DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

II B.Tech I Semester

Subject Name: APPLICATIONS OF ARTIFICIAL INTELLIGENCE LAB
Subject Code: C6602
Regulations: MR-22

Lab Manual

Academic Year: 2024-25

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)
MAIN CAMPUS

(An UGC Autonomous Institution, Approved by AICTE and Affiliated to JNTUH,
Hyderabad, Accredited by NAAC with 'A++' Grade (III Cycle))
NBA Accredited Programmes – UG (CE, EEE, ME, ECE, & CSE), PG (CE-SE, EEE, EPS, ME-TE)
Maisammaguda(H), Gundlapochampally Village, Medchal Mandal,
Medchal-Malkajgiri District, Telangana State – 500100

MALLA REDDY ENGINEERING COLLEGE (AUTONOMOUS)

MR22 – ACADEMIC REGULATIONS (CBCS)

for B.Tech. (REGULAR) DEGREE PROGRAMME

Applicable for the students of B.Tech. (Regular) programme admitted from the Academic Year 2022-23 onwards

The B.Tech. Degree of Jawaharlal Nehru Technological University Hyderabad, Hyderabad shall be conferred on candidates who are admitted to the programme and who fulfill all the requirements for the award of the Degree.

VISION OF THE INSTITUTE

To be a premier center of professional education and research, offering quality programs in a socio-economic and ethical ambience.

MISSION OF THE INSTITUTE

- To impart knowledge of advanced technologies using state-of-the-art infrastructural facilities.
- To inculcate innovation and best practices in education, training and research.
- To meet changing socio-economic needs in an ethical ambience.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT VISION

To attain global standards in Computer Science and Engineering education, training and research to meet the growing needs of the industry with socio-economic and ethical considerations.

DEPARTMENT MISSION

- To impart quality education and research to undergraduate and postgraduate students in Computer Science and Engineering.
- To encourage innovation and best practices in Computer Science and Engineering utilizing state-of-the-art facilities.
- To develop entrepreneurial spirit and knowledge of emerging technologies based on ethical values and social relevance.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will demonstrate technical skills, competency in AI & ML and exhibit team management capability with proper communication in a job environment

PEO2: Graduates will function in their profession with social awareness and responsibility

PEO3: Graduates will interact with their peers in other disciplines in industry and society and contribute to the economic growth of the country

PEO4: Graduates will be successful in pursuing higher studies in engineering or management

PROGRAMME OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

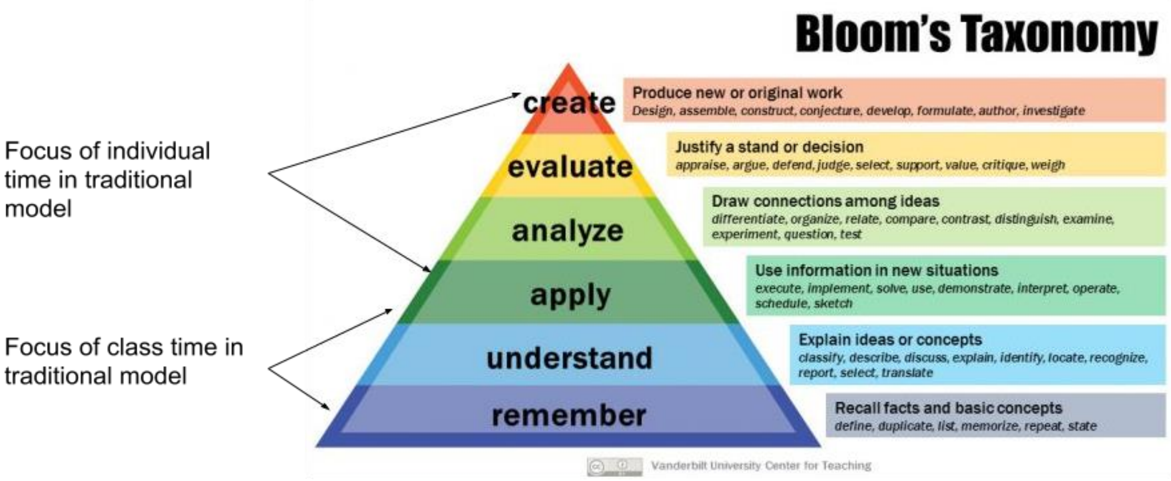
PROGRAMME SPECIFIC OUTCOMES (PSOs)

PSO1: Design and develop intelligent automated systems applying mathematical, analytical, programming and operational skills to solve real world problems

PSO2: Apply machine learning techniques, software tools to conduct experiments, interpret data and to solve complex problems

PSO3: Implement engineering solutions for the benefit of society by the use of AI and ML

BLOOM’S TAXONOMY (BT) TRIANGLE & BLOOM’S ACTION VERBS



Bloom’s Taxonomy

Focus of individual time in traditional model

Focus of class time in traditional model

create — Produce new or original work
Design, assemble, construct, conjecture, develop, formulate, author, investigate

evaluate — Justify a stand or decision
appraise, argue, defend, judge, select, support, value, critique, weigh

analyze — Draw connections among ideas
differentiate, organize, relate, compare, contrast, distinguish, examine, experiment, question, test

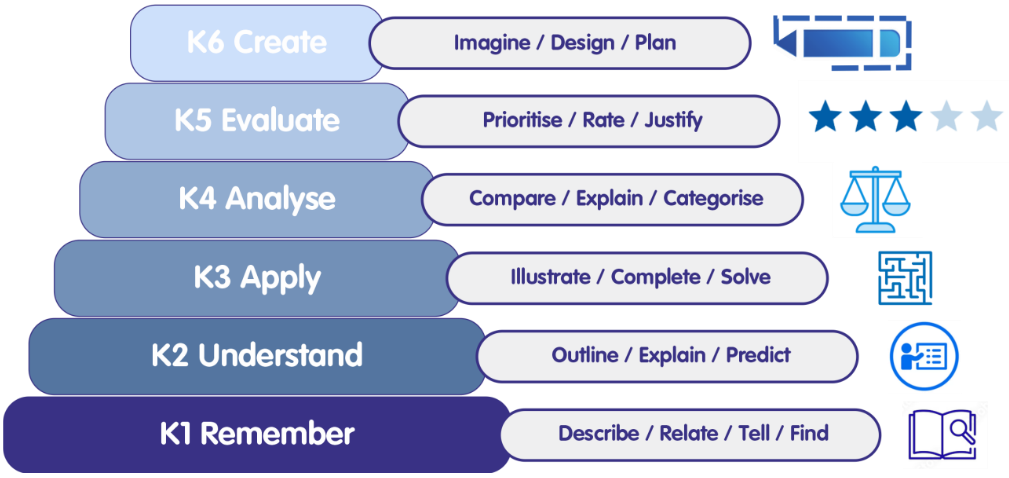
apply — Use information in new situations
execute, implement, solve, use, demonstrate, interpret, operate, schedule, sketch

understand — Explain ideas or concepts
classify, describe, discuss, explain, identify, locate, recognize, report, select, translate

remember — Recall facts and basic concepts
define, duplicate, list, memorize, repeat, state

Vanderbilt University Center for Teaching

BLOOM’S TAXONOMY



K6 Create — Imagine / Design / Plan

K5 Evaluate — Prioritise / Rate / Justify

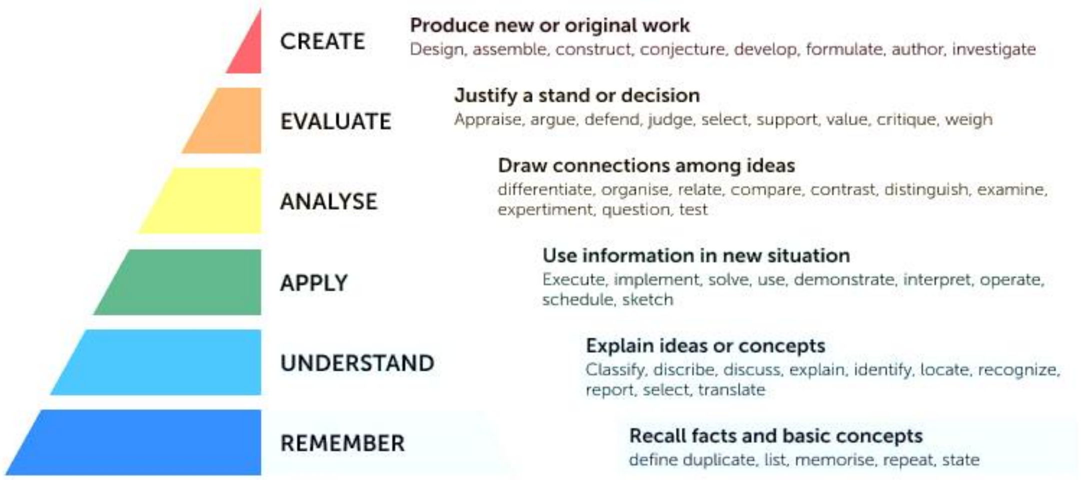
K4 Analyse — Compare / Explain / Categorise

K3 Apply — Illustrate / Complete / Solve

K2 Understand — Outline / Explain / Predict

K1 Remember — Describe / Relate / Tell / Find

Bloom’s Taxonomy



CREATE — **Produce new or original work**
Design, assemble, construct, conjecture, develop, formulate, author, investigate

EVALUATE — **Justify a stand or decision**
Appraise, argue, defend, judge, select, support, value, critique, weigh

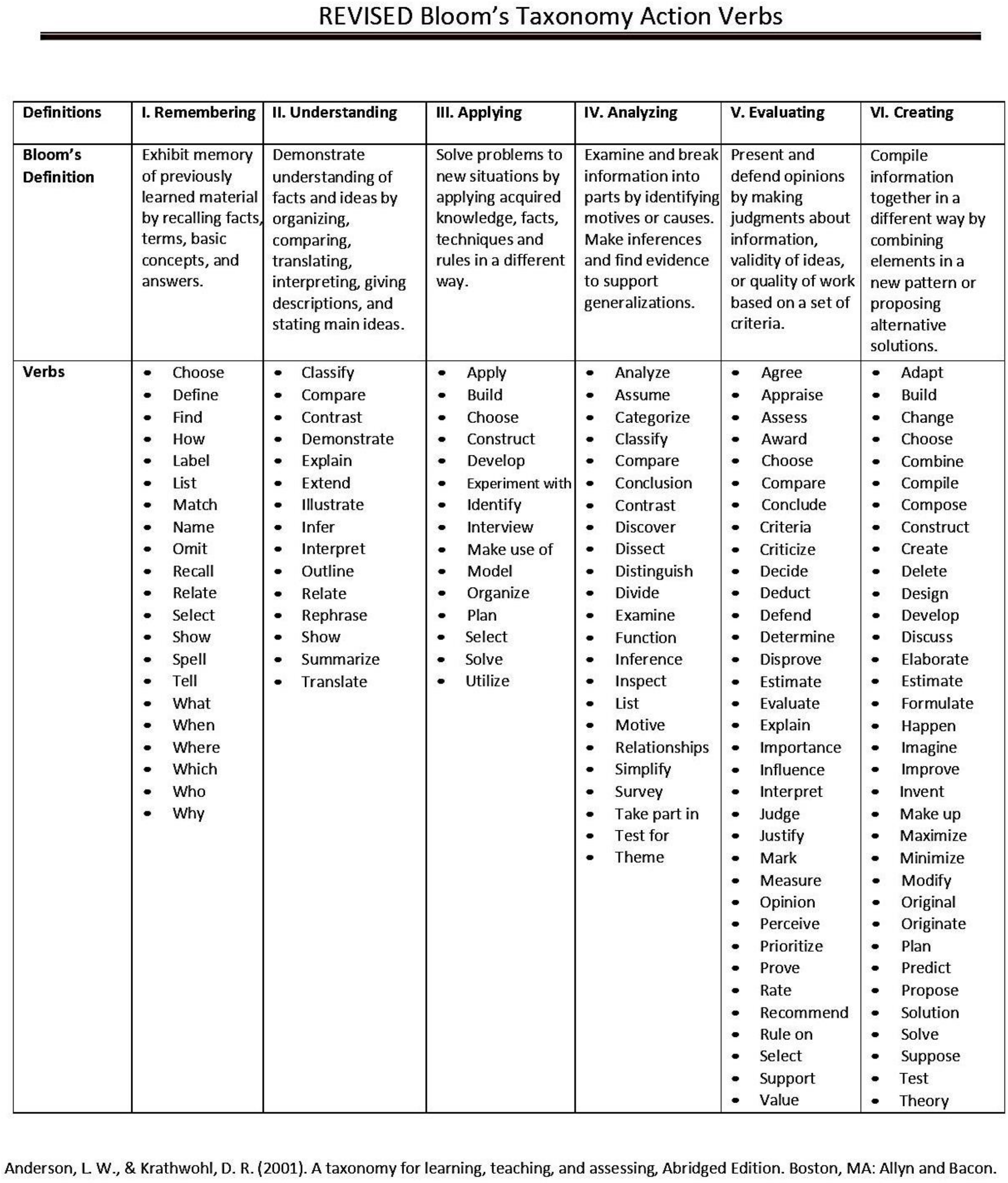
ANALYSE — **Draw connections among ideas**
differentiate, organise, relate, compare, contrast, distinguish, examine, expertiment, question, test

APPLY — **Use information in new situation**
Execute, implement, solve, use, demonstrate, interpret, operate, schedule, sketch

UNDERSTAND — **Explain ideas or concepts**
Classify, discribe, discuss, explain, identify, locate, recognize, report, select, translate

REMEMBER — **Recall facts and basic concepts**
define duplicate, list, memorise, repeat, state

BLOOM'S ACTION VERBS



REVISED Bloom's Taxonomy Action Verbs

Definitions	I. Remembering	II. Understanding	III. Applying	IV. Analyzing	V. Evaluating	VI. Creating
Bloom's Definition	Exhibit memory of previously learned material by recalling facts, terms, basic concepts, and answers.	Demonstrate understanding of facts and ideas by organizing, comparing, translating, interpreting, giving descriptions, and stating main ideas.	Solve problems to new situations by applying acquired knowledge, facts, techniques and rules in a different way.	Examine and break information into parts by identifying motives or causes. Make inferences and find evidence to support generalizations.	Present and defend opinions by making judgments about information, validity of ideas, or quality of work based on a set of criteria.	Compile information together in a different way by combining elements in a new pattern or proposing alternative solutions.
Verbs	• Choose • Define • Find • How • Label • List • Match • Name • Omit • Recall • Relate • Select • Show • Spell • Tell • What • When • Where • Which • Who • Why	• Classify • Compare • Contrast • Demonstrate • Explain • Extend • Illustrate • Infer • Interpret • Outline • Relate • Rephrase • Show • Summarize • Translate	• Apply • Build • Choose • Construct • Develop • Experiment with • Identify • Interview • Make use of • Model • Organize • Plan • Select • Solve • Utilize	• Analyze • Assume • Categorize • Classify • Compare • Conclusion • Contrast • Discover • Dissect • Distinguish • Divide • Examine • Function • Inference • Inspect • List • Motive • Relationships • Simplify • Survey • Take part in • Test for • Theme	• Agree • Appraise • Assess • Award • Choose • Compare • Conclude • Criteria • Criticize • Decide • Deduct • Defend • Determine • Disprove • Estimate • Evaluate • Explain • Importance • Influence • Interpret • Judge • Justify • Mark • Measure • Opinion • Perceive • Prioritize • Prove • Rate • Recommend • Rule on • Select • Support • Value	• Adapt • Build • Change • Choose • Combine • Compile • Compose • Construct • Create • Delete • Design • Develop • Discuss • Elaborate • Estimate • Formulate • Happen • Imagine • Improve • Invent • Make up • Maximize • Minimize • Modify • Original • Originate • Plan • Predict • Propose • Solution • Solve • Suppose • Test • Theory

Anderson, L. W., & Krathwohl, D. R. (2001). A taxonomy for learning, teaching, and assessing, Abridged Edition. Boston, MA: Allyn and Bacon.

2022-23 Onwards (MR-22)	**MALLA REDDY ENGINEERING COLLEGE** **(AUTONOMOUS)**	**B.Tech. VI Semester**		
Code: C6602	**APPLICATIONS OF ARTIFICIAL INTELLIGENCE LAB**	**L**	**T**	**P**
Credits: 1.5		**-**	**-**	**3**

List of Experiments:

1. Write a program to conduct uninformed search.
2. Write a program to conduct informed search.
3. Write a program to conduct game search.
4. Write a program to construct a Bayesian network from given data.
5. Write a program to infer from the Bayesian network.
6. Write a program to illustrate Hidden Markov Model.
7. Write a program to run value and policy iteration in a grid world.
8. Write a program to do reinforcement learning in a grid world.
9. Write a program to implement adaptive dynamic programming.
10. Write a program to implement active dynamic programming.
11. Write a program to implement Q learning.
12. Case Study

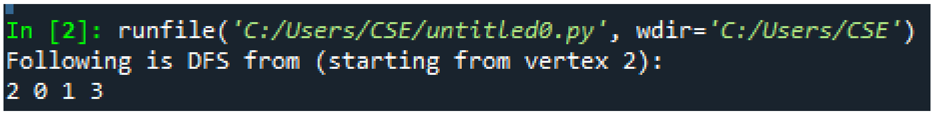
CO- PO, PSO Mapping (3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak															
COs	**Programme Outcomes (POs)**											**PSOs**			
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1	2	3	1									2	1		
CO2	2	2										2	2		
CO3	1	2										1	1		

1. Write a program to conduct uninformed search.

DFS(depth first search)

```
        from collections import defaultdict

class Graph:
    # Constructor
    def __init__(self):
        # Default dictionary to store the graph
        self.graph = defaultdict(list)

    # Function to add an edge to the graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # A function used by DFS
    def DFSUtil(self, v, visited):
        # Mark the current node as visited and print it
        visited.add(v)
        print(v, end=' ')

        # Recur for all the vertices adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses recursive DFSUtil()
    def DFS(self, v):
        # Create a set to store visited vertices
        visited = set()
        # Call the recursive helper function to print DFS traversal
        self. DFSUtil(v, visited)

# Driver code
# Create a graph given in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2):")
g.DFS(2)
```

Output:



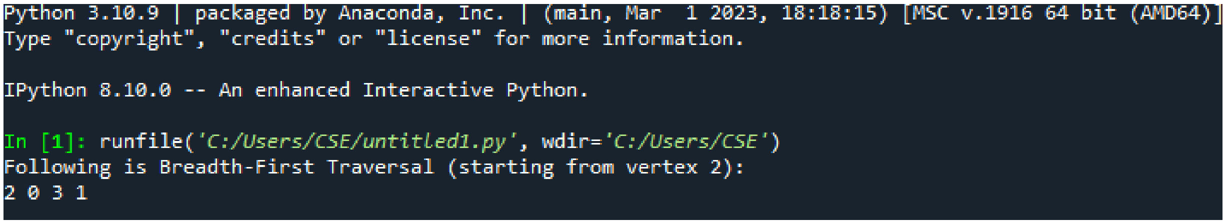
```
In [2]: runfile('C:/Users/CSE/untitled0.py', wdir='C:/Users/CSE')
Following is DFS from (starting from vertex 2):
2 0 1 3
```

#Breadth first search

```
from collections import defaultdict

class Graph:
    # Constructor
    def __init__(self):
        # Default dictionary to store the graph
        self.graph = defaultdict(list)

    # Function to add an edge to the graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # Function to print a BFS of the graph
    def BFS(self, s):
        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:
            # Dequeue a vertex from the queue and print it
            s = queue.pop(0)
            print(s, end=" ")

            # Get all adjacent vertices of the dequeued vertex s.
            # If an adjacent has not been visited, then mark it visited and enqueue it
            for i in self.graph[s]:
                if not visited[i]:
                    queue.append(i)
                    visited[i] = True
```

```
 # Driver code
 # Create a graph given in the above diagram
 g = Graph()
 g.addEdge(0, 1)
 g.addEdge(0, 2)
 g.addEdge(1, 2)
 g.addEdge(2, 0)
 g.addEdge(2, 3)
 g.addEdge(3, 3)

 print("Following is Breadth-First Traversal (starting from vertex 2):")
g.BFS(2)
```

Output:



```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/CSE/untitled1.py', wdir='C:/Users/CSE')
Following is Breadth-First Traversal (starting from vertex 2):
2 0 3 1
```

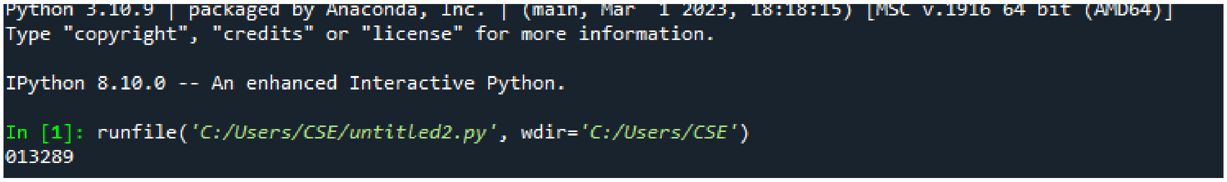
2.program to conduct informed search.

#BestFirstSearch

```
from queue import PriorityQueue


v = 14

graph = [[] for _ in range(v)]


# Function for Implementing Best-First Search

# Gives output path having the lowest cost

def best_first_search(source, target, n):

    visited = [0] * n

    visited[source] = True

    pq = PriorityQueue()

    pq.put((0, source))
    while not pq.empty():
        u = pq.get()[1]
        # Displaying the path having the lowest cost
        print(u, end="")
        if u == target:
            break

        for v, c in graph[u]:
            if not visited[v]:
                visited[v] = True
                pq.put((c, v))
    print()

# Function for adding edges to graph
def add_edge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

# The nodes shown in the above example (by alphabets) are implemented using integers
add_edge(0, 1, 3)
add_edge(0, 2, 6)
add_edge(0, 3, 5)
add_edge(1, 4, 9)
add_edge(1, 5, 8)
add_edge(2, 6, 12)
add_edge(2, 7, 14)
add_edge(3, 8, 7)
add_edge(8, 9, 5)
add_edge(8, 10, 6)
add_edge(9, 11, 1)
add_edge(9, 12, 10)
add_edge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

Output:

013289



```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/CSE/untitled2.py', wdir='C:/Users/CSE')
013289
```

3.program to conduct gamesearch

```
# Tic-Tac-Toe Program

# importing all necessary

librariesimport numpy as np
```

```
Import random

From time import sleep

# Creates an empty

boarddefcreate_board():

    return(np.array([[0,0,0],

                     [0, 0,0],

                     [0,0,0]]))

#Checkforemptyplacesonboarddefpossibili

ties(board):l=[]

    for i in

        range(len(board)):forjinrange(l

        en(board)):

            if board[i][j] ==

                0:l.append((i,j))

    return(l)


# Select a random place for the

playerdefrandom_place(board, player):

    selection = possibilities(board)current_loc =

    random.choice(selection)board[current_loc]

    = playerreturn(board)
```

```
# Checks whether the player has three# of
their marks in a horizontal
rowdefrow_win(board, player):
    for x in range(len(board)):True
    for y in
        range(len(board)):ifboard[x,y
        ] !=player:
                win=Falsecontinue
        ifwin==True:return(win)
    return(win)
# Checks whether the player has three#of
their marksin avertical row
def col_win(board,
    player):forxinrange(len(board)):
        win=True
        for y in
            range(len(board)):ifboard[y][x
            ]!=player:
                win=Falsecontinue
        ifwin==True:return(win)
    return(win)
```

```
# Checks whether the player has three#of

their marks inadiagonal row

def diag_win(board,

    player):win=True

    y =0

    for x in

        range(len(board)):ifboard[x,x

        ] !=player:

            win=Falseifwin:

        return

    winwin=Trueifwi

    n:

        for x in

            range(len(board)):y=len(boar

            d)-1-x

            if board[x, y] !=

                player:win=False

    return win
# Evaluates whether there

is#awinner or atie

def
```

```
    evaluate(board):winner

    =0

    forplayerin [1,2]:

        if (row_win(board, player)

            orcol_win(board,player)

            ordiag_win(board,player)):

            winner=player

    if np.all(board != 0) and

        winner==0:winner=-1

    return winner

# Main function to start the

gamedefplay_game():

    board, winner, counter = create_board(), 0,

    1print(board)

    sleep(2)

    while winner ==

        0:forplayerin[1,2]:

            board = random_place(board, player)print("Board

            after " + str(counter) + " move")print(board)

            sleep(2)counter+=1

            winner = evaluate(board)if
            winner!=0:

                breakreturn(

    winner)

#DriverCode

print("Winneris:"+str(play_game()))
```

Output:

```
[[000]
 [0 00]
 [00 0]]
Board after 1
move[[000]
 [0 00]
 [10 0]]
Board after 2
move[[000]
 [0 20]
 [10 0]]
Board after 3
move[[010]
 [0 20]
  [10 0]]
 Boardafter4
 move[[0 10]
  [2 20]
  [10 0]]
 Boardafter5
 move[[1 10]
  [2 20]
  [10 0]]
```

```
Boardafter6
move[[1 10]
 [2 20]
 [12 0]]
Boardafter7
move[[1 10]
 [2 20]
 [12 1]]
Boardafter8
move[[1 10]
 [2 22]
 [12 1]]
Winneris: 2
```

4.Write a program toconstructa Bayesiannetwork from givendata.

1. age:ageinyears
2. sex:sex(1=male;0= female)
3. cp:chestpaintype
   Value 1: typical
   anginaValue 2: atypical
   anginaValue3:non-
   anginalpainValue4:asym
   ptomatic
4. trestbps:restingblood pressure(in mmHg onadmission tothehospital)
5. chol:serumcholestoral inmg/dl
6. fbs:(fasting bloodsugar >120 mg/dl)(1 =true; 0= false)
7. restecg:restingelectrocardiographicresultsV

alue0: normal
Value1:havingST-
Twaveabnormality(Twaveinversionsand/orSTelevationordepression
of>0.05mV)
Value2:showingprobableordefiniteleftventricularhypertrophybyEstes'criteria
8. thalach:maximumheartrateachieved
9. exang:exercise induced angina(1 =yes;0 = no)
10. oldpeak=STdepressioninducedbyexerciserelativetorest11.sl
ope:theslope ofthepeak exercise ST segment
Value1:upsloping
Value2:flat
Value3:downsloping
12. ca=number ofmajorvessels(0-3) colored byflourosopy
13. thal:3= normal;6=fixeddefect;7 =reversabledefect
14.Heartdisease: Itisinteger valued
from0(nopresence)to4.Diagnosisofheartdisease(angiographicdiseasestatus)
Someinstancefromthedataset:
Age sex cp trestbps chol fbs restecg thalach exang oldpeakslopecathal Heartdisease
63 1 1 145 233 1 2 150 0 2.3 3 0 6 0
67 1 4 160286 0 2 108 1 1.5 2 3 3 2
67 1 4 120 229 0 2 129 1 2.6 2 2 7 1
41 0 2 130 204 0 2 172 0 1.4 1 0 3 0
62 0 4 140 268 0 2 160 0 3.6 3 2 3 3
60 1 4 130 206 0 2 132 1 2.4 2 2 7 4

Program:

```
import numpy as
npimportcsv
importpandasaspd
frompgmpy.modelsimportBayesianModel
frompgmpy.estimatorsimportMaximumLikelihoodEstimatorfr
ompgmpy.inferenceimportVariableElimination
#read Cleveland Heart Disease
dataheartDisease =
pd.read_csv('heart.csv')heartDisease=heartDis
ease.replace('?',np.nan)#displaythedata
print('Fewexamplesfromthedatasetaregivenbelow')prin
t(heartDisease.head())
#Model Bayesian
NetworkModel=BayesianModel([('age','trestbps'),('age','fbs'),
('sex','trestbps'),('exang','trestbps'),('trestbps','heartdise
ase'),('fbs','heartdisease'),('heartdisease','restecg'),
('heartdisease','thalach'),('heartdisease','chol')])#Learning
CPDsusingMaximumLikelihoodEstimators
print('\n Learning CPD using Maximum likelihood
estimators')model.fit(heartDisease,estimator=MaximumLikelihoo
dEstimator) #Inferencing with Bayesian Network
print('\nInferencing withBayesian
Network:')HeartDisease_infer =
VariableElimination(model)#computing the
Probability of HeartDisease given
Ageprint('\n1.ProbabilityofHeartDiseasegiven
Age=30')
q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'age':28})
print(q['heartdisease'])
#computing the Probability of HeartDisease given
cholesterolprint('\n 2. Probability of HeartDisease given
cholesterol=100')q=HeartDisease_infer.query(variables=['heartdis
ease'],evidence
={'chol':100})
print(q['heartdisease'])
```

Output:

```
Fewexamplesfromthedatasetaregivenbelowage
sex cp trestbps ...slope ca thal heartdisease0 63
1 1 145 ... 3 0 6 0
1 67 1 4 160 ... 2 3 3 2
2 67 1 4 120... 2 2 7 1
3 37 1 3 130 ... 3 0 3 0
4 41 0 2 130 ... 1 0 3 0
[5rowsx14columns]
Learning CPD using Maximum likelihood
estimatorsInferencingwith Bayesian Network:
1. ProbabilityofHeartDiseasegivenAge=28
```

heartdisease ∣ phi(heartdisease) ∣	
heartdisease_0 ∣ 0.6791 ∣	
heartdisease_1 ∣ 0.1212 ∣	
heartdisease_2 ∣ 0.0810 ∣	
heartdisease_3 ∣ 0.0939 ∣	
heartdisease_4 ∣ 0.0247 ∣	

2. ProbabilityofHeartDiseasegiven cholesterol=100

```
+----------------+-------------------+
| heartdisease | phi(heartdisease) |
+================+===================+
| heartdisease_0 | 0.5400 |
+----------------+-------------------+
| heartdisease_1 | 0.1533 |
+----------------+-------------------+
| heartdisease_2 | 0.1303 |
+----------------+-------------------+
| heartdisease_3 | 0.1259 |
+----------------+-------------------+
| heartdisease_4 | 0.0506 |
```

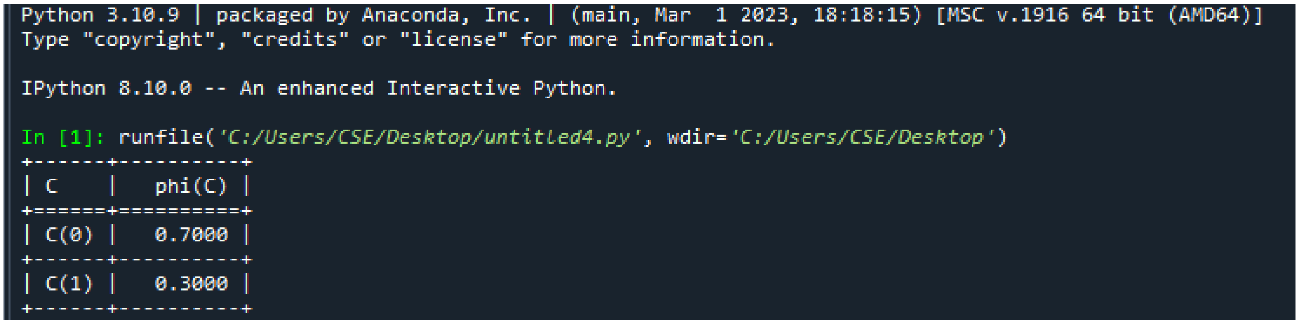
5.Write a program to infer from the Bayesian network.

```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# Define the structure of the Bayesian network
model = BayesianNetwork([('A', 'C'), ('B', 'C')])

# Define the conditional probability distributions (CPDs)
cpd_a = TabularCPD(variable='A', variable_card=2, values=[[0.6], [0.4]])
cpd_b = TabularCPD(variable='B', variable_card=2, values=[[0.7], [0.3]])
cpd_c = TabularCPD(variable='C', variable_card=2,
                   values=[[0.8, 0.9, 0.7, 0.1], [0.2, 0.1, 0.3, 0.9]],
                   evidence=['A', 'B'], evidence_card=[2, 2])

# Add CPDs to the model
model.add_cpds(cpd_a, cpd_b, cpd_c)

# Perform inference
```

```
inference = VariableElimination(model)
# Computing the probability of C given evidence for A=1 and B=0
query = inference.query(variables=['C'], evidence={'A': 1, 'B': 0})
print(query)
```

output:

```
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.7000 |
+------+----------+
| C(1) |   0.3000 |
+------+----------+
```



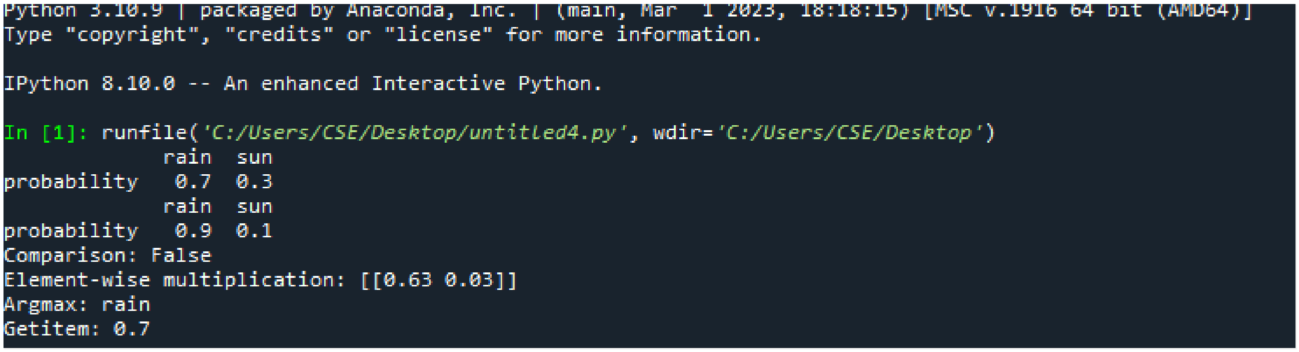
```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/CSE/Desktop/untitled4.py', wdir='C:/Users/CSE/Desktop')
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.7000 |
+------+----------+
| C(1) |   0.3000 |
+------+----------+
```

6.Write a program to illustrate HiddenMarkovModel.

```
import numpy as np

import pandas as pd



class ProbabilityVector:

    def __init__(self, probabilities: dict):

        states = probabilities.keys()
        probs = probabilities.values()

        assert len(states) == len(probs), "The probabilities must
match the states."
        assert len(states) == len(set(states)), "The states must be
unique."
        assert abs(sum(probs) - 1.0) < 1e-12, "Probabilities must
sum up to 1."
        assert len(list(filter(lambda x: 0 <= x <= 1, probs))) ==
len(probs), "Probabilities must be numbers from [0, 1]
interval."

        self.states = sorted(probabilities)
        self.values = np.array(list(map(lambda x: probabilities[x],
self.states))).reshape(1, -1)

    @classmethod
    def initialize(cls, states: list):
        size = len(states)
        rand = np.random.rand(size) / (size ** 2) + 1 / size
        rand /= rand.sum(axis=0)
        return cls(dict(zip(states, rand)))

    @classmethod
    def from_numpy(cls, array: np.ndarray, states: list):
        return cls(dict(zip(states, list(array))))

    @property
    def dict(self):
        return {k: v for k, v in zip(self.states,
list(self.values.flatten()))}

    @property
    def df(self):
        return pd.DataFrame(self.values, columns=self.states,
index=['probability'])

    def __repr__(self):
        return "P({})={}".format(self.states, self.values)

    def __eq__(self, other):
        if not isinstance(other, ProbabilityVector):
            raise NotImplementedError
        if (self.states == other.states) and (self.values ==
other.values).all():
            return True
        return False
    def __getitem__(self, state: str) -> float:
        if state not in self.states:
            raise ValueError("Requesting unknown probability
state from vector.")
        index = self.states.index(state)
        return float(self.values[0, index])

    def __mul__(self, other) -> np.ndarray:
        if isinstance(other, ProbabilityVector):
            return self.values * other.values
        elif isinstance(other, (int, float)):
            return self.values * other
        else:
            raise NotImplementedError

    def __rmul__(self, other) -> np.ndarray:
        return self.__mul__(other)

    def __matmul__(self, other) -> np.ndarray:
        if isinstance(other, ProbabilityMatrix):
            return self.values @ other.values

    def __truediv__(self, number) -> np.ndarray:
        if not isinstance(number, (int, float)):
            raise NotImplementedError
        x = self.values
        return x / number if number != 0 else x / (number + 1e-12)

    def argmax(self):
        index = self.values.argmax()
        return self.states[index]



a1 = ProbabilityVector({'rain': 0.7, 'sun': 0.3})
a2 = ProbabilityVector({'sun': 0.1, 'rain': 0.9})
print(a1.df)
print(a2.df)


print("Comparison:", a1 == a2)
print("Element-wise multiplication:", a1 * a2)
print("Argmax:", a1.argmax())
print("Getitem:", a1['rain'])
```

OUTPUT

```
rain  sun
probability   0.7  0.3
             rain  sun
probability   0.9  0.1
Comparison: False
Element-wise multiplication: [[0.63 0.03]]
Argmax: rain
Getitem: 0.7
```



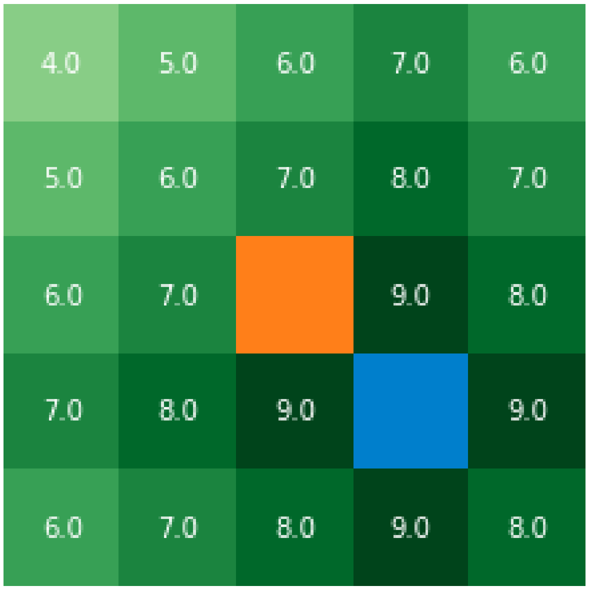
```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/CSE/Desktop/untitled4.py', wdir='C:/Users/CSE/Desktop')
             rain  sun
probability   0.7  0.3
             rain  sun
probability   0.9  0.1
Comparison: False
Element-wise multiplication: [[0.63 0.03]]
Argmax: rain
Getitem: 0.7
```

7. Write a program to run valuea nd policy iteration in agrid world.

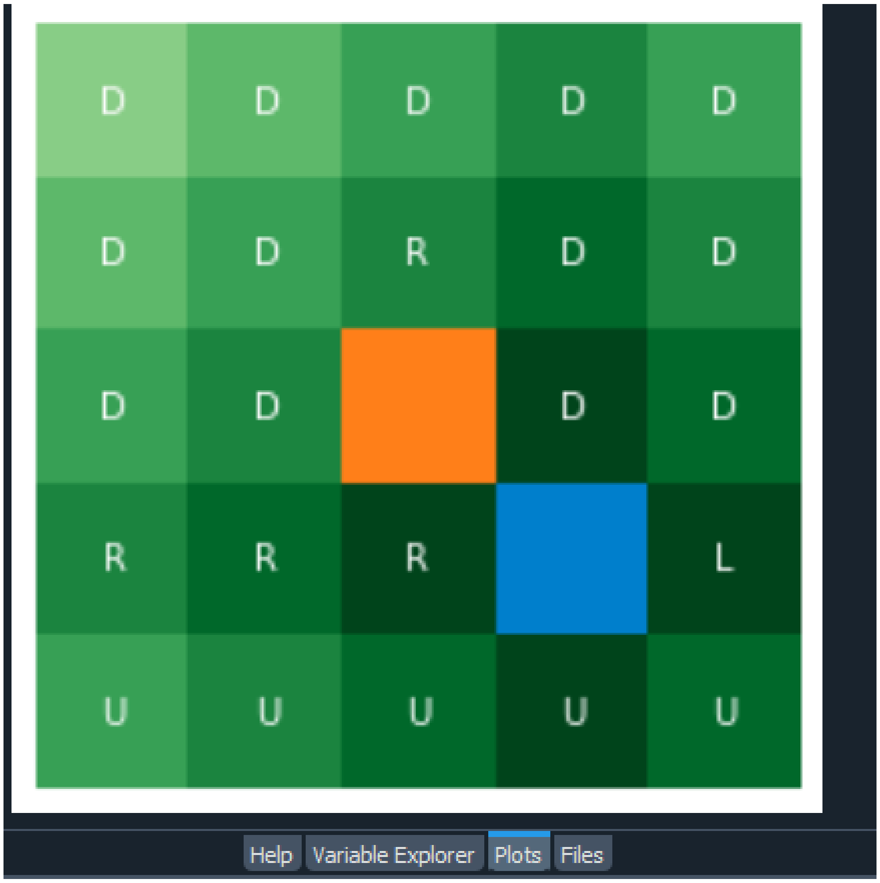
```
 import numpy as np
import matplotlib.pyplot as plt

class GridWorld(object):
    def __init__(self, gridSize, items):
        self.step_reward = -1
        self.m = gridSize[0]
        self.n = gridSize[1]
        self.grid = np.zeros(gridSize)
        self.items = items
        self.state_space = list(range(self.m * self.n))
        self.action_space = {'U': -self.m, 'D': self.m, 'L': -1, 'R': 1}
        self.actions = ['U', 'D', 'L', 'R']
        self.P = self.int_P()

    def int_P(self):
        P = {}
        for state in self.state_space:
            for action in self.actions:
                reward = self.step_reward
                n_state = state + self.action_space[action]
                if n_state in self.items.get('fire').get('loc'):
                    reward += self.items.get('fire').get('reward')
                elif n_state in self.items.get('water').get('loc'):
                    reward += self.items.get('water').get('reward')
                elif self.check_move(n_state, state):
                    n_state = state
                P[(state ,action)] = (n_state, reward)
        return P

    def check_terminal(self, state):
        return state in self.items.get('fire').get('loc') + self.items.get('water').get('loc')

    def check_move(self, n_state, oldState):
        if n_state not in self.state_space:
            return True
        elif oldState % self.m == 0 and n_state % self.m == self.m - 1:
            return True
        elif oldState % self.m == self.m - 1 and n_state % self.m == 0:
            return True
        else:
            return False

def print_v(v, grid):
    v = np.reshape(v, (grid.n, grid.m))
    cmap = plt.cm.get_cmap('Greens', 10)
    norm = plt.Normalize(v.min(), v.max())
    rgba = cmap(norm(v))
    for w in grid.items.get('water').get('loc'):
        idx = np.unravel_index(w, v.shape)
        rgba[idx] = 0.0, 0.5, 0.8, 1.0
    for f in grid.items.get('fire').get('loc'):
        idx = np.unravel_index(f, v.shape)
        rgba[idx] = 1.0, 0.5, 0.1, 1.0
    fig, ax = plt.subplots()
    im = ax.imshow(rgba, interpolation='nearest')
    for i in range(v.shape[0]):
        for j in range(v.shape[1]):
            if v[i, j] != 0:
                text = ax.text(j, i, v[i, j], ha="center", va="center", color="w")
    plt.axis('off')
    plt.show()

def print_policy(v, policy, grid):
    v = np.reshape(v, (grid.n, grid.m))
    policy = np.reshape(policy, (grid.n, grid.m))
    cmap = plt.cm.get_cmap('Greens', 10)
    norm = plt.Normalize(v.min(), v.max())
    rgba = cmap(norm(v))
    for w in grid.items.get('water').get('loc'):
        idx = np.unravel_index(w, v.shape)
        rgba[idx] = 0.0, 0.5, 0.8, 1.0
    for f in grid.items.get('fire').get('loc'):
        idx = np.unravel_index(f, v.shape)
        rgba[idx] = 1.0, 0.5, 0.1, 1.0
    fig, ax = plt.subplots()
    im = ax.imshow(rgba, interpolation='nearest')
    for i in range(v.shape[0]):
        for j in range(v.shape[1]):
            if v[i, j] != 0:
                text = ax.text(j, i, policy[i, j], ha="center", va="center", color="w")
    plt.axis('off')
    plt.show()

def interate_values(grid, v , policy, gamma, theta):
    converged = False
    i = 0
    while not converged:
        DELTA = 0
        for state in grid.state_space:
            i += 1
            if grid.check_terminal(state):
                v[state] = 0
            else:
                old_v = v[state]
                new_v = []
                for action in grid.actions:
                    (n_state, reward) = grid.P.get((state, action))
                    new_v.append(reward + gamma * v[n_state])
                v[state] = max(new_v)
                DELTA = max(DELTA, np.abs(old_v - v[state]))
        converged = True if DELTA < theta else False

        for state in grid.state_space:
            i += 1
            new_vs = []
            for action in grid.actions:
                (n_state, reward) = grid.P.get((state, action))
                new_vs.append(reward + gamma * v[n_state])
            new_vs = np.array(new_vs)
            best_action_idx = np.where(new_vs == new_vs.max())[0]
            policy[state] = grid.actions[best_action_idx[0]]
    print(i, 'iterations of state space')
    return v, policy

if __name__ == '__main__':
    grid_size = (5, 5)
    items = {'fire': {'reward': -10, 'loc': [12]},
             'water': {'reward': 10, 'loc': [18]}}
    gamma = 1.0
    theta = 1e-10
    v = np.zeros(np.prod(grid_size))
    policy = np.full(np.prod(grid_size), 'n')
    env = GridWorld(grid_size, items)
    v, policy = interate_values(env, v, policy, gamma, theta)
    print_v(v, env)
```

```
print_policy(v, policy, env)
```

Output:



```
4.0  5.0  6.0  7.0  6.0
5.0  6.0  7.0  8.0  7.0
6.0  7.0       9.0  8.0
7.0  8.0  9.0       9.0
6.0  7.0  8.0  9.0  8.0
```



```
D  D  D  D  D
D  D  R  D  D
D  D     D  D
R  R  R     L
U  U  U  U  U
```

- - - - - - - - -

8.Write a program to do reinforcement learning in a grid world.

```
import numpy as np

# global
variablesBOARD_ROWS=3
BOARD_COLS=4
WIN_STATE= (0,3)
LOSE_STATE=(1,3)
START = (2,
0)DETERMINISTIC=Tru
e
classState:
    definit(self,state=START):
        self.board=np.zeros([BOARD_ROWS,BOARD_COLS])s
        elf.board[1,1] =-1
        self.state =
        stateself.isEnd=F
        alse
        self.determine=DETERMINISTIC

    defgiveReward(self):
        ifself.state==WIN_STATE:r
            eturn1
        elif self.state ==
            LOSE_STATE:return-1
        else:
            return0
    defisEndFunc(self):
        if(self.state==WIN_STATE)or(self.state==LOSE_STATE):self.isEn
            d=True

    defnxtPosition(self,action):"
        ""
        action:up, down,left, right

        0 |1|2|3|
        1|
        2|
        returnnextposition"
        ""
        ifself.determine:
            ifaction=="up":
                nxtState=(self.state[0]-
            1,self.state[1])elifaction =="down":
                nxtState = (self.state[0] + 1,
            self.state[1])elifaction =="left":
                nxtState = (self.state[0], self.state[1] -
            1)else:
                nxtState = (self.state[0], self.state[1] +
            1)#if next state legal
            if(nxtState[0] >=0)and(nxtState[0]<=(BOARD_ROWS-
                1)):if(nxtState[1]>=0)and(nxtState[1]<=(BOARD_COLS -1)):
                    ifnxtState!=(1,1):re
                        turn nxtState
            returnself.state

    def
        showBoard(self):self.board
        [self.state]=1
        for i in range(0,
            BOARD_ROWS):print('------')
            out= '|'
            forjinrange(0,BOARD_COLS):ifs
                elf.board[i, j] ==1:
                    token='*'
                ifself.board[i,j]==-
                    1:token='z'
                ifself.board[i,j]==0:to
                    ken ='0'
                out+=token+'|'pri
            nt(out)
        print('-------------------------')

#Agent ofplayer
```

```
classAgent:

    definit(self):self.s
        tates=[]
        self.actions = ["up", "down", "left",
        "right"]self.State=State()
        self.lr=0.2
        self.exp_rate=0.3

        # initial state
        rewardself.state_valu
        es={}
        for i in
            range(BOARD_ROWS):forjinr
            ange(BOARD_COLS):
                self.state_values[(i,j)]=0  #set initial valueto 0

    defchooseAction(self):
        #chooseactionwithmostexpectedvaluemx
        _nxt_reward =0
        action=""

        if np.random.uniform(0, 1) <=
            self.exp_rate:action=np.random.choice(se
            lf.actions)
        else:
            #greedy action
            forain self.actions:
                #iftheactionis deterministic
                nxt_reward =
                self.state_values[self.State.nxtPosition(a)]ifnxt_reward
                >=mx_nxt_reward:
                    action=a
                    mx_nxt_reward =
        nxt_rewardreturnaction

    deftakeAction(self,action):
        position =
        self.State.nxtPosition(action)returnStat
        e(state=position)

    def
        reset(self):self.stat
        es =
        []self.State=State(
        )
    defplay(self,rounds=10):i
        = 0
```

```
whilei <rounds:
               #totheendofgamebackpropagaterewardifsel
               f.State.isEnd:
                   #back propagate
                   reward=self.State.giveReward()
                   # explicitly assign end state to reward
                   valuesself.state_values[self.State.state] = reward# this is
                   optionalprint("GameEnd Reward", reward)
                   forsinreversed(self.states):
                       reward=self.state_values[s]+self.lr *(reward -
                       self.state_values[s])self.state_values[s]=round(reward, 3)
                   self.reset()
                   i +=1
               else:
                   action =
                   self.chooseAction()#append
                   trace
                   self.states.append(self.State.nxtPosition(action))
                   print("currentposition{}action{}".format(self.State.state,action))#b
                   y taking the action, itreaches the nextstate
                   self.State=self.takeAction(action)#
                   mark is endself.State.isEndFunc()
                   print("nxtstate",self.State.state)p
                   rint("------------------------------")
       defshowValues(self):
           for i in range(0,
               BOARD_ROWS):print('
               -------------------------------------------------------')
               out = '|'
               forj inrange(0, BOARD_COLS):
                   out+=str(self.state_values[(i,j)]).ljust(6)+'
               |'print(out)
           print('----------------------------------------------------')
   ifname_____==
       "main":ag=Agent()
       ag.play(50)print(ag.sho
       wValues())
```

Output:

```
|0.951|0.969|0.991|1.0

|0.933|0|0.563|-1.0
```

```
|0.781|0.184|-0.025| -0.2
```

9.Write a program to implement adaptive dynamic programming.

```
import libraries
import os
import random
import gym
import copy
import pickle
import numpy as np
import matplotlib.pyplotaspl
t#Plot values
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
defplot_values(V):

    #reshapevaluefunction
            V_sq=np.reshape(V,(8,8))#pl
            otthestate-valuefunction
            fig=plt.figure(figsize=(10,10))a
            x=fig.add_subplot(111)
            im=ax.imshow(V_sq,cmap='cool')for(j
            ,i),labelinnp.ndenumerate(V_sq):
                ax.text(i, j, np.round(label, 5), ha='center', va='center',
            fontsize=12)plt.tick_params(bottom='off',left='off',labelbottom='off',lab
            elleft='off')plt.title('State-ValueFunction')
            plt.show()
#Performapolicyevaluation
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynbdef
policy_evaluation(env,policy,gamma=1,theta=1e-8):
    V=np.zeros(env.nS)
    whileTrue:
        delta=0
        fors in range(env.nS):
            Vs=0
            fora,action_probinenumerate(policy[s]):
                forprob,next_state,reward,doneinenv.P[s][a]:
                    Vs += action_prob * prob * (reward + gamma *
            V[next_state])delta=max(delta, np.abs(V[s]-Vs))
             V[s]=Vs
         ifdelta<theta:
            break
    returnV
#Performpolicyimprovement
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
defpolicy_improvement(env, V, gamma=1):
    policy=np.zeros([env.nS,env.nA])/env.nAfor
    s in range(env.nS):
        q=q_from_v(env,V,s,gamma)

        #OPTION1:constructadeterministicpolicy#po
        licy[s][np.argmax(q)]=1

        #OPTION2:constructastochasticpolicy
thatputsequalprobabilityonmaximizingactions
        best_a=np.argwhere(q==np.max(q)).flatten()
        policy[s]=np.sum([np.eye(env.nA)[i] foriinbest_a],axis=0)/len(best_a)

    returnpolicy
 #Obtain qfrom V
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
defq_from_v(env, V, s,gamma=1):
    q =
    np.zeros(env.nA)forai
    nrange(env.nA):
        for prob, next_state, reward, done in
            env.P[s][a]:q[a]+=prob* (reward+gamma*
            V[next_state])
    returnq
#Performpolicyiteration
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
defpolicy_iteration(env,gamma=1,theta=1e-8):
    policy=np.ones([env.nS,env.nA])/env.nAwh
    ileTrue:
        V=policy_evaluation(env,policy,gamma,theta)ne
        w_policy=policy_improvement(env, V)

        #OPTION1:stopifthepolicyisunchanged
        afteranimprovementstepif(new_policy ==policy).all():
            break;

        #OPTION2:stopifthevaluefunctionestimatesforsuccessivepolicieshasconverged#ifnp.ma
        x(abs(policy_evaluation(env,policy) -policy_evaluation(env,new_policy)))<
theta*1e2:
        #    break;
        policy=copy.copy(new_policy)r
    eturnpolicy, V
#Truncatedpolicy evaluation
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
deftruncated_policy_evaluation(env,policy,V,max_it=1,gamma=1):nu
    m_it=0
    while num_it <
        max_it:forsinrange(en
        v.nS):
            v =0
            q=q_from_v(env,V, s,gamma)
            fora,action_probinenumerate(policy[s]):v
                +=action_prob * q[a]
            V[s] =
        vnum_it+=1
    returnV
#Truncated policy iteration
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
def truncated_policy_iteration(env, max_it=1, gamma=1, theta=1e-
    8):V=np.zeros(env.nS)
    policy=np.zeros([env.nS,env.nA])/env.nAwh
    ileTrue:
        policy=policy_improvement(env,V)o
        ld_V =copy.copy(V)
        V=truncated_policy_evaluation(env,policy,V,max_it,gamma)ifm
        ax(abs(V-old_V))<theta:
            break;retu
    rnpolicy,V
#Valueiteration
# https://github.com/xadahiya/frozen-lake-dp-
rl/blob/master/Dynamic_Programming_Solution.ipynb
defvalue_iteration(env,gamma=1,theta=1e-8):
    V=np.zeros(env.nS)
    whileTrue:
        delta=0
        for s in
            range(env.nS):v=V[
            s]
            V[s]=max(q_from_v(env,V,s,gamma))del
            ta=max(delta,abs(V[s]-v))
        ifdelta<theta:b
            reak
    policy=policy_improvement(env,V,gamma)re
    turnpolicy, V
# Get an action (0:Left, 1:Down, 2:Right,
3:Up)defget_action(model, state):
    returnnp.random.choice(range(4),p=model[state])#
Saveamodel
defsave_model(bundle:(),type:str):
    withopen('models\\frozen_lake'
        +type+'.adp','wb')asfp:pickle.dump(bundle, fp)
#Load amodel
def load_model(type:str) ->
    ():if(os.path.isfile('models\\frozen_lake'+type+'.adp')==True):
        withopen('models\\frozen_lake'+type+'.adp','rb')asfp:returnpi
            ckle.load(fp)
    else:
        return(None,None)
# The main entry point for this
moduledefmain():
    #Createanenvironment
    env=gym.make('FrozenLake8x8-
    v0',is_slippery=True)#Print information about the
    problem
    print('---FrozenLake ---')
    print('Observationspace:{0}'.format(env.observation_space))p
    rint('Actionspace: {0}'.format(env.action_space))
    print()
    #Printone-
    stepdynamics(probability,next_state,reward,done)print('---One-
    step dynamics')
    print(env.P[1][0])
    print()
    # (1) Random
    policy#model,V=load_mode
    l('1')
    model=np.ones([env.nS,env.nA])/env.nAV
    =policy_evaluation(env, model)
    print('OptimalPolicy(LEFT=0,DOWN =1,RIGHT=2,
    UP=3):')print(model,'\n')
    plot_values(V)save_model(
    (model,V),'1')#(2)Policy
    iteration
    ##model, V =
    load_model('2')#model,V=policy
    _iteration(env)
    #print('OptimalPolicy(LEFT=0, DOWN=1,RIGHT=2,UP
    =3):')#print(model,'\n')
    #plot_values(V)#save_model
    ((model,V),'2')
    #(3)Truncatedpolicyiteration##
    model, V =load_model('3')
    #model, V = truncated_policy_iteration(env,
    max_it=2)#print('OptimalPolicy(LEFT=0, DOWN=1,RIGHT=2,UP
    =3):')#print(model,'\n')
    #plot_values(V)

    #save_model((model, V),
    '3')#(4)Valueiteration##mode
    l,V=load_model('4')
    #model,V=value_iteration(env)
    #print('OptimalPolicy(LEFT=0, DOWN=1,RIGHT=2,UP
    =3):')#print(model,'\n')
    #plot_values(V)#save_mode
    l((model,V),'4')#Variables
    episodes=10
    timesteps=200
    total_score=0#L
    oopepisodes
    forepisodein range(episodes):
        # Start episode and get initial
        observationstate=env.reset()
        #Resetscores
        core=0
        #Loop timesteps
        fort inrange(timesteps):
            #Getanaction(0:Left,1:Down,2:Right,3:Up)acti
            on=get_action(model, state)

            #Performastep
            # Observation (position, reward: 0/1, done: True/False, info:
            Probability)state,reward, done, info =env.step(action)
            # Update
            scorescore+=re
            ward
            total_score+=reward
            #Checkifwearedone(gameover)ifdo
            ne:
                #Render themap
                print('--- Episode {} ---
                '.format(episode+1))env.render(mode='hum
                an')
                print('Score:{0},Timesteps:{1}'.format(score,t+1))pr
                int()
                break

    # Close the
    environmentenv.close()
    #Print thescore
    print('---Evaluation---')
    print ('Score: {0} / {1}'.format(total_score,
    episodes))print()
```

```
#Tell python torunmain method
ifname_____=="main":main()

output:

RandomPolicy
---FrozenLake---
Observationspace:Discrete(64)
Actionspace: Discrete(4)
---One-stepdynamics
[(0.3333333333333333,1,0.0,False),(0.3333333333333333,0,0.0,False),
(0.3333333333333333,9,0.0,False)]
OptimalPolicy(LEFT=0,DOWN=1,RIGHT=
2,UP=3):[[0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.250.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.250.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.250.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]]
---Episode1---
(Down)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:0.0,Timesteps:10
---Episode2---
(Down)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFF
```

```
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
Score:0.0,Timesteps:75
---Episode3---
(Up)SFFFFFFF
FFFFFFFFFFF
HFFFFFFFFFH
FFFFFHFFFFF
HHFFFHFFHF
FHFHFFFFHF
FFG
Score:0.0,Timesteps:28
---Episode4---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:0.0,Timesteps:20
---Episode5---
(Down)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:0.0,Timesteps:8
---Episode6---
(Left)SFFFFFF
FFFFFFFFFFF
FHFFFFFFFFF
HFFFFFHFFFF
FHHFFFHF
```

```
FHFFHFHF
FFFHFFFG
Score:0.0,Timesteps:51
---Episode7---
(Up)SFFFFFFF
FFFFFFFFFFF
HFFFFFFFFFH
FFFFFHFFFFF
HHFFFHFFHF
FHFHFFFFHF
FFG
Score:0.0,Timesteps:19
---Episode8---
(Down)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:0.0,Timesteps:26
---Episode9---
(Left)SFFFFFF
FFFFFFFFFFF
FHFFFFFFFFF
HFFFFFHFFFF
FHHFFFHFFH
FFHFHFFFFH
FFFG
Score:0.0,Timesteps:24
---Episode10---
(Down)SFFFFF
FFFFFFFFFFFF
FHFFFFFFFFF
HFFFFFHFFFF
FHHFFFHFFHF
FHFHFFFFHFF
FG
```

```
Score:0.0,Timesteps:31
---Evaluation---
Score:0.0 /10

Output:

PolicyIteration
---FrozenLake---
Observationspace:Discrete(64)
Actionspace: Discrete(4)
---One-stepdynamics
[(0.3333333333333333,1,0.0,False),(0.3333333333333333,0,0.0,False),
(0.3333333333333333,9,0.0,False)]
OptimalPolicy(LEFT =0, DOWN=1,RIGHT=2,UP =3):[[0.
0.50.50.]
 [0.0.1.0.]
 [0.0.1.0.]
 [0.0.1.0.]
 [0.0.1.0.]
 [0.0.1.0.]
 [0.0.1.0.]
 [0.0.1.0.]
 [0.0.0.1.]
 [0.0.0.1.]
 [0.0. 0. 1.]
 [0.0.0.1.]
 [0.0.0.1.]
 [0.0.0.1.]
 [0.0.0.1.]
 [0.0.1.0.]
 [1.0.0.0.]
 [1.0.0.0.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [0.0.0.1.]
 [0.0.0.1.]
 [0.0.1.0.]
 [1.0.0.0.]
 [1.0.0.0.]
 [1.0.0.0.]
 [0. 0.50.0.5 ]
 [1. 0.0.0.  ]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [0.0.1.0.]
 [1.0.0.0.]
 [0.0.0.1.]
 [0.50. 0.0.5 ]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [0.1.0.0.]
 [0.0.0.1.]
 [0.0.1.0.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0. 0.50.5  0.  ]
 [0.0.0.1.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.0.5  0.5  0.]
 [0.50. 0. 0.5 ]
 [0.250.25 0.25 0.25]
 [0.50.0.50.]
 [0.250.25 0.25 0.25]
 [0.   0.   1.   0.  ]
 [1.   0.   0.   0.  ]
 [0.   1.   0.   0.  ]
 [1.   0.   0.   0.  ]
 [0.2   0.250.250.2  5]
 [0.   0.50.5  0.  ]
 [0.   0. 1.0.  ]
 [0.   1. 0.0.  ]
 [0.250.25 0.25 0.25]]
---Episode1---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:36
---Episode2---
(Right)SFFFFF
FF
```

```
FFFFFFFFF
FFHFFFFF
FFFFHFFF
FFHFFFFF
HHFFFHFF
HFFHFHFF
FFHFFFG
Score:1.0,Timesteps:169
---Episode3---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:113
---Episode4---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:94
---Episode5---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:66
---Episode6---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFF
```

```
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
Score:1.0,Timesteps:111
---Episode7---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:132
---Episode8---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:40
---Episode9---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:111
---Episode10---
(Right)SFFFFFF
FFFFFFFFFFFF
HFFFFFFFFFH
FFFFFHFFFF
```

```
FHHFFFHF
FHFFHFHF
FFFHFFFG
Score:1.0,Timesteps:116
---Evaluation---
Score:10.0/ 10


TruncatedPolicyIteration
---FrozenLake---
Observationspace:Discrete(64)
Actionspace: Discrete(4)
---One-stepdynamics
[(0.3333333333333333,1,0.0,False),(0.3333333333333333,0,0.0,False),
(0.3333333333333333,9,0.0,False)]
OptimalPolicy(LEFT =0, DOWN=1, RIGHT=2,UP= 3):
[[0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   1.   0.  ]
 [1.   0.   0.   0.  ]
 [1.   0.   0.   0.  ]
 [1.   0.   0.   0.]
 [0.2   0.  5 0.25 0 25]
 [0.   0.   1.   0.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   1.   0.  ]
 [1.   0.   0.   0.  ]
 [1.   0.   0.   0.  ]
 [1.   0.   0.   0.  ]
 [0.   0.50.     0.5 ]
 [1.   0.0.      0.]
  [0.250.25 0.25 0.25]
```

```
[0.0.      1.   0.]
[0.0.      1.   0.]
[1.0.      0.   0.]
[0.0.      0.   1.]
[0.50.     0.   0.5 ]
[0.250.25 0.25 0.25]
[0.   0.   1.   0.  ]
[0.   1.   0.   0.  ]
[0.   0.   0.   1.  ]
[0.   0.   1.   0.  ]
[1.   0.   0.   0.  ]
[0.250.25 0.25 0.25]
[0.250.25 0.25 0.25]
[0.   0.50.5  0.  ]
[0.   0. 0.1.  ]
[1.   0. 0.0.  ]
[0.250.25 0.25 0.25]
[0. 0.1.0.  ]
```

```
  (Right)SF
FFFFFFFFF
FFFFFFFF
HFFFFFFF
FFHFFFFF
HFFFFFHH
FFFHFFHF
FHFHFFFF
HFFFG
Score:1.0,Timesteps:97
---Episode3---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:127
---Episode4---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:113
---Episode5---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:44
---Episode6---
(Right)SFFFFF
FF
```

```
FFFFFFFFF
FFHFFFFF
FFFFHFFF
FFHFFFFF
HHFFFHFF
HFFHFHFF
FFHFFFG
Score:1.0,Timesteps:166
---Episode7---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:42
---Episode8---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:170
---Episode9---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:75
---Episode10---
(Right)SFFFFFF
FFFFFFFFFFFF
HFFFF
```

```
FFFFFHFF
FFFHFFFF
FHHFFFHF
FHFFHFHF
FFFHFFFG
Score:1.0,Timesteps:57
---Evaluation---
Score:10.0/ 10


ValueIteration
---FrozenLake---
Observationspace:Discrete(64)
Actionspace: Discrete(4)
---One-stepdynamics
[(0.3333333333333333,1,0.0,False),(0.3333333333333333,0,0.0,False),
(0.3333333333333333,9,0.0,False)]
OptimalPolicy(LEFT =0, DOWN=1, RIGHT=2,UP= 3):
[[0.   1.   0.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   1.   0.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   0.   1.  ]
 [0.   0.   1.   0.  ]
 [1.   0.   0.   0.  ]
 [1.   0.   0.   0.  ]
 [1.   0.   0.   0.]
 [0.2   0.  5 ).25 ( 25]
 [0.0.1.0.]
 [0.0.0.1.]
 [0.0.0.1.]
 [0.0.1.0.]
 [1.0.0.0.]
 [1.0.0.0.]
 [1.0.0.0.]
 [0. 0.50.0.5 ]
 [1. 0.0.0.  ]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [0.0.1.0.]
 [1.0.0.0.]
 [0.0.0.1.]
 [0.50. 0.0.5 ]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [0.1.0.0.]
 [0.0.0.1.]
 [0.0.1.0.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.250.25 0.25 0.25]
 [0. 0.50.5  0.  ]
 [0.0. 0. 1.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.0.1.0.]
 [1.0.0.0.]
 [0.250.25 0.25 0.25]
 [0.0.5  0.5  0.]
 [0.50. 0. 0.5 ]
 [0.250.25 0.25 0.25]
 [0.50.0.50.]
 [0.250.25 0.25 0.25]
 [0.   0.   1.   0.  ]
 [1.   0.   0.   0.  ]
 [0.   1.   0.   0.  ]
 [1.   0.   0.   0.  ]
 [0.2   0.250.250.2  5]
 [0.   0.50.5  0.  ]
 [0.   0. 1.0.  ]
 [0.   1. 0.0.  ]
 [0.250.25 0.25 0.25]]
---Episode1---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
```

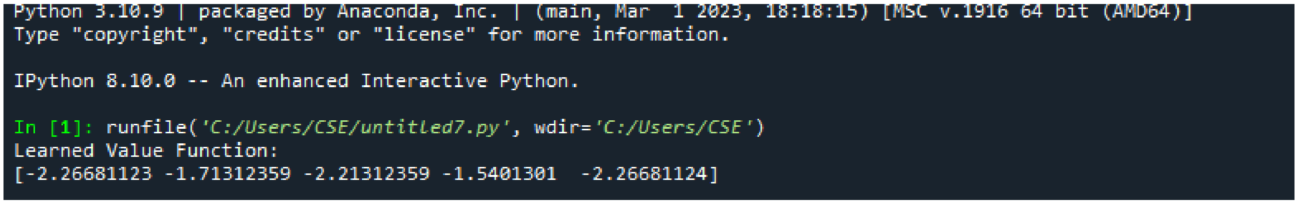
```
Score:1.0,Timesteps:96
---Episode2---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:116
---Episode3---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:188
---Episode4---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:124
---Episode5---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:0.0,Timesteps:200
---Episode 6---
```

```
  (Right)SF
FFFFFFFFF
FFFFFFFF
HFFFFFFF
FFHFFFFF
HFFFFFHH
FFFHFFHF
FHFHFFFF
HFFFG
Score:1.0,Timesteps:71
---Episode7---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:90
---Episode8---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:102
---Episode9---
(Right)SFFFFF
FFFFFFFFFFF
FFHFFFFFFFF
FHFFFFFHFFF
FFHHFFFHFF
HFFHFHFFFF
HFFFG
Score:1.0,Timesteps:52
---Episode10---
(Right)SFFFFFF
```

10 .Write a program to implement active dynamic programming.

```
import numpy as np

# Define the environment
num_states = 5
num_actions = 2
gamma = 0.9  # Discount factor

# Initialize value function
V = np.zeros(num_states)

# Define the reward matrix
rewards = np.array([[0, -1],
                    [-1, 1],
                    [0, -1],
                    [0, 1],
                    [-1, 0]])

# Define the transition matrix
transitions = np.array([[1, 2],
                        [0, 3],
                        [3, 4],
                        [4, 0],
                        [2, 1]])

# Active dynamic programming algorithm (Policy Evaluation)
num_iterations = 100

for iteration in range(num_iterations):
    for state in range(num_states):
        value_sum = 0
        for action in range(num_actions):
            next_state = transitions[state, action]
            reward = rewards[state, action]
            value_sum += (1 / num_actions) * (reward + gamma * V[next_state])
        V[state] = value_sum

# Print the learned value function
print("Learned Value Function:")
print(V)
```

output:



```
Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.10.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/CSE/untitled7.py', wdir='C:/Users/CSE')
Learned Value Function:
[-2.26681123 -1.71312359 -2.21312359 -1.5401301  -2.26681124]
```

11. Write a program to implement Q learning.

Scenario–RobotsinaWarehouse
Agrowinge-commercecompany isbuildinganew warehouse,andthecompanywouldlikeallofthepicking operationsin the newwarehousetobeperformed by warehouserobots.

In the context of e-commerce warehousing, “picking” is the task of gathering individual itemsfromvarious locations inthe warehousein order tofulfill customer orders.
After picking items from the shelves, the robots must bring the items to a specific location withinthewarehousewheretheitems can bepackagedfor shipping.

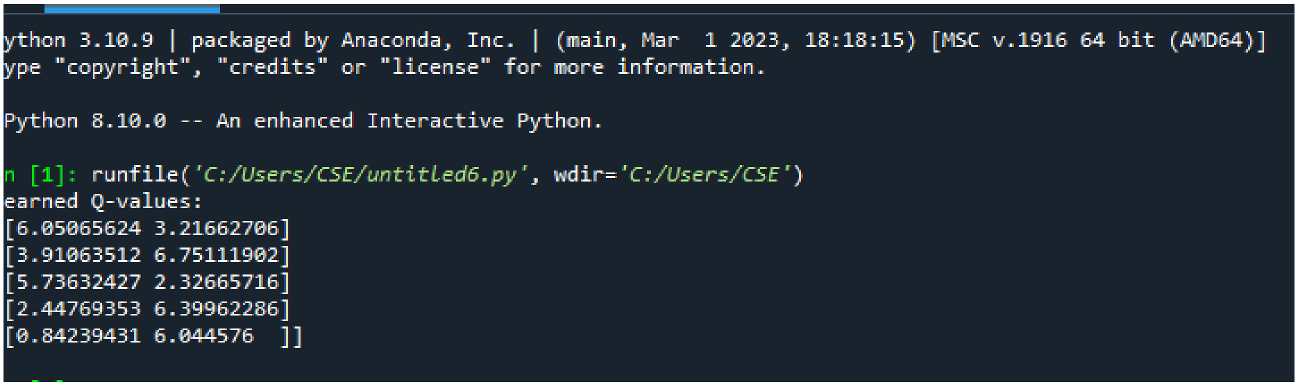
Inordertoensuremaximum efficiencyandproductivity,therobotswill needtolearntheshortestpath between the item packaging area and all other locations within the warehouse where therobotsareallowed to travel.

WewilluseQ-learningtoaccomplish thistask!

```
import numpy as np

# Define the environment
num_states = 5
num_actions = 2
gamma = 0.9  # Discount factor

# Initialize Q-values
Q = np.zeros((num_states, num_actions))

# Define the reward matrix
rewards = np.array([[0, -1],
                    [-1, 1],
                    [0, -1],
                    [0, 1],
                    [-1, 0]])

# Define the transition matrix
transitions = np.array([[1, 2],
                        [0, 3],
                        [3, 4],
                        [4, 0],
                        [2, 1]])

# Q-learning parameters
learning_rate = 0.1
num_episodes = 1000

# Q-learning algorithm
for episode in range(num_episodes):
    state = np.random.randint(0, num_states)  # Start in a random state

    while True:
        action = np.argmax(Q[state, :]) if np.random.rand() < 0.9 else np.random.randint(0,
num_actions)

        next_state = transitions[state, action]
        reward = rewards[state, action]

        Q[state, action] += learning_rate * (reward + gamma * np.max(Q[next_state, :]) - Q[state,
action])

        state = next_state

        if state == 3:  # Reached the goal state
            break

# Print the learned Q-values
print("Learned Q-values:")
print(Q)
```

output:

```
  Learned Q-values:
  [[6.05065624 3.21662706]
   [3.91063512 6.75111902]
   [5.73632427 2.32665716]
   [2.44769353 6.39962286]
[0.84239431 6.044576  ]]
```



```
ython 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)]
ype "copyright", "credits" or "license" for more information.

Python 8.10.0 -- An enhanced Interactive Python.

n [1]: runfile('C:/Users/CSE/untitled6.py', wdir='C:/Users/CSE')
earned Q-values:
[6.05065624 3.21662706]
[3.91063512 6.75111902]
[5.73632427 2.32665716]
[2.44769353 6.39962286]
[0.84239431 6.044576  ]]
```

12. Artificialintelligencetest:a casestudyofintelligentvehicles.

12.Artificialintelligence test:acasestudyofintelligentvehicles

Abstract:

To meet the urgent requirement of reliable artificial intelligence applications, we discuss the tightlink between artificial intelligence and intelligence test in this paper. We highlight the role oftasks in intelligence test for all kinds of artificial intelligence. We explain the necessity anddifficulty of describing tasks for intelligence test, checking all the tasks that may encounter inintelligence test, designing simulation-based test, and setting appropriate test performanceevaluation indices. As an example, we present how to design reliable intelligence test forintelligentvehicles. Finally,wediscussthefuture research directionsof intelligencetest.

Introduction:

Artificial intelligence (AI) usually refers to intelligence exhibited by machines. Nowadays, AIhas transformed our lives in many aspects, from semi-autonomous cars on the roads to roboticvacuumsinour homes.With nodoubts, AIwill continueto invadeeveryareaofour lives,fromhealthcareto education, entertainmentto security, in the next20years.

To answer such questions, we need to rethink what artificial intelligence is. Clearly, thedefinitiongivenat thebeginning ofthis paperis notprecise. Amorerigorousdefinition canbegiven as “Artificial intelligence is the intelligence (that is similar to or the same kind as humanintelligence)exhibited bymachines (in thesame task)”.

We can see that this new definition reveals the tight link between artificial intelligence andintelligence test. If and only if a machine finishes a set of specially designed tasks, we can saythat this machine exhibits intelligence as human. This new definition is similar to Minsky’sdefinition: AI is “the science of making machines capable of performing tasks that would requireintelligenceifdoneby[humans]”(Minsky1968).

Thedifferenceisthatourdefinitionfocuses onthe result (performing tasks); while Minsky’s definition highlights the cause (the requiredintelligence). This definition belongs to the so-called behavior type AI definition proposed in(Russelland Norvig2010).

Moreover, the choice of the designed tasks characterizes the kind of intelligence that thismachine can have. Two sets of tasks may have no or few overlaps so that we cannot simplydeterminewhichoneis moredifficult.For example,anilliterate humanmaybeadriverandawell-educatedblinded human may not be able to drive.

Turing is the first researcher who realized the importance of intelligence test for developingartificial intelligence (Turing 1950). He proposed a test in which a human evaluator would judgenatural language conversations between a human and a machine designed to generate human-likeresponses. If the evaluator cannot reliably distinguish the machine from the human, the machineissaid to havefinished thetask and passed the test.

However, Turing test has several shortcomings and cannot be directly applied in many otherapplications which require reliable intelligence test for machines (Levesque 2014, 2017;Ackerman 2014; Schoenick et al. 2017). One example is intelligent vehicles that draw greatattention from researchers, automobile manufacturers and the public in the last 10 years (Li andWang 2007; Eskandarian 2012). In order to solve this problem, some initial attempts had beencarried out recently (Broggi et al. 2013, 2015; Huang et al. 2014; Wagner and Koopman 2015; Liet al. 2017; Koopman and Wagner 2017; Watzenig and Horn 2017a, b; Zhao et al. 2017), butnone of them give a clear portrait of the difficulties of intelligence test and explain the origins ofthesedifficulties.

Facing such a predicament, some researchers claimed that machine-learning based autonomy isbrittle and lacks ‘legibility’. In contrast, more researchers believed that the field of autonomy isundergoingamachinelearning revolution. Theythought thattheright timehas comeandwe shouldcombineadvancesinintelligentmachinelearningwithintelligentmachinetestingofempiri calautonomy applications.

Noticing that testing of intelligence is attracting more interests in recent studies, we survey thestate-of-the-art achievements in this field in this paper. We account for the difficulties ofintelligence test, highlight the role of tasks in intelligence test for all kinds of artificialintelligence, and discuss how to design reliable intelligence test for intelligent vehicles. We willnot discuss the so-called strong (or hard) artificial intelligence which requires an intelligentmachine to have an artificial general (full) intelligence and exhibit behavior as flexible ashumans do (Ohlsson et al. 2017). Instead, we will focus on intelligence test for weak (or soft)artificialintelligencewhichrequiresanintelligentmachinetosolvespecific problemsashumanswould do (Newell and Simon 1976; Kurzweil 2005). Furthermore, the recent progress inintelligent vehicles indicates that appropriate testing methods could help significantly improvetheefficiencyofintelligencetestandthusincrease thereliability ofsomeintelligentmachines. Allthepromising achievementsurgeusto putmoreeffortsintothis researchfield.Thevalidation oftasks:

The above assumption naturally leads to the second difficulty of intelligence test: *How toguarantee that the machine acts accordingly for all the tasks that may encounter in a scenario*?In general, we could view task validation as a decision problem that has been studied incomputability (complexity) theory (Bradley and Manna 2007; Ding et al. 2013; Kroening andStrichman 2016). The input of the machine is the setting of tasks. If the machine passes a task,we assume it outputs “yes”; otherwise it outputs “no”. We hope that the machine outputs “yes”forallpossible inputs.

The complexity of decision problem varies significantly. Though few theoretical analysis hadbeen made for intelligence test, we can easily find that some tasks are at least as hard as thenondeterministic polynomial time (NP) decision problems (Karp 1972). Till now, we still do nothave the ranking standard to evaluate the complexity level of special kinds of artificialintelligence. We believe more and more research interests will be attracted to such a field in thenearfuture.

For some relatively simple intelligence tests, if the scenario can be described in terms of discretevariables, we enumerate all the tasks that may occur and validate the performance of machine ineach possible task. This is often troublesome and time-consuming, due to the famouscombinatorial explosion problem. For example, a brute force validation reported in (Lamb 2016)had generated a 200-terabyte proof. If the scenario is described in terms of continuous variables,things may become worse, since we cannot enumerate all the combinations of variables due totheircontinuity.

One widely-used strategy to handle such problems is to sample the countless combinations ofvariables and just check the performance of the machine within these limited sampled tasks. Ifthese representative test samples are appropriately selected, the machine which has finished allthe sampled tasks is expected to behave well for all the remaining tasks, since the capability ofthe machine is built to be generalizable. For example, AlphaGo does not enumerate all thebranchesofGogame,if weviewallthedecisionspaceof Gogame asadecisiontree.Instead,itsbuild-in policy-network helps to filter many branches of the Go game tree and just sample a fewnodesofthis treeto trainthe machine(Silver etal. 2016,2017b; HeuleandKullmann 2017). Competition between AlphaGo and human masters show that the policy-network based samplingstrategygenerallyworkswell.However,AlphaGostilllostonegametoLeeSedol,due to incompletetrainingsamples in2016.ThedesignersofAlphaGo usedmore samples toteachthemachineto fix this problem and wonall the official 60 games in 2017.

The sampling process can be guided by deterministic rules, or randomly data-driven, or evenmixed. For example, researchers had proved that solving the Sudoku minimum number of cluesproblem is 16 via hitting set enumeration (Mcguire et al. 2014). Differently, at least partiallyrandomly, data-driven adversarial decision-exploration and self-playing help build AlphaGofroma zero-knowledgebeginnerof Go gametoasuper Go master.

It should be pointed out that gaming is found to be a very effective task exploration tool whichprovides a good way to find the new samples for continuous learning and testing. Interestingly,Turing may be the first one to realize the power of gaming in artificial intelligenceimplementation and testing (Turing 1950). The emerging Generative Adversarial Nets (GAN)(Goodfellow et al. 2014) and the recently proposed parallel learning framework (Li et al. 2017)canall beviewed asapplicationsof gaming based(adversarial) learning.

For some artificial intelligence applications, we will require the machine to pass all therepresentative tasks that will cover the whole task space. For example, we aim to test everypossible extreme task an intelligent vehicle may encounter in practice (Zheng et al. 2004; Li etal. 2012,2017; Huang et al. 2014; Wagner and Koopman 2015; Watzenig and Horn 2017a, b;Zhao et al. 2017), so as to avoid any severe accidents (A Tragic Loss 2016). However, no onecan guarantee that AlphaGo will not lose a game anymore (Wang 2016a,b). How many sampletasksthat areneededremains to befathomed.

Thedesignofsimulation-basedtest

The desire to sample enough tasks forces us to resort to simulation-based intelligence test, sincethe time and financial costs of practical intelligence tests are often too high to afford. This leadsto the third difficulty of intelligence test: *How to make the simulation-based test as “real” aspossible*?

We could roughly categorize the simulating objects into three kinds: natural objects, man-madeobjects and human ourselves. Man-made objects are relatively easy to simulate because weusually know the exact math or physical disciplines that govern the behaviors of these objects.Some natural objects are difficult to simulate since they are much more complex to model. Weusually introduce certain simplification and just reproduce the major features of these objects.For example, we assume that the arriving rate of vehicles follows certain distributions to test theperformanceof intelligenttraffic controlsystems (Tonget al.2015; Liet al.2016a,b).

To mimic human behaviors is difficult. Actually, we meet a causal loop here: to test whether amachine behaves like a human, we need to set up simulation-based test; and to better simulatehuman that may interact with the machine, we need to well describe and simulate behaviors ofhuman. This again requires us to judge whether the machine behaves like a human. The onlypossible solution to this dilemma is to build a spiral escalation process: the simulation willincreaseourknowledgeabouthowtodescribeandsimulatebehaviorsofhuman,andmeanwhile,the gained knowledge helps better simulate human behaviors (Wang et al. 2016a; Li et al. 2017).Thesetting of performanceindices

Inmany applications,wehavedifferentgoalsofusingintelligentmachines.Thisleadstothefourth difficulty of intelligence test: *How to establish the appropriate test performanceevaluationindices for tasks*?

The first kind of performance indices is to require the machine to behave like a human. A simpleyet effective is to first observe how human operate in a certain task and then set up a criterion tomeasurehow closeartificialintelligentmachineoperationsdifferfromhumanoperations(Argall etal.2009;Bagnell 2015;Kuefler etal.2017).Therefore,theproblemistransferredintofindingan appropriate criterion that is able to robustly and smartly distinguish between intelligentmachine operations and human operations, based on limited samples. Many researchers againresortedtotheemerging GenerativeAdversarialNets(GAN)(Ho and Ermon2017;Merelet al. 2017), since we do not need to provide explicit rules to measure the difference. The implicit(dis)similarity between man-made and machine-made data will be automatically extracted andcomparedwhenGANiscorrectlyused.However, wehavetoadmitthat,forsomeapplications,westill do not know how to set anappropriatequantitative criteria.

The second kind of performance indices is to reach the best performance. For example, in allchessgames, weaim tobuild themachinethat canbeat alltheotheropponentsratherthanmakeit play like a human player. It is relatively easy to set the corresponding performance indices forsuchsingle-objectiveapplications.

Unlike chess games in which players only aim to win, many intelligent applications have multi-objectives. For example, intelligent vehicles consider driving safety, travel speed, fuelconsumption, and some other issues. Because different performance indices may lead to quitedifferent implementations of intelligent machines, we should be very careful to set appropriateperformanceindices to balance different objectives.

In 2016–2017 Intelligent Vehicle Future Challenge hold in Changshu city of China, the timeusedbyaparticipatingvehicletopassthegiven10taskswastakenasone ofthestandardsofgrading for intelligence level, since it is a nice synthetic criterion. Any traffic violation (e.g.running through a red light) will lead to a deduction of the final score. It is interesting thatchallenge participators have noticeably different preferences of the deduction values for eachtask.Thejudges had to holda 3-h meeting to finallysettle down the scoring rules.

Moreover,whenthepersonalfeelingisconsidered,itbecomesevenhardertosettheappropriateperformance indices. For example, personal preferences of driving may vary significantly fromperson to person (Classen et al. 2011; Butakov and Ioannou 2015; Lefèvre et al. 2015). To thebest of our knowledge, few studies had established an accurate, flexible, and adjustable standardofgrading fordifferent personalizing aspects of driving.

Intelligencetestforintelligentvehicles

Since it is impossible to summarize all the AI applications, we take intelligent vehicles as anexampletopresent aframeworkofintelligencetestandreview thelatestadvanceinthisfield.Thedefinition and generation ofintelligencetest tasks forvehicles

Most previous tests of intelligent vehicles did not provide a clear definition of drivingintelligence. We can roughly categorize them into two kinds: scenario-based tests andfunctionality-basedtests.

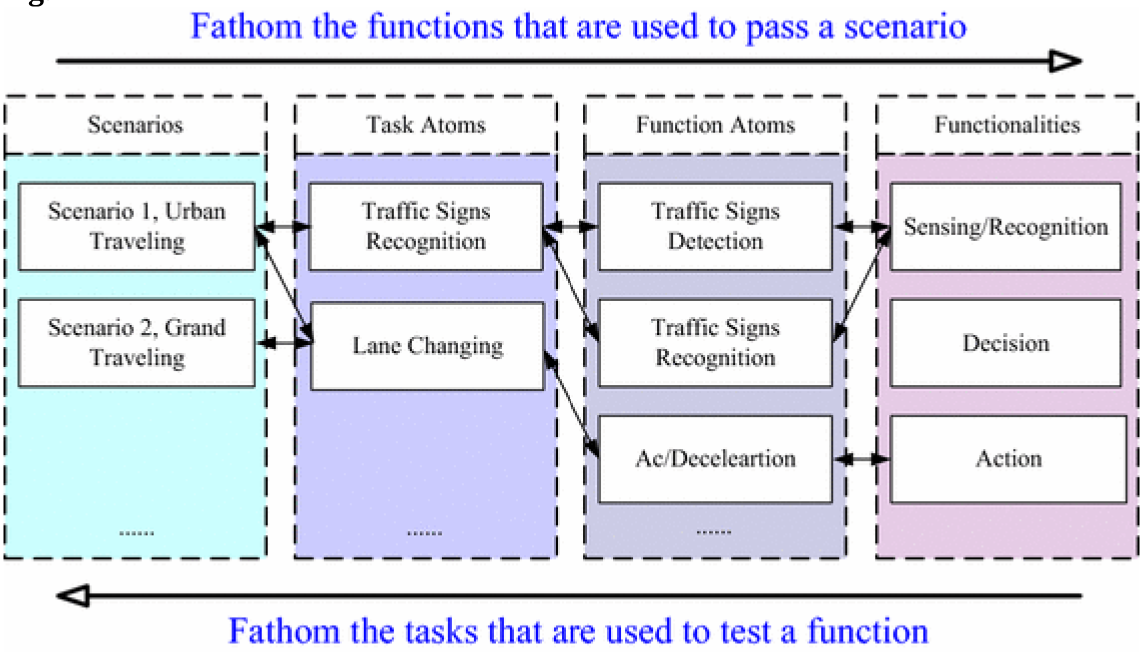
Scenario-based tests, such as DARPA Grand Challenge and DARPA Urban Challenge, justrequire an autonomous vehicle to pass a special region safely within a limited time (DARPAGrandChallengeandDARPAUrbanChallenge2004–2007;Buehleretal.2009;Campbelletal. 2010). The number and the kind of traffic participants are not clearly defined. The scene andthedrivingenvironmentisnotexplicitlygiven,either.Thisismainlybecauseresearcherscannotenumerate all the possiblesettings of driving situations.

Functionality-based (ability-based) tests examine three components of driving intelligence:sensing/recognition functionality, decision functionality according to the recognized information,and action functionality with respect to the decision (Li et al. 2012, 2016a, b; Huang et al. 2014;Hernández-Orallo 2017). Special detailed functions (e.g., traffic sign recognition) will be furthertestedwithspeciallydesignedtasks (GTSDB 2014).However,existingfunctionality-based tests

are carried out separately and independently, which makes it impossible to get a comprehensiveunderstandingoftheintelligencelevel ofvehiclesandthusdegrades thereliabilityofsuchtests.

Recently,asemantic relationdiagramfordrivingintelligencewasproposedin(Li et al. 2016a,b) to better define the intelligence of vehicles. Task atoms are on one side of thissemantic relation diagram, while function atoms are on the other side of this semantic relationdiagram.Thelinksbetweenthesetwosidesdenotethatitusuallyrequiresanautonomousvehiclet o perform several function atoms to fulfill any task atom. Moreover, various combinations oftask atoms can be grouped into different kinds of driving scenarios. Meanwhile, analogous tohuman drivers, the function atoms can also be grouped into three major categories:sensing/recognitionfunctionality,decisionfunctionality,andactionfunctionality;seeFig.1 .

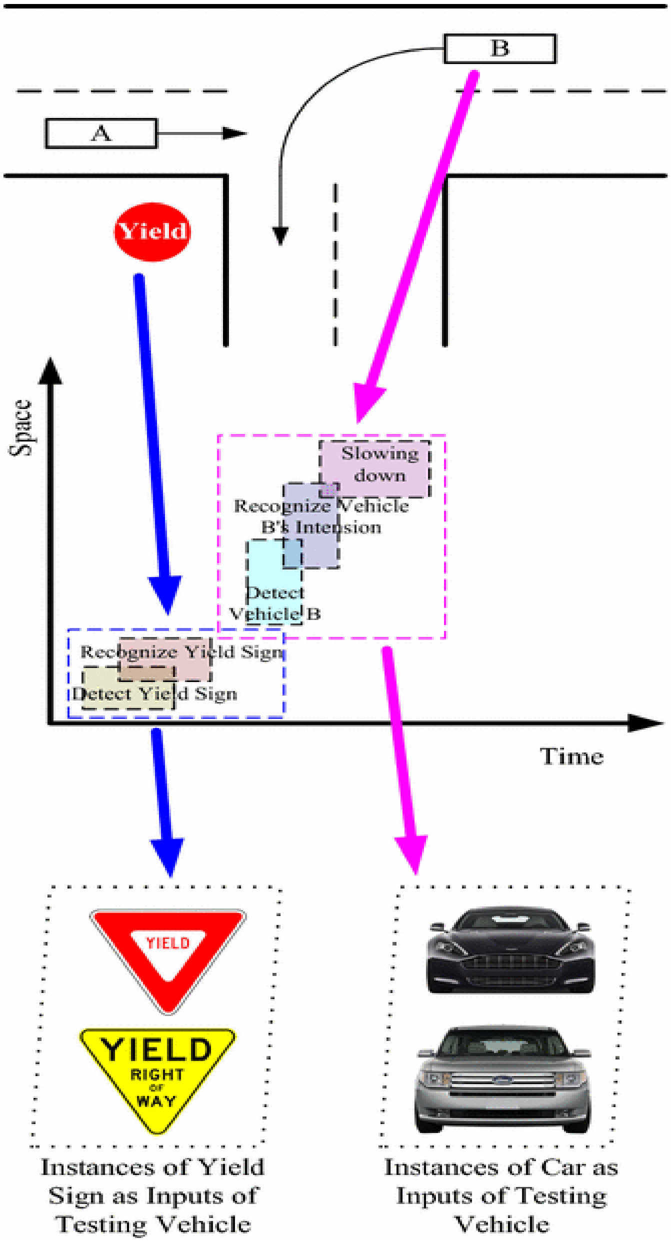
Fig. 1


Fathom the functions that are used to pass a scenario
Scenarios
Task Atoms
Function Atoms
Functionalities
Scenario 1, Urban Traveling
Scenario 2, Grand Traveling
Traffic Signs Recognition
Lane Changing
Traffic Signs Detection
Traffic Signs Recognition
Ac/Deceleartion
Sensing/Recognition
Decision
Action
......
......
......
Fathom the tasks that are used to test a function

An illustration of the semantic relation diagram for driving intelligence of autonomous vehiclesWe can see that scenario-based tests only emphasize the left part of this semantic relationdiagram; while functionality-based (ability-based) tests only emphasize the right part of it. So,this semantic relation diagram actually integrates the two major kinds of intelligent vehicletesting approaches. Moreover, if we transverse from the right side of the semantic relationdiagram to the left side of the semantic diagram, we will generate the desired test task that isneededforsomespecialfunctions(abilities).So, thissemanticrelationdiagramnotonlydefinestheintelligencerequiredto drive avehiclebutalso gives theway oftest task generation.

Based on this semantic relation diagram definition, a detailed test design can be simplified as aspecial temporal and spatial arrangement of task atoms. As shown in Fig. 2, each task can betaken as a rectangle. The left vertical boundary of this rectangle denotes the time that a taskstarts,and;theright verticalboundarydefines the maximalallowable timewhenataskmustbefinished. The left horizontal boundary of this rectangle denotes the position that a task starts,and; the right horizontal boundary defines the maximally allowable position where a task mustbe finished. Since a vehicle may need to process and finish several task atoms simultaneously,thetemporal-spatial rangeof atask may beoverlapped withthoseof othertasks.

Fig. 2

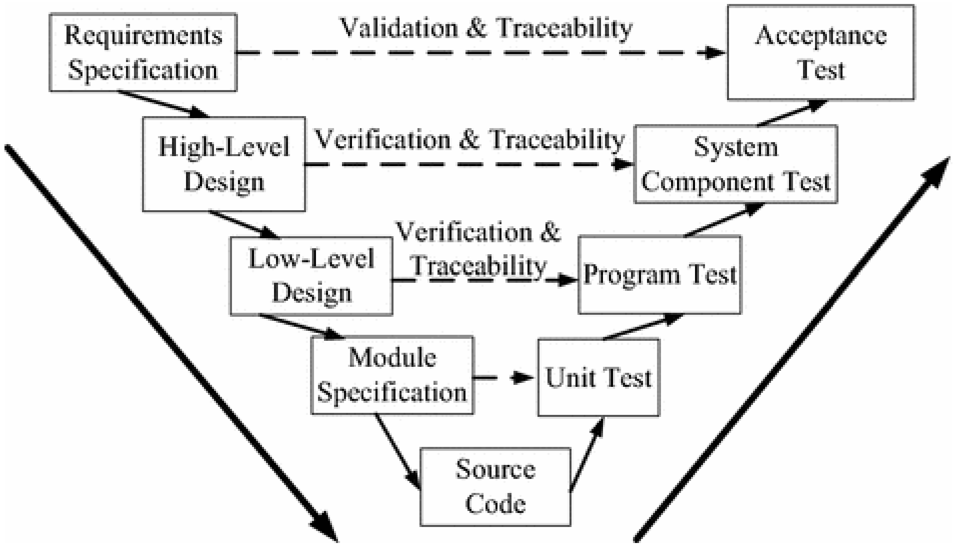

B
A
Yield
Space
Slowing down
Recognize Vehicle B's Intension
Detect Vehicle B
Recognize Yield Sign
Detect Yield Sign
Time
YIELD
YIELD RIGHT OF WAY
Instances of Yield Sign as Inputs of Testing Vehicle
Instances of Car as Inputs of Testing Vehicle

An illustration of transforming a typical driving scenario into the corresponding temporal-spatialplot of the assigned tasks and generating sample instances of the related objects in simulation,accordingto theassignedtemporal-spatial positions of tasks

The number of task atoms, the difficulties of task atoms, and the numbers of concurrent taskatoms all influence the difficulty of a particular task. Varying all these factors, we can sampleandtest tasks with different difficulty levels;seeFig.2.

Itisinterestingtocomparetheabovetaskdefinitionandgenerationprocesswiththeso-calledV-model which is frequently used for conventional automobile software development. V-modelmeans Verification and Validation model. As shown in the right part of Fig. 3, it assumes thattestingof the systemis planned inparallel with acorresponding phaseofdevelopment.

Fig. 3



Anillustration ofthe V-model

The first phase of the V-model is the requirement phase which creates a system testing planbeforedevelopmentstarts.Thecorrespondingtestplanfocusesonmeetingthefunctionalityspecifiedin therequirements gathering.

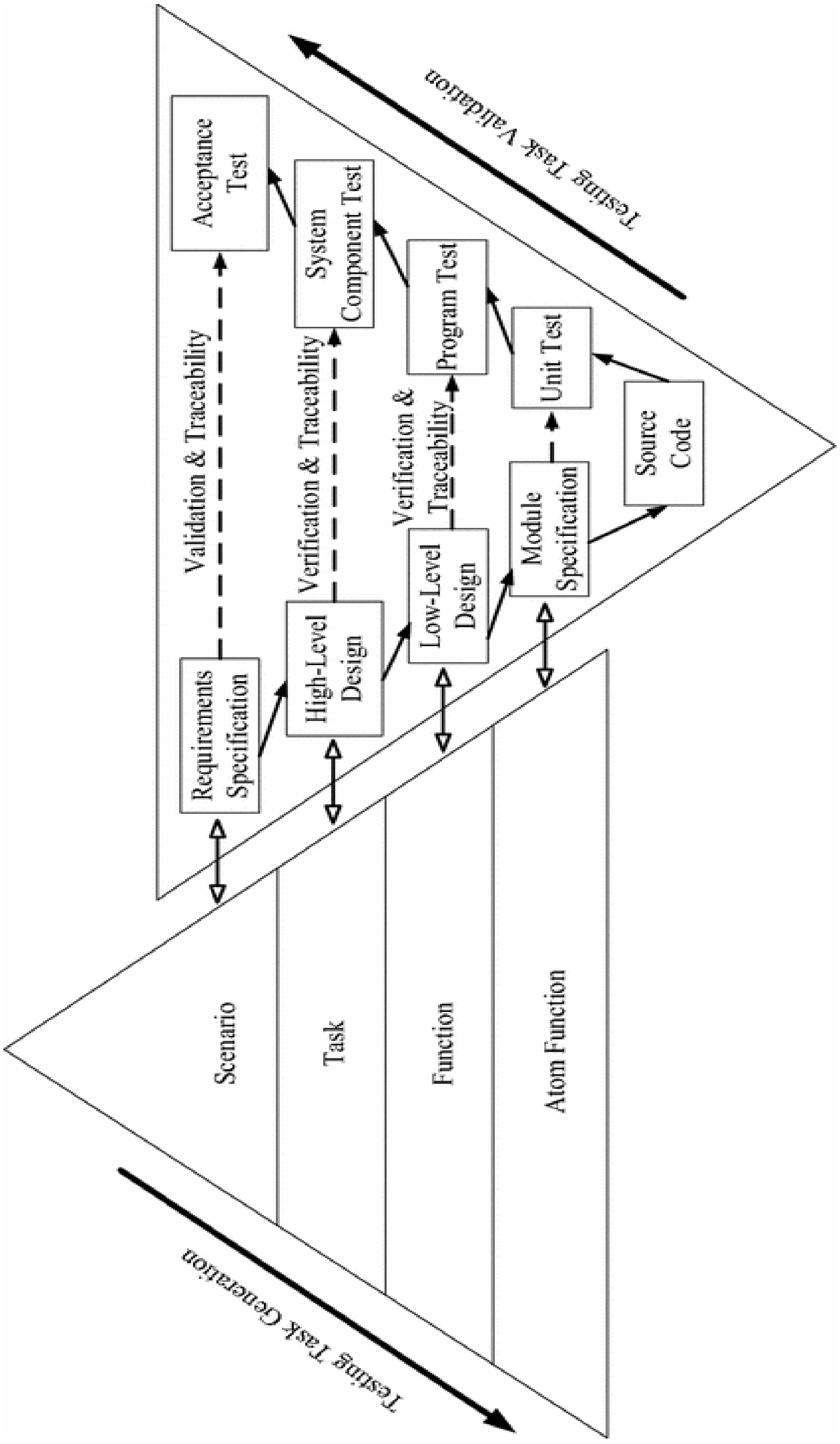
The second phase of the V-model is the high-level design phase which characterizes systemarchitecture and design, providing an overview of the solution. Correspondingly, an integrationtestplanis createdinthisphaseaswell inordertotestthepieces ofthesoftwaresystemsabilitytowork together.

The third phase of the V-model is the low-level design phase which designs the actual softwarecomponents, defines the operation rules for each component of the system, and sets therelationshipbetweeneachdesignedclasses.Correspondingly,componenttestsarecreatedinthisphase.

The fourth phase of the V-model is the module design phase which further decomposes thecomponents into a number of software modules that can be freely combined. The bottom phaseof the V-model is the coding phase where all design is converted into the code by developers.Thedependencesofdifferentmodulesareminimized.Correspondingly,unittestingisperformedbythe developers on theobtained codeto checktheperformanceof modules.

Ifwecombinethe aforementionedtesttasksgeneration methodwiththeV-model,wecangetaΛΛ–V-model asshown in Fig. 4. Sincethe definition oftheup-level“scenario”is usually much more abstract than the definition of the low-level “task” and “function”, we use the Greeksymbol ΛΛ to represent this top-down design. The phase-by-phase specification in the V-modelis right a transverse from the left side of the semantic relation diagram to the right side of thesemanticdiagram.

Fig. 4



AnillustrationoftheΛΛ–V-model

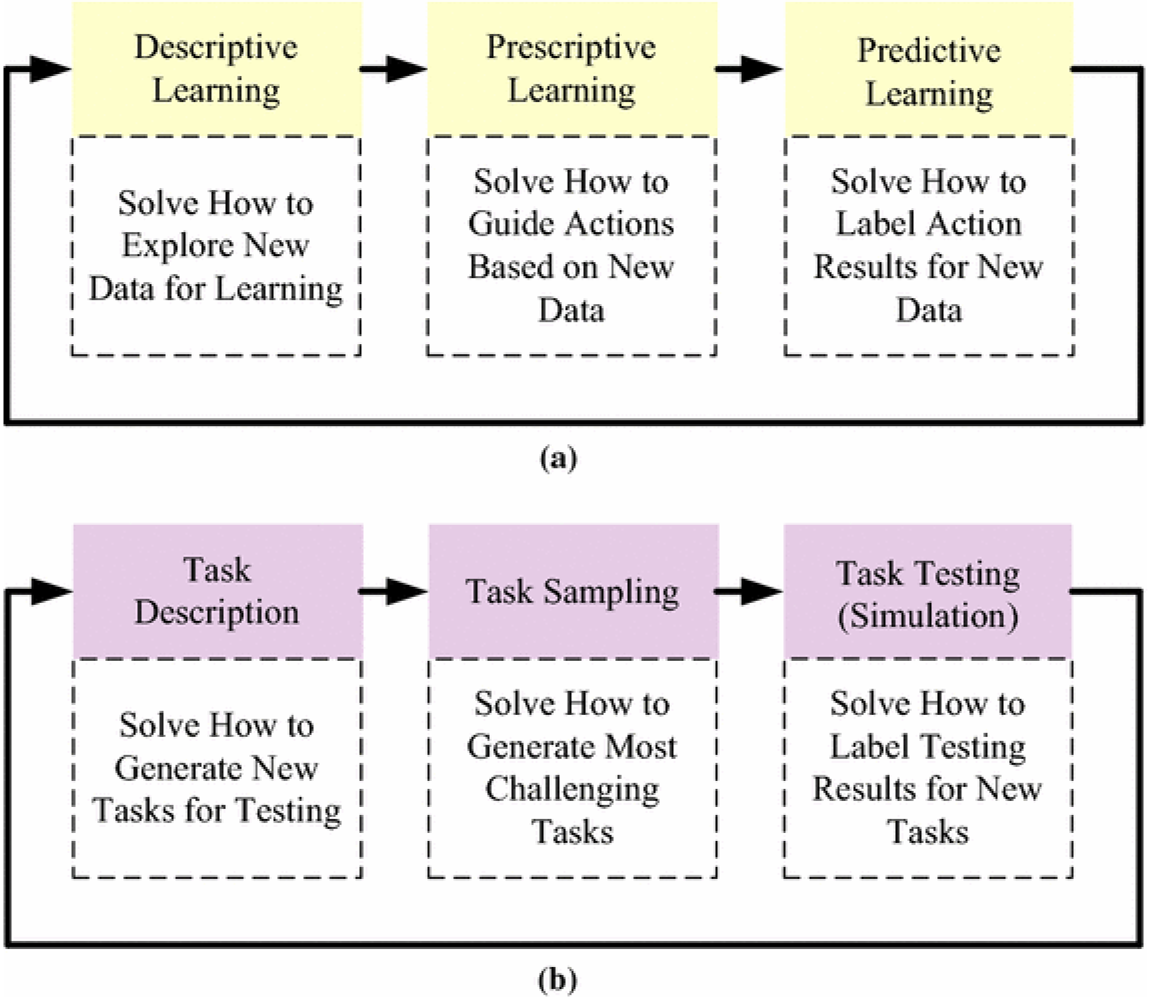
Theframeworkof intelligencetestingsystemforvehicles

Whentest tasksaredetermined,wewillbuild thetesting system.

V-model is simple and easy to use for small system development where requirements can bestraightforwardly understood. However, test designing happens before coding in the V-model.This makes V-model very rigid and inflexible for complex artificial intelligent systemdevelopment.

As pointed out in (Boehm 1988; Raccoon 1997; Black 2009), we should take a spiral loop to findmost challenging test tasks. Because learning and testing are two sides of the same coin, thearchitecture of such a powerful testing system should share a similar loop structure with somecertainpowerful artificialintelligencelearning systems.

Let us take the recently proposed parallel learning framework (Li et al. 2017) as an example. Asshown in Fig. 5a, parallel learning first applies descriptive learning to create the same (kind of)new data. This is just as Prof. Richard Feynman had said: “What I cannot create, I do notunderstand.” Then, parallel learning applies prescriptive learning to make system evolveappropriately by special trying-and-testing and guide system with growing knowledge. Finally,parallel learning applies predictive learning to label data-action pair and leads the system toevolve in an unsupervised manner. The new action will generate new data and forms a loop intheend.Thesystemwillfinally mastertheknowledgeof choosingthe appropriateactionsfor allthe tested data. Such knowledge will be generalized to choose the actions for the untested data.**Fig. 5**


Descriptive Learning
Prescriptive Learning
Predictive Learning
Solve How to Explore New Data for Learning
Solve How to Guide Actions Based on New Data
Solve How to Label Action Results for New Data
(a)
Task Description
Task Sampling
Task Testing (Simulation)
Solve How to Generate New Tasks for Testing
Solve How to Generate Most Challenging Tasks
Solve How to Label Testing Results for New Tasks
(b)

A comparison of **a** parallel learning loop (Li et al. 2017); and **b** testing loop for artificialintelligence

Check the inner mechanism of AlphaGo, we can find that it indeed does the same thing. Therules of Go game is first encoded (descriptive learning). The system sets up a deep neuralnetworkbased policynetwork (prescriptivelearning) to learnhow tochooseamovein thegame (the action). The Monte Carlo sampling based self-playing (predictive learning) Browne et al.(2012)isused todeterminewhetherthemove(theaction)is correctand howtoupdate thepolicynetwork.Such aspiral loop makes the system become better andbetter.

Followingasimilarlogic,anintelligentsystemforvehicleintelligencetest exploresthespaceofstate,policy and state transitions inaloop as illustrated in Fig.5b.

Taskdescriptionpart solveshow togenerate new tasks fortesting.Themaingoal ofthis partisto set up and refine a methodology, which can guide to set up environments for the followingtests. For tasks in every scenario, the descriptor will break it down into several task atoms, andthen function atoms and functionalities. The connection between these elements will bedescribedas well.

Given detailed descriptions of tasks, task sampling part will explore the policy space to choosechallengingtasks. There wereseveralways toreach thisgoal (Zhaoet al.2017; Evtimovet al.2017).However,none oftheexistingapproachesisself-motivated.

To implement rapidly adaptive intelligence test, we consider challenging task sampling as adecision process which can be formalized as a 4-tuple (S,A,P,R)(S,A,P,R). The state stst in thisdecision process is the confidence we had on the performance of vehicle intelligence at time tt,andthe action atat is thetesting procedures that we chooseto updateour confidence.Pra(st,s′t+1)Pra(st,st+1′)denotesthe probabilitythatwechooseaspecifictaskwilllead to another understanding level s′s′ from state ss, and the reward rtrt gives how muchconfidencewegained at time tt.

Undersuchsetting,thelong-termunderstandingofvehicleintelligencecanbeformalizedasVπ(s)=E(∑t=0∞rt|s,π).Vπ(s)=E(∑t=0∞rt|s,π).

(1)

Thegoal oftask sampling partis to findan optimal policyπ∗π∗which canmaximizethe long-termunderstanding

π∗=argmaxπVπ(s).π∗=argmaxπVπ(s).

(2)

With a detailed description of the task and sampling policy, testing (simulation) part can finallysolve how to label testing results by actually generate the test scenarios and see how well thevehicle intelligence can perform. Two kinds of relationships need to be labeled during thisprocedure.Oneisthe relationshipbetweenvehicleintelligenceanditsperformanceundercertainenvironments. The evaluation of vehicle intelligence is the main output we want from anintelligenttest system,and suchresults canhelp ussamplebetter tasksin thenext episode.

Another is the relationship between the test and real environments. Differences of twoenvironments and behaviors of subjects (e.g., the characteristic of traffic situations and featuresofvehicledynamics)needtobe paired,sothetask description canbemoredetailedandrealisticin thenext loop.

Theabove frameworkof intelligencetestingsystemforvehiclesisdesignedbasedonthefollowingconsiderations:

First, we can hardly know in advance whether intelligent vehicles will behave unless we testthem. So, we cannot directly answer which task is most challenging. So, we need to graduallybuild our knowledge of testing from zero knowledge state and adopt a prescriptive learning style.Second, testing can actually be viewed as a self-labeling (prediction learning) process. Since wedo not know the outcome of a special test, we have to wait and let the results label whether thevehiclecan pass thetestor not.

Third, it requires huge an amount of resources and a long time to cover most of thefunctionalitiesthatavehicle intelligenceshouldhave.So, weneedtofind anefficientway tomaximizethe long-termunderstanding of vehicle intelligence.

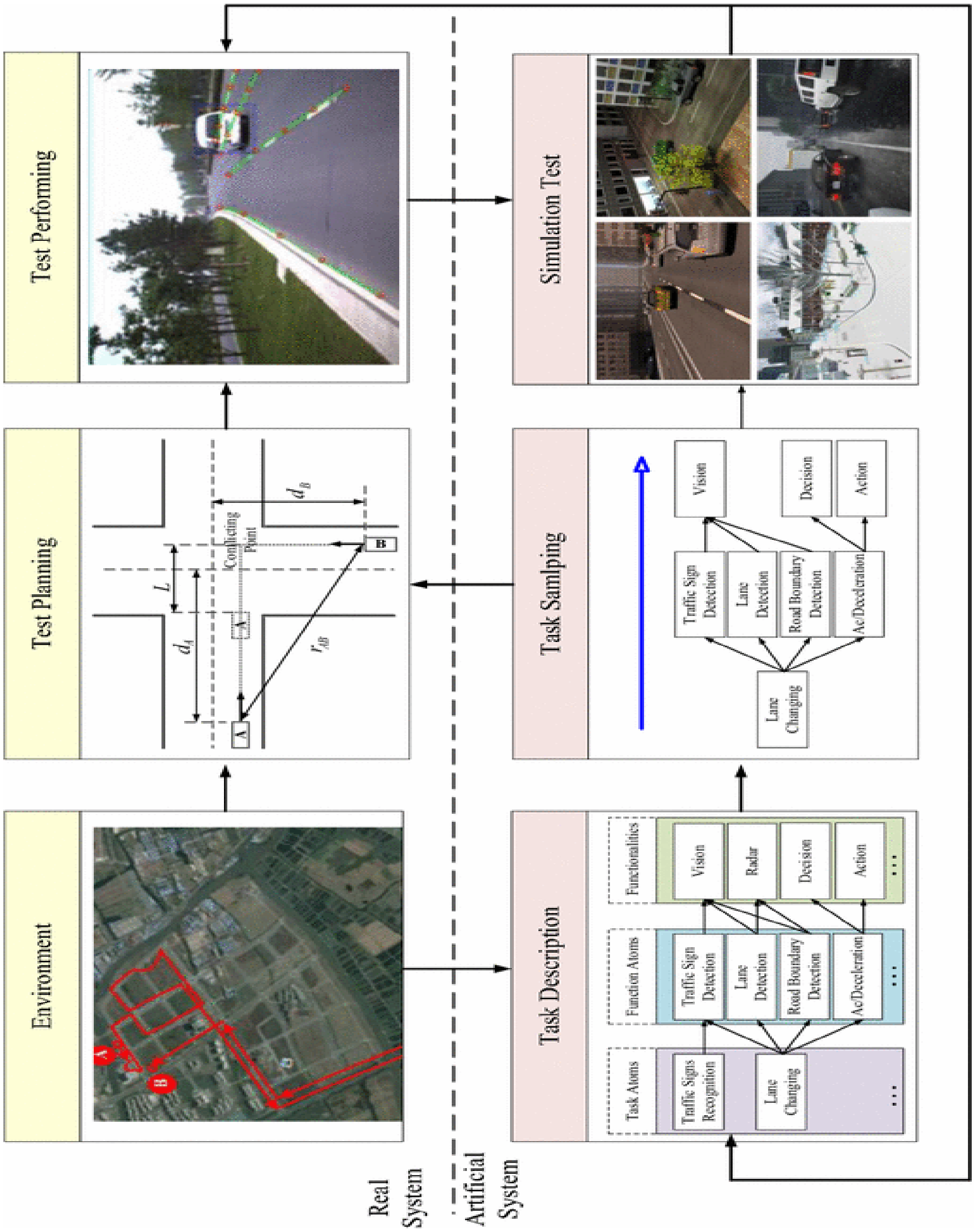
Wedonot restricttheimplementation detailsofsuch tasksamplingdecision problem.Wearenow testing whether deep reinforcement learning needs to be used. We will write a dedicatedpaperto report the progress in the near future.

Paralleltestingforvehicleintelligencetest

When the detailed task is assigned, simulation-based tests can then be applied for tests ofintelligent vehicles. Researchers began to show interests in accurately reproducing humanbehaviors(Wangetal.2017b).While,currently,mosteffortshadbeenputintogeneratingvirtualimage/video data as inputs of intelligent vehicles, since most information is collected by visualsensors(Gaidonet al. 2016; Santanaand Hotz2016; Liu et al.2017). Someapproachesfirstacceptedreal2Dimage/videodata,thenbuiltthecorresponding3Dobjectmodels in rendering engines, and finally generated 2D virtual image/video data as sensing inputsof intelligent vehicles (Gaidon et al. 2016; Richter et al. 2016; Greengard 2017). Some otherapproaches directly employed GAN to generate new virtual 2D image/video data from existingreal 2D image/video data (Santana and Hotz 2016; Gatys et al. 2016; Liu et al. 2017). The latestapproach mixed these two methods to produce more virtual data as "real" as possible and as"rich" aspossible(Veeravasarapuetal.2015;Wang etal. 2017a; Roset al.2016).

In this subsection, we propose a parallel system framework that combines real-world andsimulation-world for vehicle intelligence test. As illustrated in Fig. 6, a vehicle intelligence testcan be decomposed into three parts, the environment, the test planning part, and the testperforming part. Following the logic we predicated in the last subsection, a parallel system canbebuilt by connecting thesethreeparts.

Fig. 6



Ademonstrationofparallelsystemforvehicleintelligencetest

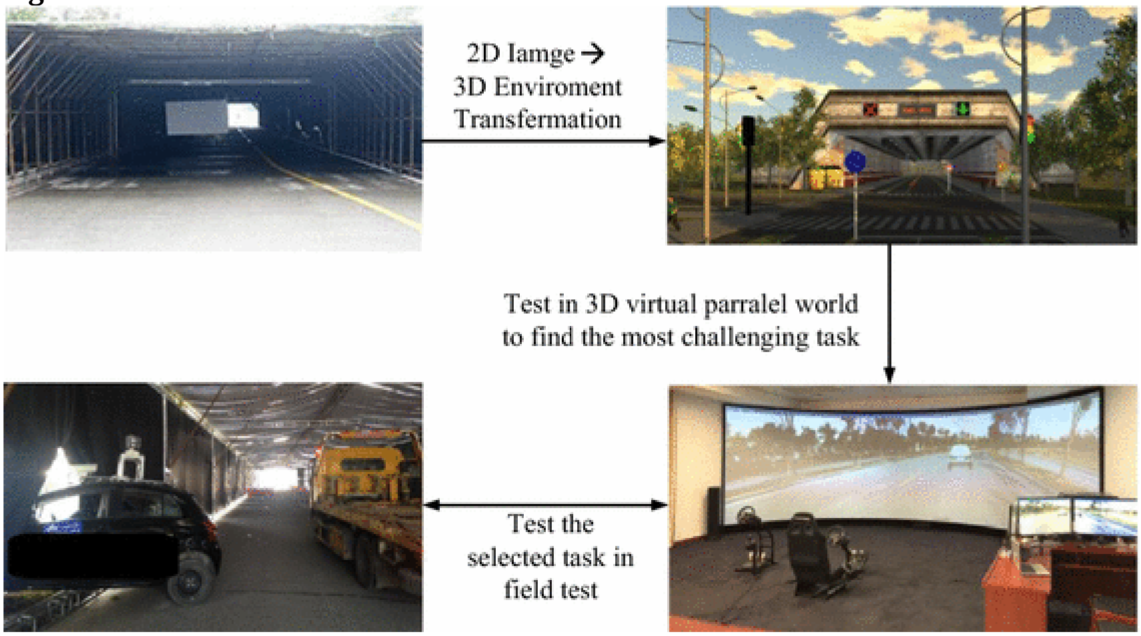
Theloopofintelligencetestinthe parallelsystemstarts fromareal environment,whichisanarea withintersections,traffic signsandotherelements ofsomespecificdrivingscenarios. Depending on the mission, a task description, which is a directed acyclic graph (DAG) can firstbe initialized according to some prior knowledge. It breaks down the task into task atoms,function atoms, and functionalities atoms. Then, it establishes the connection between theseatoms. The weights of DAG are estimations of confidence gained by performing a certain step.Basedonthedescription,anagentwillbetrainedtoplanthebestschedule oftasks.Forexample,if there are two task atoms, traffic signs recognition and lane changing, the optimal agent willfind that, the traffic signs recognition atoms can actually be neglected, since most of theconfidencescanbegained byperformingthelane changingatom.Weighing theprosand consof different routes in the DAG, the agent prunes some routes and picks important ones to perform.The most important tasks will be checked in the real environments and the less important oneswillbetested in simulation.

Once the schedule is provided, a special task can be tested. Depend on the confidence of testaccuracyandtheimportanceof atom,wecancalculateaweightedscorebased ontheresultsinboth real and simulative environments. Meanwhile, data generated in the real environment willbe fed into the simulative environment, so the simulation can be improved continuously. Theloop in the real system and the artificial system is asynchronous, and multiple loops can beperformedin theartificial systemwhile oneloop in thereal environment.

Comparing totraditionalsimulativeenvironments,theparallel systemforvehicleintelligencetesthastwomaindifferences.Firstof all,theparallelsystemisnotmerelyareflectionoftherealsystem, but a combination of two systems with equal status. Things happened in both systemswill affect each other and form a closed self-boosting loop. Second, the parallel system is alearning system which can evolve over time. Several key components in the artificial system(e.g., the task sampling agent and simulative environment) are data-driven instead of arbitrarymodels.Such designs maketheparallel system moreautonomous andquantifiable.

It should be pointed out that a prototype parallel intelligence testing system had already beenbuilt in Changshu city, Jiangsu Province, China and had successfully supported the 2016 and2017IntelligentVehicle FutureChallenge(IVFC).AsshowninFig.7,sometestingvehiclespassed a number of relatively simple tasks but failed to do so when encountering the mostchallengingtask thathadbeen foundin virtual testsin thevirtualparallel world.

Fig. 7


[redacted]

Ademonstrationofusingparallelsystemtofindthemostchallengingtask

Discussions

Ethicalproblems

Most researchers, starting from Turing, have implicitly assumed that human will do the rightthingstofinishthestudiedtasksandintelligent machinesshouldlearntodo thesameright thingto finish the studied tasks. So, we only need to check whether intelligent machines do the samethingsas human, during intelligencetest.

However, in some cases, even a human will feel difficult to know what should be done. Onefamous case is the so-called trolley problem that has mulled for about 50 years. Suppose arunaway trolley speeding down a track to which five people are tied. You can pull a lever toswitchthetrolleytoanothertracktowhichonlyonepersonistied.Wouldyousacrificetheoneperson to savethe other five, orlet the trolley kill thefivepeople?

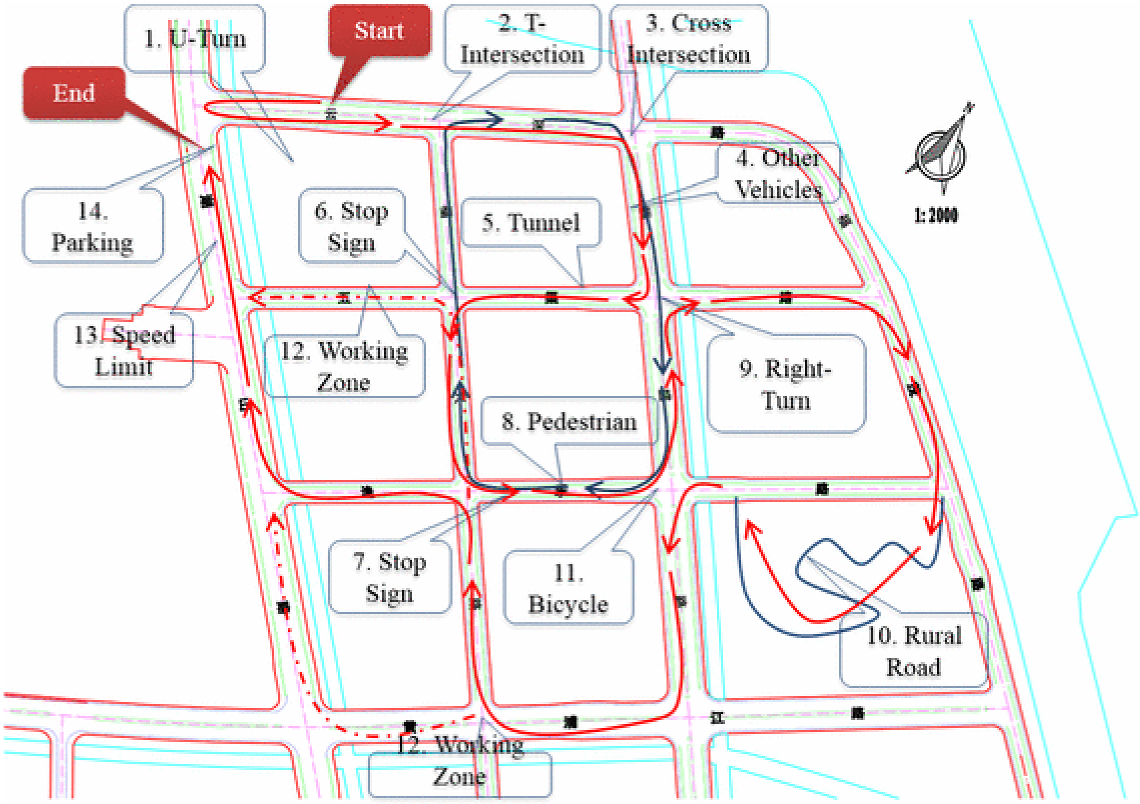
Trolley problems caused much debate that we do not want to discuss in this paper. If we think ofhumans as moral decision-makers and take artificial intelligent machines as moral agents thatactually replace our capacities, we can hardly find a commonly accepted answer (Goodall 2014;Kumfer and Burgess 2015; Maurer et al. 2015; Thornton et al. 2017). If we assume thatintelligent machines reason and act just what human had told them to do, the only decision-makers are human but not intelligent machines. In this paper, all such problems involved ethicaldecisionmakingarenotconsidered. As aresult,wedonotdiscusshow todesignanyintelligencetest tasks forethics, sinceweshould pay to Caesarwhat belongs to Caesar–and Godwhatbelongs toGod.

Real-timeandautomatedevaluationoftestingresults

One major difference between Turing test and the new approach of intelligence test is theselectionofthejudge. Turingchosehumanto bethejudge toarbitratewhetheramachinehasintelligence in Turing test; while many new intelligence testing systems use machines toarbitrate. This is not only because we have a more clear description of tasks in many recentlystudied intelligence test problems, but also because a human is unable to accurately examinemanyresults of intelligencetest without thehelp ofmachines.

Let us still use testing for intelligent vehicles as an example. To save time and money, severalindependenttasksofanintelligentvehicleareoften linkedalongaspecialpathofthevehicleandare tested sequentially in practice. For instance, a vehicle needs to finish 14 tasks in 2017Intelligent Vehicle Future Challenge, including: (1) make U-turn, (2) pass the signalized T-intersection, (3) pass the non-signalized cross-intersection, (4) pass other vehicles, (5) pass thetunnel in which GPS is blocked, (6) recognize the stop sign dedicated for vehicles and behaveappropriately,(7)passanotherstopsigndedicatedtoschoolchildren, (8)giveway topedestrian,
(9) make a right-turn, (10) pass the rural road, (11) give way to bicycle, (12) pass the workingzone,(13)recognizethe speedlimitandbehave appropriately,(14)parkintotheassignedberth;seeFig.8for an illustration.

Fig. 8

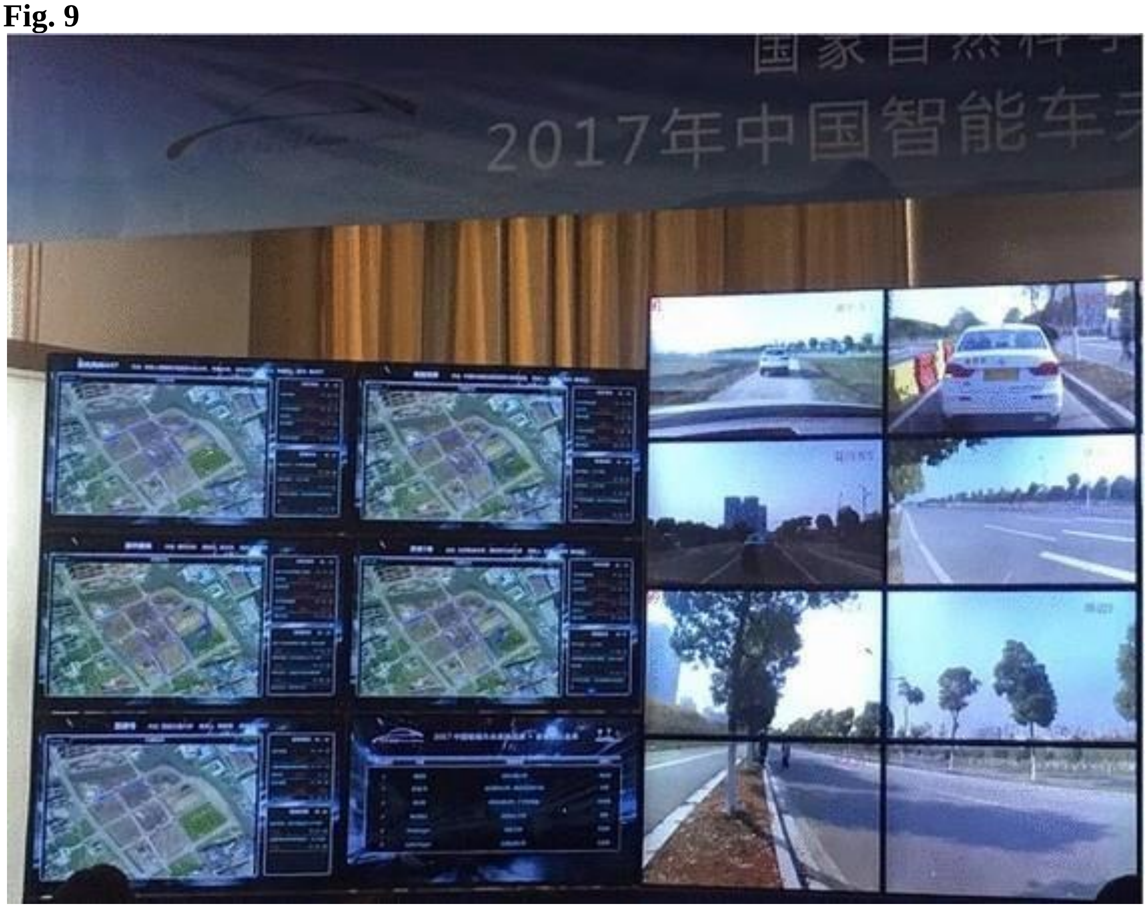


Anillustrationofdifferenttest tasksfor2017 intelligentvehicle futurechallenge

Usually, we do not require the vehicle to stop after it passes a task. In order to achieve a real-time and automated evaluation of the testing results for each individual task, researchers hadused vehicle-to-everything (V2X) communications to connect the onboard sensors and controlcenter, share a number of information of vehicle (e.g., position, speed, ac/deceleration rate) andrapidlycalculatetheperformancevaluesofeach taskbasedonthecollectedinformation.Such amethodreduces theburden of testingand becomes increasingly popular.

Figure 9gives a demonstration of the evaluation system designed by Tsinghua University andQingdao *VIPioneers* company, for 2017 Intelligent Vehicle Future Challenge. The left screensshow the real-time trajectories of 5 vehicles that were running in the Challenge and their ranks.The right screens show the real-time monitoring video data collected from the cameras that wereinstalled inside the tested vehicles, the cameras that were installed inside the following arbitratorvehicles, and the roadside cameras. All the data were transferred to the testing center via variousways, including V2X communication, 4G wireless communication, and optical fibercommunication.

Fig. 9

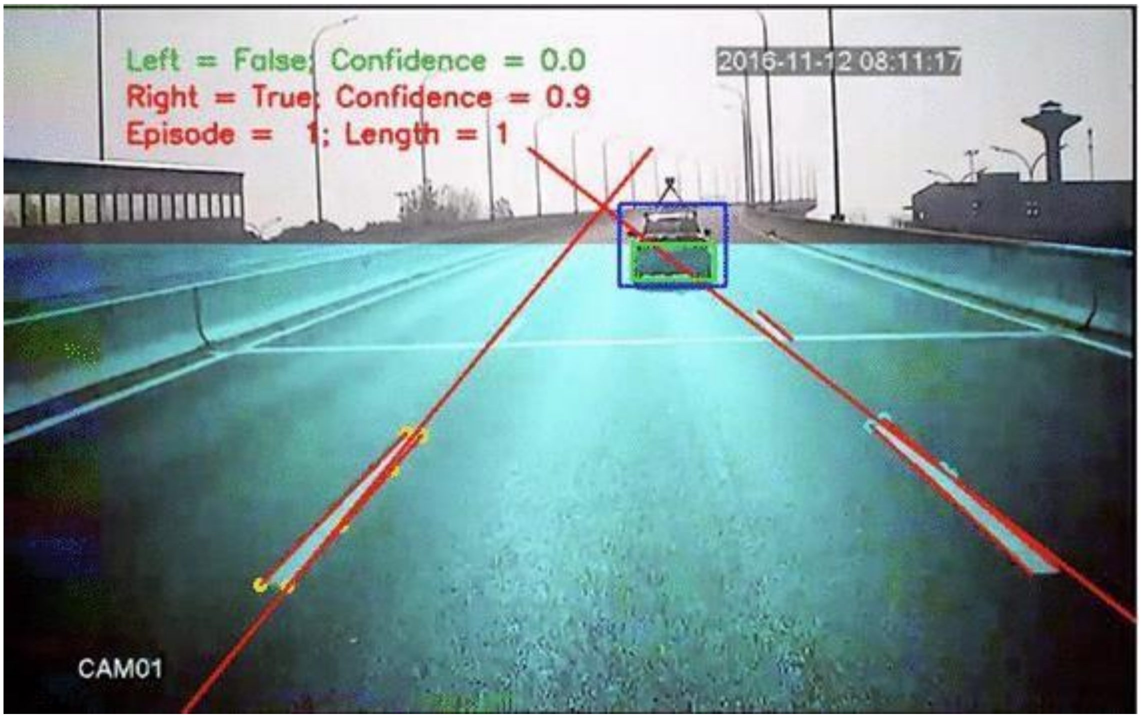


A demonstration of the real-time automated evaluation system designed for vehicle intelligencetestsof 2017 intelligent vehiclefuturechallenge(IVFC)

In 2009–2015 Intelligent Vehicle Future Challenges, human judges determine how to evaluatethe performance of intelligent vehicles. Such manual evaluation is tedious, time-consuming andprone to error. In Intelligent Vehicle Future Challenge 2017, most evaluations were done bymachines based on the measured data collected from various resources. Comparisons show thatthe evaluations became more accurate and much quicker. For example, in the previous match,human judges stared at the dashboard to check whether the tested vehicle is speeding. Based onthe high-resolution position information measured via BeiDou navigation satellite system(Wang 2016a, b), we can easily reconstruct the whole trajectory of the tested vehicle anddeterminewhenand wherethe vehicleis speeding.

For another example, Fig. 10gives a demonstration of the deep learning (LeCun et al. 2015;Goodfellowetal.2016)basedautomated evaluationsystemdesignedtorecognizewhetherthevehiclehadcrossed thelaneboundaries (You 2017). This systemused YOLO(Redmonet al. 2016; Redmon and Farhadi 2016) to recognize the tested vehicle, based on the video datacollectedfromthejudgingvehiclethat followsthe testedvehicleall thewayalong.Itcan helpcatch each incorrect crossing of the lane boundaries during the long-time running tests andgreatlyrelievethe burdens of human judges.

Fig. 10


Left = False; Confidence = 0.0
Right = True; Confidence = 0.9
Episode = 1; Length = 1
2016-11-12 08:11:17
CAM01

Ademonstrationoftheautomatedevaluation systemdesignedtolanedeparturewarningHuman–machineintegrated testing

However,wedonotclaimthatweshouldremovehumanfromtestsofartificialintelligence. Inthecurrent stage,humanparticipates inevery aspectof artificial intelligencetests.

First, human experts are heavily involved in the description of test tasks. Indeed, every test isdescribed by a certain kind of language that is established by human. Till now, we do notobserve any artificial intelligent machine generates its own language. The capability of anintelligent machine and that of the corresponding testing system is constrained by humandesigners,too.So,wealways resorttohumanexpertsto makesubstantive improvementforthedesignand tests of artificial intelligence.

Second, human experts also help to design the most challenging tasks in many intelligentapplications,accordingtotheirexperience andintuitionthatisgainedthroughfinishingthesametasks. For example, researchers inquired human drivers to set up different testing levels fordifferenttasksforintelligent vehicles (Zheng etal.2017).

Third, human experts usually monitor the testing process and take the final responsibility toguarantee that the testing results are correct. As shown in Fig. 8, the automated evaluationsystem designed for 2017 Intelligent Vehicle Future Challenge provides real-time visualizationfor human experts. This enables human experts to track the entire progress of testing, monitorwhether the automated evaluation system works well, and gain an intuitive understanding oftesting result. Such a hybrid-augmented intelligence (Zheng et al. 2017) setting helps combinebothhuman andmachines tobetter evaluatethe performanceofintelligentmachines.

It should be pointed out that, till now, human's intelligence levels are tested via the tasksdesigned by human experts (Sternberg and Davidson 1983; Sternberg 1985; Mackintosh 2011;Rindermann et al. 2016; Ohlsson et al. 2017). Can we use some tasks that generated by machinesvia some technologies similar to what we had discussed above? We believe this interestingquestionwill attract moreattention in thenear future.

Testingasameasurementofintelligencelevel

SAE International defines the six levels of driving automation, from no automation to fullautomation in 2016 (SAE J3016 2016). However, there is not a clear description of thecorresponding test tasks. So, it becomes widely accepted that testing results for intelligentvehicles can be viewed as a measurement of intelligence level. Only if a vehicle passes all thetasks that are designed for a special level of driving automation, we can claim that this vehiclehassuch an intelligencelevel.

Intelligent machines are becoming smarter and smarter now. Now, intelligent machines hadbeaten all human players in Shogi, chess and Go games (Silver et al. 2017a, b). The AI 'TopGun' beat the military's best pilots repeatedly. It is probably safe to say that all artificialintelligenceresearchers aimtodesignandimplementsomemachinesthatbeathumanincertainkinds of tasks, since aeronautical engineers had shown that they can do something better thanmakingmachines fly so exactly likepigeons (Russell and Norvig2010).

Maybe in the future, we should renew our definition of artificial intelligence as "Artificialintelligence is intelligence (that is similar to, or the same kind as, or even superior to humanintelligence) exhibited by machines (in the same task)". At the current stage, human experts arestill the major referring standard for tests of artificial intelligence. Sometimes in the future, theperformancethatanintelligentmachinecouldachievewillserveasanew evaluatingstandardofintelligencelevelinstead.

When we cannot enumerate all the test tasks, it becomes increasingly complex to set a fairmeasurement of intelligence for two different artificial machines dedicated for the same purpose.For example, in Go game, researchers used the Elo rating scores (Elo 1978; Coulom 2008; Silveret al. 2017b) that were computed from evaluation games between different players, becauseconventional static rating systems do not consider time-varying strengths of players. When theinformation that we can observe from the results is limited, things become even harder. Asshown in the recent algorithms designed for the poker game, analyzing results indicated that weneed to build special algorithms to drill the useful guide so as to boost the intelligent machines(Moravčíketal.2017;BrownandSandholm2017).We believethatmoreresearcheffortswillbeput into this researchdirection.

Explainabletestingofintelligent machines

Itshouldbealsopointedout that,justlike Turinghad done67years ago,wefocus ontheoutsidebehaviors of human/machine rather than the inside mechanism that generates the outsidebehaviors. If a machine has passed all the tasks according to its outside behaviors, we admit itsintelligence in this special field. However, we usually know neither what the best way to finishallthesetasks is, norhow human finish thesetasks.

Nowadays, intelligent algorithms and machines become more and more complex. Someone iscalling them 'black box', since it becomes harder to interpret what these algorithms andmachines are doing. However, intelligent machines coded in simple rules seem do not work aswell as some state-of-the-art 'black boxes'. Actually, if we assume that the latest machinelearningtechnologyhas"theabilitytolearnfrom testingresultsandimproveitselfautomaticallywithout being explicitly programmed, we may find that these machines will be naturally hard tointerpret.Otherwise, we can turn them backto explicit codes.

To the best of our knowledge, few studies give a widely-accepted generalizable way to combineinside mechanism design with outside behavior validation of artificial intelligence. We think thisnewdirection may bring some interestingfindings in the nearfuture.

Testingasanessentialpartofartificialintelligencesoftwaredevelopmentprocess

Because artificial intelligence is coded and implemented on computers, we need to highlight theimportance of software development of artificial intelligence. The lack of reproducibility andreadability has already hindered the development of AI techniques, since researchers can hardlyrelyon animplementation that canhardly beproofedor understood tofurther their research. A proper design of AI development loop can help to alleviate such situation. Test-drivendevelopment (TDD) has already been widely adopted in modern software development process.The basic idea of TDD is to organize the development cycle as a repetition of a very shortdevelopment cycle: First turn the requirements into very specific test cases, and then improve thesoftwareto passthetests.In suchdevelopment process,thereliabilitycan beguaranteedif wesetthe test properly, and the readability of software can be improved as well, since it is organized asthecollection ofsimple componentsto each fulfillaspecificrequirement.

The development of AI software can be profited from such development methodology, if somecritic problems are solved. Despite the unclear definition of requirements which can be handledby the method we proposed in the last section, the major problem is the lack of testing anddebugging tools. Software testing had already taken an essential part of software development.Almost all state-of-the-art commercial software developing tools provide thorough support fortesting at different phases (Huizinga and Adam 2007; Ammann and Jeff 2017). However, mostcurrent software/toolbox for building artificial intelligence lacks convenient testing tools anddebuggers. We wonder software/toolbox for building artificial intelligence could be viewed asSoftware 2.0 (Karpathy 2017). We expect more attention could be drawn to this important issue.Life-longlearning and life-long testing

Researchers are developing more and more powerful testing methods of artificial intelligence,just like what they had done for design methods of artificial intelligence. However, all thechanges take time to complete. Similar to the evolutionary history of machine learning, it seemsthat machine testing will take a relatively long time to become strong enough to characterizewhat a truly intelligent machine should be. We cannot give a precise prediction of the time whenan intelligent vehicle can drive in all kinds of situations. So, we borrow the term “life-long” fromlife-longlearning(ChenandLiu 2016)andname thisevolutionprocessas“life-longtesting”.

Moreover, it should be emphasized that we should always take the design and testing ofintelligentvehicleas awhole. Theknowledgeoftesting willbefedback tothe designpartofintelligent vehicle and will be used to further improve the intelligence of intelligent vehicles.Such a spiral loop helps make intelligent vehicle into practice in every automobile lab andmanufactory.

In precision machining industry, we continuously employ low-level machines to build morepreciseshigh-levelmachines.About 400 years ago,wecanonlymakesome simplegadgets.Now,we had achieved a great success and become able to make many complex things like CPU andGPU. Similarly, in artificial intelligence research field, smart machines are used to build evensmarter machines now. Fortunately, we are now witnessing such a great change in artificialintelligencedevelopment.

Testingasaneconomical opportunity

The ongoing artificial-intelligence revolution brings changes in enormous social lives andeconomic opportunities (Harari 2017). Humans are pushed out of some part of the job market byintelligent machines (Fagnant and Kockelman 2015; Fisher et al. 2016). For example, someaggressiveresearchers advocatedto totallyreplacehumandrivers inthenearfuture.

Meanwhile, AI generates a wide range of new jobs, including some new jobs for tests of AI.Using crowdsourcing(Wanget al.2016b), we canhirea numberofhumantolabel the videodata collected in streets and plot the bounding boxes of vehicles/pedestrians, since we needground truth data to train the artificial intelligent systems for environment recognition andautonomousdriving.Severalcompanies inChinahadhired alotofretiredpeople todosuchjobsand gained gigabytes of useful in return. We hope that, in the future, many people who had beenreplacedbyintelligent machinescould jointhebuildingprocess ofmoreintelligentmachines. Thisalsorequires ustobuild moreflexibleand powerfulsoftware, likeCompletelyAutomatedPublic Turing test to tell Computers and Humans Apart (CAPTCHA) (von Ahn et al. 2003;Georgeet al. 2017).

Crowdsourcing also leads to new risks of AI developing and testing. Tencent company hadrecently announced a critical vulnerability of Google's TensorFlow. Such vulnerability alloweshackers accessto AIcodebeingwrittenby programmers,jeopardizethetraining data,orconfusethe testing results (Liao 2017). So, we have to make far more efforts to make distributed tests ofartificialintelligenceintopractice.

Conclusions

In this paper, we discuss four major difficulties of carrying out the test of artificial intelligence,withaspecial emphasison theroleoftaskin intelligencetest.Wealsopresent ourexperiencesindesigningreliable intelligencetest for intelligent vehicles.

We explain our design of intelligence test by analogy with the structure of machine learningframework. The origin of this similarity lies in the fact that learning and testing are indeed twofacesofartificialintelligence.Fromthisviewpoint,weexplainwhyaparallelsystemframeworkforvehicleintelligencetestis needed.Such aframework shouldhavetwoimportant features. First,thewholetestingshouldbe formulatedasa loopbetween threeparts: taskdescription,tasksampling and task testing (simulation). This formulation allows us to gradually build ourknowledge of testing results and automatically finds the most challenging tasks to test. Second,the simulation tests should be executed in a mirror system so that we can produce more virtualdata as "real" as possible and as "rich" as possible. This will help us reduce both the time andfinancialcosts of testing.

However,theevolutionofartificialintelligenceonlyhelpstoreducehuman participation fromsome parts but not the core of artificial intelligence test. We still do not have an intelligentmachine can self-test, self-boost and upgrade without the help of human. The singularity of AI(Vinge1993; Kurzweil2005)is yet to come.