state):  
    (J1, J2) = state  
    (C1, C2) = self.capacities  
    if J1>0: yield(('dump', 1, 0))  
    if J2>0: yield(('dump', 2, 0))  
    if J2<C2 and J1>0: yield(('pour', 1, 2))  
    if J1<C1 and J2>0: yield(('pour', 2, 1))
```

Note: we represent an action as a tuple of its name and arguments, e.g.

- *(dump, 1, 0)*
- *(pour 2, 1)*

yield? If you're unfamiliar with yield and Python generators, see [this](#).

```
def result(self, state, action):
```

```
    # Given state and action, returns successor
```

```
    # after doing action
```

```
    act, arg1, arg2 = action
```

```
    (J1, J2), (C1, C2) = state, self.capacities
```

```
    if act == 'dump':
```

```
        return (0, J2) if arg1 == 1 else (J1, 0)
```

```
    elif act == 'pour':
```

```
        if arg1 == 1:
```

```
            delta = min(J1, C2-J2)
```

```
            return (J1-delta, J2+delta)
```

```
        else:
```

```
            delta = min(J2, C1-J1)
```

```
            return (J1+delta, J2-delta)
```

```
        else:
```

```
            raise ValueError(f"Unknown action: {action}")
```

Result returns successor state

*Note: the AIMA code will call this for **each possible action** that can be done in a state*

So, we don't need to check if the action is possible in the state

Our WJ problem class

```
def h(self, node):
```

```
    # heuristic function that estimates distance  
    # from node to a goal node
```

```
    return 0 if self.goal_test(node.state) else 1
```

Note: this is only useful for informed search algorithms

For uninformed algorithms, we don't worry about finding a least costly path

So, this heuristic just returns 0 if node is a goal node and 1 for anything else

Solving a WJP

```
code> python
```

```
>>> from wj import *
```

```
# Import wj.py and search.py
```

```
>>> from search import *
```

```
>>> p1 = WJ((5,2), (5,2), (-1, 1))
```

```
# Create a problem instance
```

```
>>> p1
```

```
WJ((5, 2),(5, 2),(-1, 1))
```

```
>>> answer = breadth_first_graph_search(p1)
```

```
# Used the breadth 1st search function
```

```
>>> answer
```

```
# Will be None if the search failed or a
```

```
<Node (0, 1)>
```

```
# a goal node in the search graph if successful
```

```
>>> answer.path_cost
```

```
# The cost to get to every node in the search graph
```

```
6
```

```
# is maintained by the search procedure
```

```
>>> path = answer.path()
```

```
# A node's path is the best way to get to it from
```

```
>>> path
```

```
# the start node, i.e., a solution
```

```
[<Node (5, 2)>, <Node (5, 0)>, <Node (3, 2)>, <Node (3, 0)>, <Node (1, 2)>, <Node (1, 0)>, <Node (0, 1)>]
```

Comparing Search Algorithms Results

Uninformed searches: breadth_first_tree_search, breadth_first_search, depth_first_graph_search, iterative_deepening_search, depth_limited_search

- All but depth_limited_search are **sound** (i.e., solutions found are correct)
- Not all are **complete** (i.e., can find all solutions)
- Not all are **optimal** (find best possible solution)
- Not all are **efficient**
- AIMA code has a comparison function

Comparing Search Algorithms Results

```
HW2> python
```

```
>>> from wj import *
```

```
>>> searchers=[breadth_first_graph_search, depth_first_graph_search,  
iterative_deepening_search]
```

```
>>> compare_searchers([WJ((5,2), (5,0), (0,1))], ['SEARCH ALGORITHM',  
'successors/goal tests/states generated/solution'], searchers)
```

```
SEARCH ALGORITHM      successors/goal tests/states generated/solution
```

```
breadth_first_graph_search < 8 / 9 / 16 / (0, >
```

```
depth_first_graph_search < 5 / 6 / 12 / (0, >
```

```
iterative_deepening_search < 35 / 61 / 57 / (0, >
```

```
>>>
```

The Output

```
hhw2> python wjtest.py -s 5 0 -g 0 1
```

```
Solving WJ((5, 2),(5, 0),(0, 1)
```

```
breadth_first_tree_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
```

```
breadth_first_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
```

```
depth_first_graph_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
```

```
iterative_deepening_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
```

```
astar_search cost 5: (5, 0) (3, 2) (3, 0) (1, 2) (1, 0) (0, 1)
```

```
SUMMARY: successors/goal tests/states generated/solution
```

```
breadth_first_tree_search < 25/ 26/ 37/(0, >
```

```
breadth_first_graph_search < 8/ 9/ 16/(0, >
```

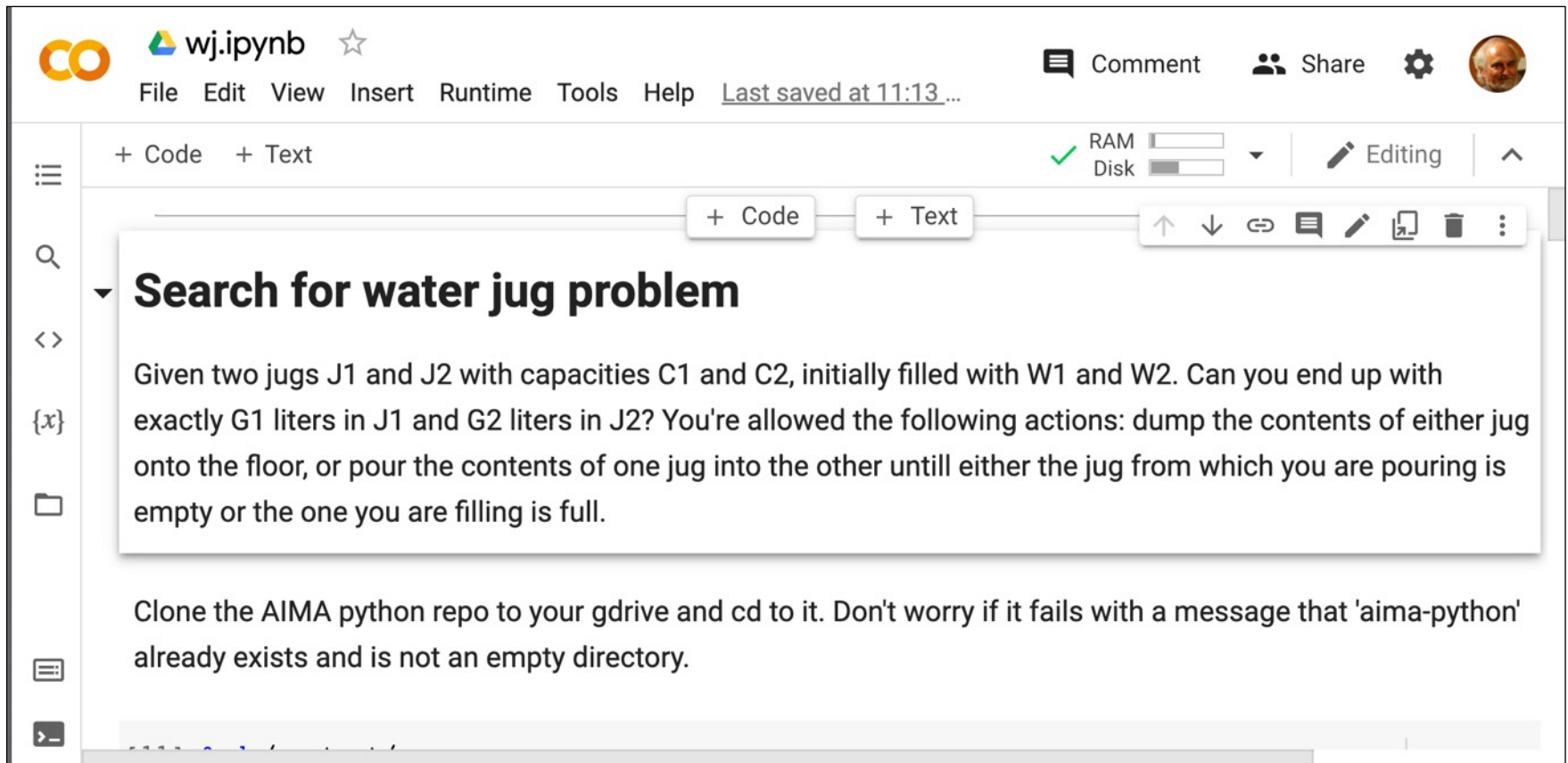
```
depth_first_graph_search < 5/ 6/ 12/(0, >
```

```
iterative_deepening_search < 35/ 61/ 57/(0, >
```

```
astar_search < 8/ 10/ 16/(0, >
```

Water Jug Problem on Colab

- See our collection of [AI notebooks on Colab](#) and the [code and data](#) in our repo
- [wj.ipynb](#) which uses [search.py](#)



The screenshot shows a Google Colab notebook titled 'wj.ipynb'. The interface includes a top navigation bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help' menus, along with 'Comment', 'Share', and 'Settings' options. Below the navigation bar, there are tabs for '+ Code' and '+ Text'. The main content area displays a search result for the 'Water Jug Problem'. The search result is titled 'Search for water jug problem' and contains the following text: 'Given two jugs J1 and J2 with capacities C1 and C2, initially filled with W1 and W2. Can you end up with exactly G1 liters in J1 and G2 liters in J2? You're allowed the following actions: dump the contents of either jug onto the floor, or pour the contents of one jug into the other until either the jug from which you are pouring is empty or the one you are filling is full.' Below the search result, there is a note: 'Clone the AIMA python repo to your gdrive and cd to it. Don't worry if it fails with a message that 'aima-python' already exists and is not an empty directory.'