Advanced AI Lab Manual

# Table of Contents

1. Experiment 1: Python Frameworks Tutorial (Jupyter & Colab) .......................................... 2
2. Experiment 2: Uninformed Search in Graph-Based Problem Spaces .......................................... 3
3. Experiment 3: Informed Search (Vacuum World & Maze) .......................................... 4
4. Experiment 4: Uninformed & Informed Search (PAC-MAN) .......................................... 5
5. Experiment 5: Multi-Agent Search in a Grid .......................................... 6
6. Experiment 6: Logical Agents & Knowledge Representation (Prolog) .......................................... 7
7. Experiment 7: Reasoning under Uncertainty (Bayesian Learning) .......................................... 8
8. Experiment 8: Reinforcement Learning (Q-Learning) .......................................... 9
9. Experiment 9: Introduction to Machine Learning & Python Data Libraries .......................................... 10

# Experiment 1: Python Frameworks Tutorial (Jupyter & Colab)

## Aim / Objective

Get comfortable with Jupyter and Google Colab; explore Python data structures.

## Theory

Jupyter and Colab provide interactive Python environments. Key data structures: list, dict, set, tuple.

## Procedure

Launch notebook, experiment with creating/appending lists, dict operations, set membership, tuple access.

## Code

# Jupyter/Colab cells  
L = [1,2,3]; L.append(4)  
D = {'a':1,'b':2}  
S = set([1,2,3]); S.add(4)  
T = (10,20,30)

## Sample Output

List: [1, 2, 3, 4], Dict: {...}, Set: {...}, Tuple: (...).

## Conclusion

Basic Python structures and interactive notebooks mastered.

# Experiment 2: Uninformed Search in Graph-Based Problem Spaces

## Aim / Objective

Implement and compare Breadth-First Search (BFS) and Depth-First Search (DFS) on a graph.

## Theory

Uninformed search explores without heuristics: BFS (level order), DFS (depth first).

## Procedure

Represent graph as adjacency list, implement BFS/DFS, run from start to goal, record results.

## Code

def bfs(...): ...  
def dfs(...): ...  
# See lab code above for details

## Sample Output

BFS A→F: ['A','C','F'], DFS A→F: ['A','B','E','F'].

## Conclusion

BFS finds shortest paths; DFS explores deeply; cycle checking is needed.

# Experiment 3: Informed Search (Vacuum World & Maze)

## Aim / Objective

Implement A\* search with Manhattan heuristic on Vacuum-World and Maze problems.

## Theory

A\* uses g(n) + h(n). Manhattan distance is admissible heuristic in grids.

## Procedure

Model grid, code A\* function, run on vacuum and maze instances, observe path and node expansions.

## Code

import heapq  
def astar(start,goal,grid): ...  
# See lab code above

## Sample Output

Path: [(0,0),(1,0),(2,0),(2,1),(2,2),(3,2),(3,3)].

## Conclusion

A\* efficiently finds optimal paths using heuristics.

# Experiment 4: Uninformed & Informed Search (PAC-MAN)

## Aim / Objective

Apply BFS, DFS, and A\* in the PAC-MAN maze to find path to pellet.

## Theory

Maze pathfinding: compare uninformed vs informed search in Pac-Man maze.

## Procedure

Load Pac-Man grid, apply BFS, DFS, A\*; compare results and performance metrics.

## Code

# Use BFS, DFS from Exp2 and A\* from Exp3 on pacman grid

## Sample Output

BFS/DFS/A\*: same path in Pac-Man example.

## Conclusion

In simple mazes all strategies may match; A\* scales better on larger problems.

# Experiment 5: Multi-Agent Search in a Grid

## Aim / Objective

Simulate multiple agents solving individual A\* paths while avoiding collisions.

## Theory

Each agent re-plans, treating others as dynamic obstacles for collision avoidance.

## Procedure

Generate random grid with obstacles, place agents, loop: plan paths avoiding collisions, move one step, visualize.

## Code

class Agent: ...  
def plan\_path(self,...): ...  
# See multi-agent code above

## Sample Output

Grid state with emojis per step.

## Conclusion

Dynamic re-planning yields collision-free multi-agent coordination.

# Experiment 6: Logical Agents & Knowledge Representation (Prolog)

## Aim / Objective

Learn Prolog: define facts/rules, query KB, trace resolution.

## Theory

Prolog uses Horn clauses; SLD-resolution solves queries via backtracking.

## Procedure

Install SWI-Prolog, write family.pl facts and rules, consult, run example queries, draw search trees.

## Code

% Prolog code in family.pl  
parent(alice,bob).  
grandparent(X,Z):- parent(X,Y),parent(Y,Z).

## Sample Output

?- grandparent(alice,Z). Z = carol.

## Conclusion

Prolog’s declarative paradigm validated via backtracking resolution.

# Experiment 7: Reasoning under Uncertainty (Bayesian Learning)

## Aim / Objective

Implement Naïve Bayes classification and Bayesian parameter estimation.

## Theory

Bayes’ theorem: P(H|D) ∝ P(D|H)P(H). Naïve independence assumption.

## Procedure

Load dataset, compute class priors and likelihoods, classify new instance, validate results.

## Code

import pandas as pd  
data = pd.DataFrame(...)  
# Compute priors and likelihoods

## Sample Output

Priors: {...}, Likelihood: {...}, Posteriors: {...}.

## Conclusion

Naïve Bayes effective on small data; smoothing avoids zero-frequency issues.

# Experiment 8: Reinforcement Learning (Q-Learning)

## Aim / Objective

Implement Q-Learning on a gridworld to learn optimal policy.

## Theory

Q-learning: off-policy TD control: Q(s,a) update rule: Q ← Q + α(r+γmaxQ−Q).

## Procedure

Define gridworld, initialize Q-table, run episodes with ε-greedy policy, update Q, evaluate policy.

## Code

import numpy as np  
Q = np.zeros((4,4,4))  
# Q-learning update loop

## Sample Output

Learned Q-values table (converged).

## Conclusion

Q-learning learns optimal policies; balance exploration and exploitation.

# Experiment 9: Introduction to Machine Learning & Python Data Libraries

## Aim / Objective

Familiarize with Pandas, NumPy, Matplotlib for data loading, manipulation, visualization.

## Theory

NumPy arrays, Pandas DataFrame, Matplotlib plotting for data analysis.

## Procedure

Load CSV into DataFrame, use .describe(), plot histograms and scatter plots with Matplotlib.

## Code

import pandas as pd, matplotlib.pyplot as plt  
df = pd.read\_csv('iris.csv')  
df.describe()  
df.plot(...)

## Sample Output

Summary and plots displayed.

## Conclusion

Pandas, NumPy, and Matplotlib streamline data analysis and visualization.