

**Department of Computer Engineering**

**Journal**

**Artificial Intelligence**

**Semester VI**

**Department of Computer Engineering**

**K. J. Somaiya College of Engineering**

**(Constituent College of Somaiya Vidyavihar University)**

**Academic Year 2023-24**

**Subject: Artificial Intelligence**

**List of Experiments for academic year 2023-24**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Topic** | **CO Mapping** |
|  | Study of AI projects and Turing Test | CO 1 |
|  | Implementation of family tree PROLOG program using condition-action rules based agent | CO 1 |
|  | Implementation of PROLOG program to solve a Crime mystery/medical diagnosis using Goal based agent architecture | CO 3 |
|  | Implementation of uninformed search algorithms – BFS,DFS, DLS for the given problem | CO 2 |
|  | Implementation of informed search algorithm(Greedy Best First search/A\*) | CO 2 |
|  | Implementation of alpha-beta pruning | CO 2 |
|  | Implementation of Constraint Satisfaction Problem | CO4 |
|  | IEEE Case study on learning/planning techniques | CO4 |
|  | Virtual Lab experiment - NLP | CO4 |
|  | Virtual Lab experiment – POS tagging : Hidden Markov Model | CO4 |

**Subject Incharges-**

Ms. Rohini Nair

Dr. Ayesha Hakim

Ms. Mansi Kambli

**Batch: Roll No.:**

**Experiment / assignment / tutorial No. 1**

|  |
| --- |
| **Title:** Study of Artificial Intelligent project. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** To study various AI projects for their architectures. Complexity, programming language, applications and other AI related concepts.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO1** | Understand the history & various application of AI and choose appropriate agent architecture to solve the given problem. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **http://en.wikipedia.org/wiki/List\_of\_artificial\_intelligence\_projects**
2. [**http://www.cs.cornell.edu/courses/cs478/2002sp/mllinks/interesting\_ai\_demos\_and\_project.htm**](http://www.cs.cornell.edu/courses/cs478/2002sp/mllinks/interesting_ai_demos_and_project.htm)
3. **http://homepages.inf.ed.ac.uk/rbf/AIMOVIES/AImovai.htm**
4. **“Artificial Intelligence: a Modern Approach” by Russell and Norving, Pearson education Publications**
5. **“Artificial Intelligence” By Rich and knight, Tata McGraw Hill Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

History and evolution of AI, Artificial intelligence: definitions and theories.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

AI research is highly technical and specialised and is also divided by several multidisciplinary technical issues. So far there are many projects those have been developed and are in progress to work on those issues. Students must learn the applications of intelligent robots by studying various such projects to know the depth and complexity of the course.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:**

Applications of AI, Current research and future research potential in the field.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Chosen Project Name:**

**Project Description:**

**Project category/field (Game, NLP etc):**

**Agent architecture: (State just name):**

**Programming language in which the project is/was developed:**

**Awards won by the project:**

**Nature of the project (Experimental/in use in real world):**

*(Explain in brief why experimental or how it is useful in real world)*

**Applications of the project:**

**Chat session with any chatbot(questions should vary difficulty level):**

**Post Lab Descriptive Questions:**

**1. ………….. is called the father of AI.**

A) James C Gosling

B) Dennis Ritchie

C) Alan Turing

D) Isaac Newton

**Answer:**

**2. In AI ………………………. is a combination of data structures and interpretive procedures.**

A) Knowledge

B) Meta-knowledge

C) Artificial Knowledge

D) Performance

**Answer:**

**Post Lab Descriptive Questions:**

1. **Define Artificial Intelligence in terms of human performance.**
2. **What is a Turing test?**
3. **Define an Omniscient agent. Are intelligent agents Omniscient?**
4. **What can today’s AI systems do?**
5. **What can today’s AI systems not do?**
6. **Design ten questions to pose to a man or a machine that is taking a Turing test.**

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 2**

|  |
| --- |
| **Title: Implementation of condition-action rules based agent using PROLOG** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** Developing a basic level agent program that runs on condition-action rules

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO1** | Understand the history & various application of AI and choose appropriate agent architecture to solve the given problem. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **https://www.csupomona.edu/~jrfisher/www/prolog\_tutorial/contents.html**
2. **http://www.csupomona.edu/~jrfisher/www/prolog\_tutorial/pt\_framer.html**
3. **http://www.doc.gold.ac.uk/~mas02gw/prolog\_tutorial/prologpages/**
4. **“Artificial Intelligence: a Modern Approach” by Russell and Nerving, Pearson education Publications**
5. **“Artificial Intelligence” By Rich and knight, Tata McGraw Hill Publications**
6. **“Prolog: Programming for Artificial Intelligence” by Ivan Bratko, Pearson education Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:** Intelligent Agent, Agent Architectures, Rule base Vs Knowledge Based approach

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:** Agent programs for simple applications need not be very complicated. They can be based on condition-action rules and still they give better results, though not always rational. The family tree program makes use of similar concept.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:**

Defining rules, using and programming with PROLOG

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

A simple agent program can be defined mathematically as an agent function which maps every possible percepts sequence to a possible action the agent can perform or to a coefficient, feedback element, function or constant that affects eventual actions:

*F*: *P* \* − >*A*

**Algorithm for ‘Condition-Action Rule Table’ Agent function:**

**function**SIMPLE-REFLEX-AGENT (percept) **returns** an action

**Static:** *rules,* a set of condition-action rules

*State*🡨INTERPRET-INPUT (percept)

*Rule*🡨*RULE-MATCH (state, rules)*

*Action*🡨*RULE-ACTION [rule]*

**Returnaction**

This approach follows a table for lookup of condition-action pairs defining all possible condition-action rules necessary to interact in an environment.

**Example Family Tree/disease-symptom mapping/ City map with their distances between them:**

A diagram of a family tree

Description automatically generated

**Base Knowledgebase:**

male(jaspal).

male(pargat).

male(bikram).

male(uttam).

male(amritpal).

male(nihal).

female(charanjeet).

female(harjit).

female(sarbjit).

female(oojal).

female(eisharr).

parent(uttam,jaspal).

parent(charanjeet,jaspal).

parent(uttam,amritpal).

parent(charanjeet,amritpal).

parent(jaspal, pargat).

parent(jaspal, bikram).

parent(harjit, pargat).

parent(harjit, bikram).

parent(amritpal, nihal).

parent(amritpal, eisharr).

parent(amritpal, oojal).

parent(sarbjit, nihal).

parent(sarbjit,eisharr).

parent(sarbjit,oojal).

**Rules:**

father(X, Y) :- male(X), parent(X, Y).

mother(X, Y) :- female(X), parent(X, Y).

grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

grandfather(X, Y) :- male(X), grandparent(X, Y).

grandmother(X, Y) :- female(X), grandparent(X, Y).

uncle(X, Y) :- male(X), sibling(X, Z), parent(Z, Y).

aunt(X, Y) :- female(X), sibling(X, Z), parent(Z, Y).

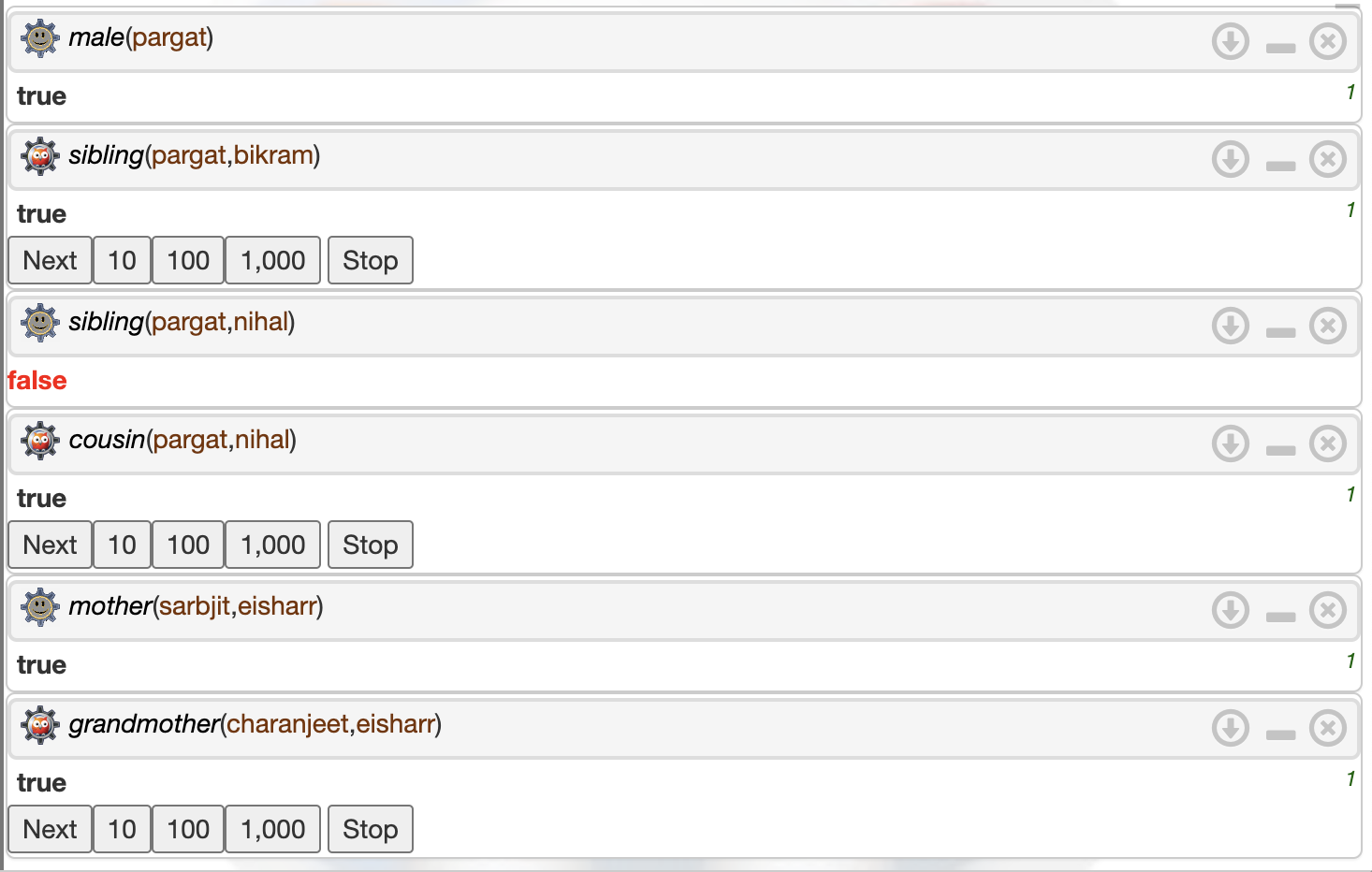
sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

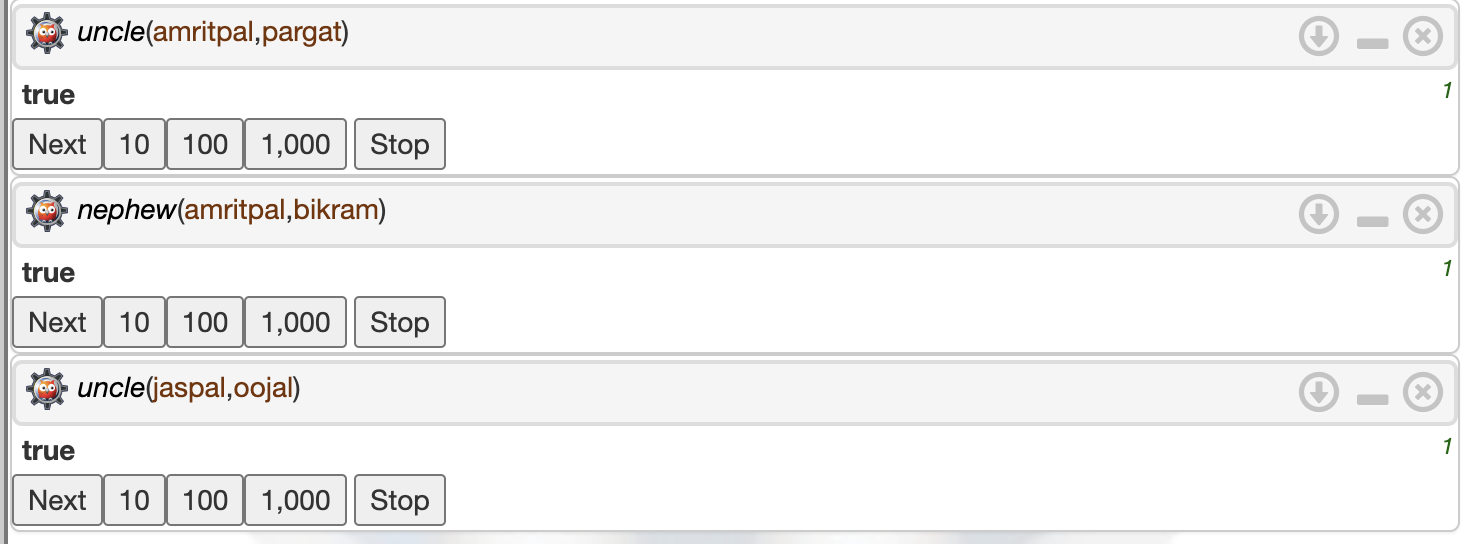
cousin(X, Y) :- parent(Z, X), sibling(Z, W), parent(W, Y).

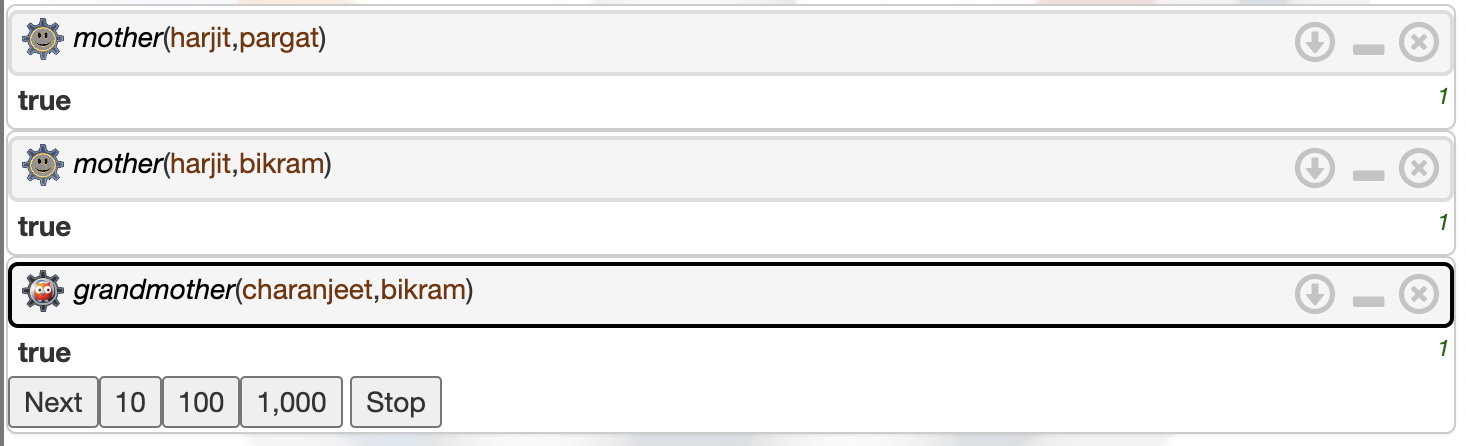
nephew(X, Y) :- male(X), sibling(Z, X), parent(Z, Y).

niece(X, Y) :- female(X), sibling(Z, X), parent(Z, Y).

**Some Sample queries and Outputs:**

****

****



.

**Post Lab Objective Questions**

1. **The PROLOG suit is based on**
   1. Interpreter
   2. Compiler
   3. None of the above

**Answer: Interpreter**

1. **State true or false**

There must be at least one fact pertaining to each predicate written in the PROLOG program.

**Answer: False**

1. **State true or false**

In the PROLOG program the variable declaration is a compulsory part.

**Answer: False**

**Post Lab Subjective Questions**

**1. Differentiate between a fact and a predicate with syntax.**

A fact is a straightforward assertion that illustrates the connection between two things. A predicate, followed by a list of constants or variables that correspond to the predicate's arguments, is used to represent a fact.

Think of a parent (nitin,rahi).

Parent is a predicate in this case, while nitin and rahi are constants that stand in for the predicate's arguments. This is a fact, not a rule, as shown by the period ('.') at the end of the line.

A more generic term, predicate, can be used to describe both facts and rules. An object relationship between objects is defined by a predicate, which is utilised in both queries and inferences. For instance: parent(Z, Y): grandparent(X, Y), parent (Y, Z).

Grandparent is the predicate in this case, and the variables X, Y, and Z stand in for the predicate's arguments. This is a rule rather than a fact, as the:- operator denotes. The parent predicate and the relationships between parents and children are used to characterise the relationship between a grandparent X and a grandchild Y.

**2. Differentiate between knowledgebase and Rule base approach.**

According to the knowledge base method, knowledge is primarily represented as a collection of claims or facts about the outside world. When the objective is to capture and represent the domain, this method is employed in numerous expert systems and knowledge-based systems.

**3. Differentiate between database and knowledgebase.**

A database is a group of organised data that is stored and accessed online. Databases are used to store and manage huge volumes of structured data, including customer information, sales transactions, and inventory levels. Data insertion, data retrieval, data update, and data deletion are all operations that databases are made to support. They are frequently applied in situations where quick and effective data access is necessary.

On the other hand, a knowledge base is a repository of knowledge that expert systems and artificial intelligence (AI) use to support decision-making and problem-solving. The domain knowledge of a specialist or group of specialists in a given topic is represented by a knowledge base. Unlike a database, a knowledge base can store more complicated information like rules, procedures, and causal links in addition to organised data.

**4. What is a ‘free variable’? Explain with an example.**

In Prolog, a variable that occurs in a query but has no assigned value is referred to as a free variable. To put it another way, a free variable is a variable that is used to indicate an undetermined value.

Take the Prolog query below, for instance: the parent (X, ashwini)

In this case, the name of Ashwini's parent is denoted by the free variable X. Prolog will search its knowledge base when this query is conducted for a fact that matches the predicate parent and the second argument harry. If a matching fact is discovered, it will combine the first argument with the unvalued free variable X.

In this method, the results returned by the search process are stored in free variables, which represent unknowns in Prolog queries. Prolog can express incomplete information in its knowledge base and identify all potential answers to a question by using free variables.

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 3**

|  |
| --- |
| **Title:** Implementation of Goal based agent architecture using PROLOG. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** To use the concepts of knowledge engineering to design and solve moderate complex problem.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO1** | Understand the history & various application of AI and choose appropriate agent architecture to solve the given problem. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **https://www.csupomona.edu/~jrfisher/www/prolog\_tutorial/contents.html**
2. **http://www.csupomona.edu/~jrfisher/www/prolog\_tutorial/pt\_framer.html**
3. **http://www.doc.gold.ac.uk/~mas02gw/prolog\_tutorial/prologpages/**
4. **http://classes.soe.ucsc.edu/cmps112/Spring03/languages/prolog/PrologIntro.pdf**
5. **“Prolog: Programming for Artificial Intelligence” by Ivan Bratko, Pearson education Publications**
6. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
7. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Agents, Agent Architecture, Programming with PROLOG

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

Knowledge is vast, uncertain and continuously changing. These properties of knowledge make it difficult to arrive at a result. A murder mystery is a kind of situation which depicts the uncertain nature of knowledge and also emphasizes the need of choosing right clauses from entire knowledgebase to make a decision. He goal based agent architecture and some knowledge engineering can help in solutioning of such problems.

The logical agents are complex but they can reason and learn from the actions and new precepts. They are less like acting and think like humans but more like acting and thinking rational agents.

Knowledge and reasoning play a crucial role in dealing with partially observable environments. A knowledge based agent can combine the general knowledge with current percept to infer the hidden aspects of the current state prior to selecting actions.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:**

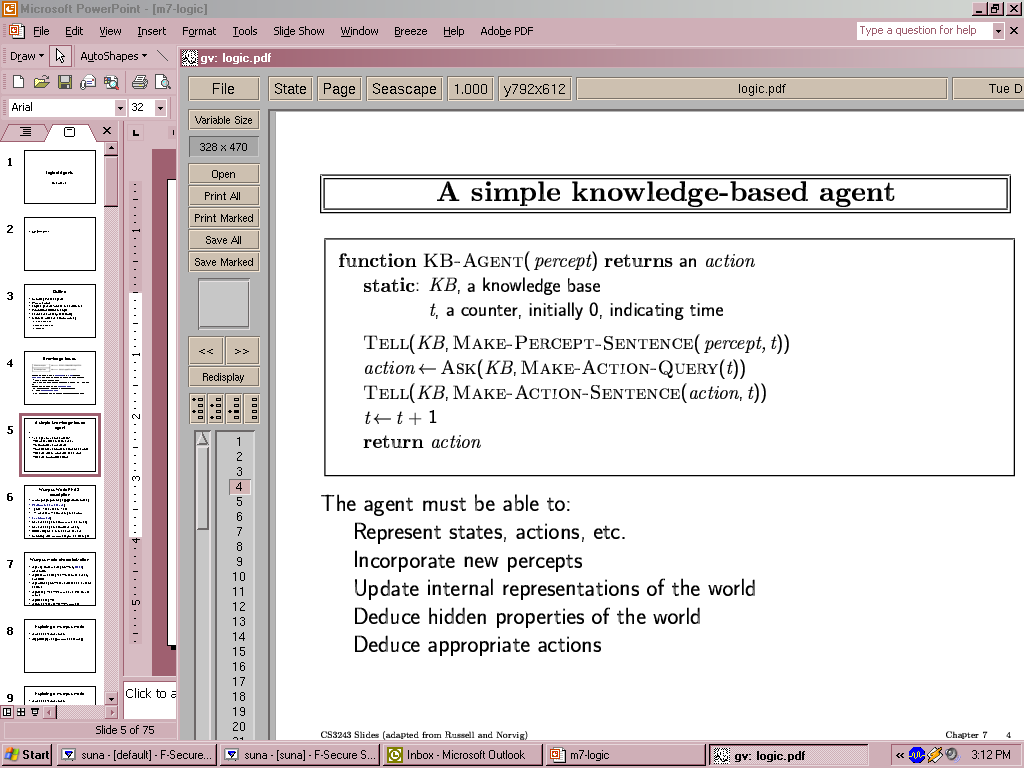
Knowledge engineering, implementing complex agent architecture, uncertainty in knowledge.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**The Knowledge Engineering Process**

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on vocabulary of predicates, functions and constants
4. Encode general knowledge about the domain
5. Encode description of specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

**Algorithm for KB-Agent:**



**Problem Statement:**

In the middle of last winter, **eight guests** were invited to a luxurious retreat at the Duke of York Grand Hotel. On the last day of their three-day getaway, the guests were free to vacate to their own occupations. **Mrs White and Reverend Green did some “gardening”** walking alongside the water fountains, **Colonel Mustard and Professor Plum played golf** (alone though, purposefully avoiding each other). The other guests spent their days either in their rooms or in the lounge, by the log fire. Later on in the afternoon, all the guests were indoors and **Colonel Mustard was seen playing cards with Reverend Green and Mrs Peacock**.

As the guests were called for dinner, they soon realised that Dr Black was missing. He was later found lying down on the floor of his bedroom. **Dr Black had been shot dead using an old fashion revolver**. Except from a **few muddy footprints** at the entrance of his bedroom, there was no other evidence left by the murderer.

Here is the list of all the guests for the weekend and the rooms they were staying in. Note that the hotel consists of twin bedrooms accommodating two guests per room. We also know that **three of the guests (Reverend Green, Colonel Mustard and Madame Rose) own a revolver that they brought with them and kept in their room**.



**Knowledge Engineering steps applied to chosen problem:**

1. Identify the task – The main goal is to find who killed Dr. Black.

The facts are the evidence found by Poirot. The main tasks include –

a) Finding all the suspects.

b) Finding all the prime suspects.

c) Finding the killer.

2. Assemble the relevant knowledge:

The relevant knowledge consists of the actors in the scene and the evidence. The actors in the scene are:

1. Dr Back
2. Reverend Green
3. Colonel Mustard
4. Prof Plum
5. Mrs Peacock
6. Madam Rose
7. Ms Scarlett
8. Mrs White

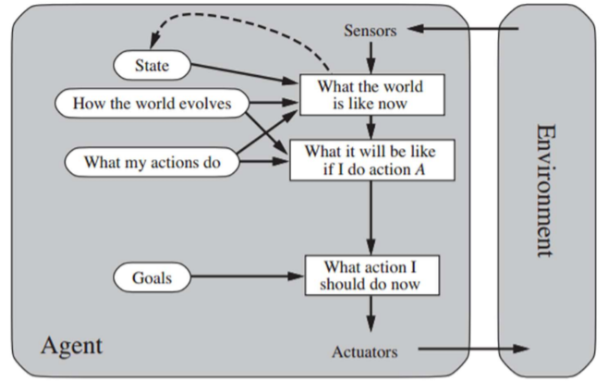
The evidences are:

1. Dr Black had been shot dead using an old fashion revolver
2. Few muddy footprints at the entrance of the bedroom

**Agent Architecture** *(Justify the blocks)***:** Goal Based Agent Architecture

These kinds of agents take decisions based on how far they are currently from their

goal(description of desirable situations). Their every action is intended to reduce its distance from the goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.



Various blocks with respect to above case study –

**What is the world like now?**

This particular question deals with the answer to what the surroundings are like at that

particular instance and what the environment is like. Dr. Black is dead there are multiple suspects . We have to find the killer.

**What it will be like if I do a particular action?**

This block checks for the state that is a direct result of conducting an action on a current state.

In this case, finding out whether the person is a suspect or not. Further, if a person is a

suspect, the next block can help us identify whether this individual is the criminal that the AI is looking for.

**What should I do now?**

Helps to take the appropriate action based on the percept and knowledge to move further towards the goal. Finally, this result is given to the environment via the actuators.

**Code:**

victim(drBlack).

suspect(reverendGreen).

suspect(colonelMustard).

suspect(professorPlum).

suspect(mrsPeacock).

suspect(madameRose).

suspect(missScarlett).

suspect(mrsWhite).

gardening(mrsWhite).

gardening(reverendGreen).

playedGolf(professorPlum).

playedGolf(colonelMustard).

playingCards(colonelMustard).

playingCards(reverendGreen).

playingCards(mrsPeacock).

ownsRevolver(reverendGreen).

ownsRevolver(colonelMustard).

ownsRevolver(madameRose).

smoker(missScarlett).

smoker(colonelMustard).

smoker(mrsWhite).

smoker(drBlack).

smoker(mrsPeacock).

room(room\_21).

room(room\_22).

room(room\_23).

room(room\_24).

room(room\_25).

stayIn(drBlack,room\_22).

stayIn(reverendGreen,room\_24).

stayIn(missScarlett,room\_21).

stayIn(colonelMustard,room\_24).

stayIn(professorPlum,room\_22).

stayIn(mrsPeacock,room\_23).

stayIn(madameRose,room\_21).

stayIn(mrsWhite,room\_23).

safe(X):- suspect(X), playingCards(X).

wentOut(X):- gardening(X).

wentOut(X):- smoker(X).

wentOut(X):- playedGolf(X).

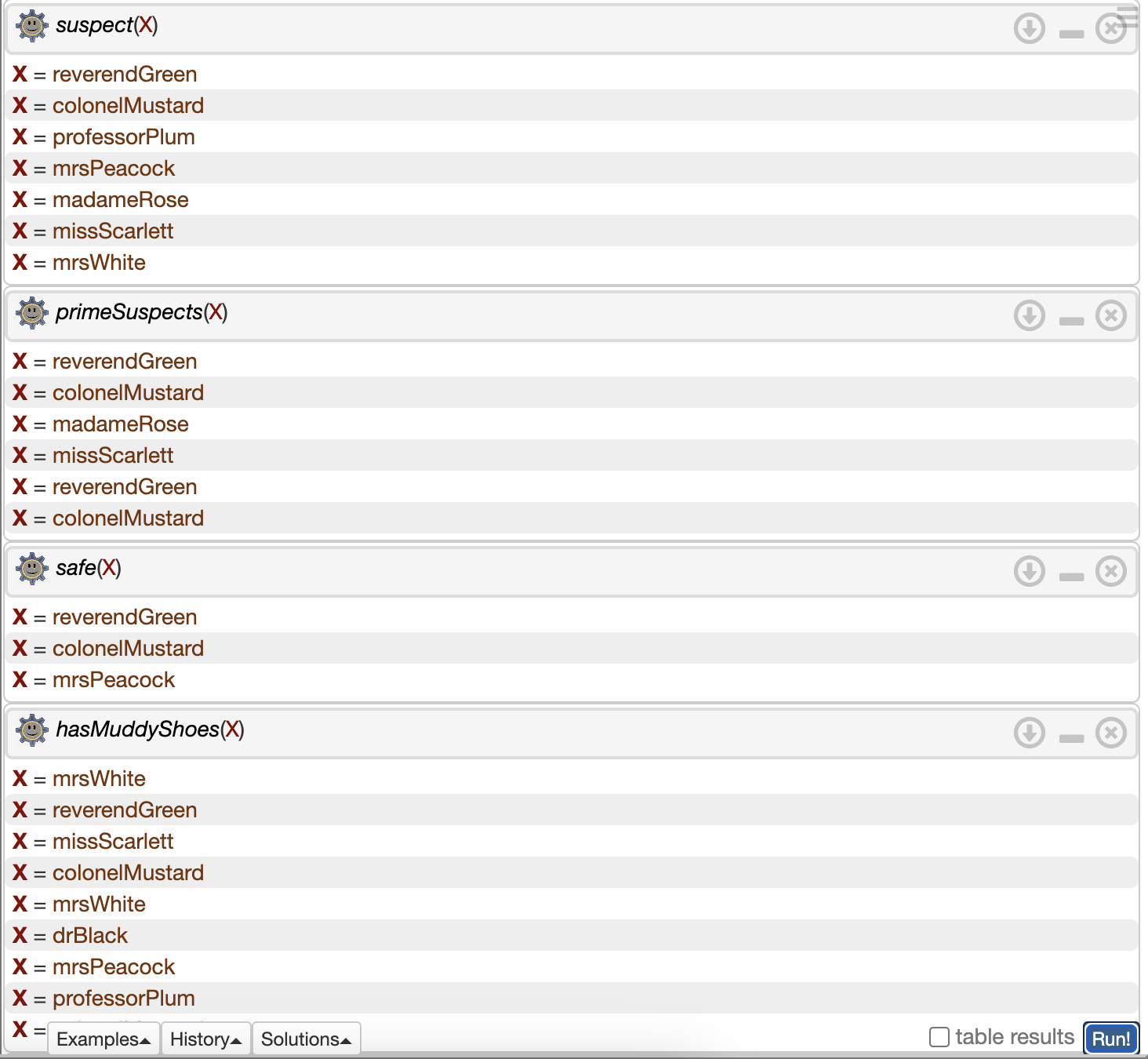
shareRoom(X,Y):- room(R), stayIn(X,R), stayIn(Y,R), X \= Y.

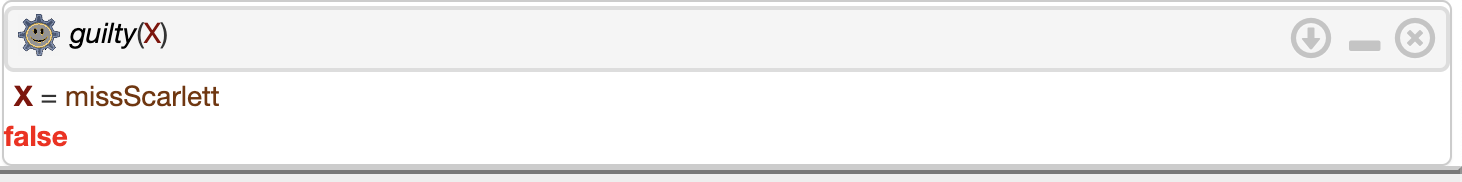
hasRevolver(X):- ownsRevolver(X).

hasRevolver(X):- shareRoom(X,Y), ownsRevolver(Y).

guilty(X):- suspect(X), wentOut(X), \+safe(X), hasRevolver(X).

**Output:**

****

****

**Team Members:**

**1. Pargat Singh**

**Post Lab Objective Questions**

**1. Which is not a Goal-based agent?**

1. Inference
2. Search
3. Planning
4. Conclusion
5. Dynamic search.

**Answer:** d) Conclusion

**2. Which were built in such a way that humans had to supply the inputs and  
interpret the outputs?**

1. Agents
2. Sensor
3. AI System
4. Actuators

**Answer:** c)AI System

**Post Lab Subjective Questions :**

**Explain the role of PEAS and task environment in choosing the agent architecture. Justify your answer with an example.**

Performance measure, Environment, Actuators, and Sensors is referred to as PEAS. The task environment must always be adequately specified before beginning to develop an agent.

The task environment aids in determining whether or not the setting is

* **Discrete / Continuous** - An environment is discrete if there are just a few clearly defined, different states of it (such as in chess); otherwise, it is continuous (For example, driving).
* **Observable / Partly Observable** – A system is observable if it is feasible to infer from the precepts the whole state of the environment at each time point; otherwise, it is only partially observable.
* **Static / Dynamic** -The environment is static if it stays the same while an agent is acting; otherwise, it is dynamic.
* **One agent / Many agents** - Other agents, which may be of the same or a different sort as the agent, may be present in the environment.
* **Accessible / Inaccessible** - Whether an environment is accessible or inaccessible depends on whether the agent's sensory apparatus can access the entire state of the environment.
* **Deterministic / Non-deterministic** - Environments can be either deterministic or non-deterministic depending on whether the next state of the environment is entirely determined by the present state and the activities of the agent.
* **Episodic / Non-episodic** − In an episodic environment, each episode consists of

the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

* Once the environment has been fully understood, the appropriate agent

architecture can be chosen based on the characteristics of the environment.

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No.**

|  |
| --- |
| **Title:** Implementation of uninformed search algorithm( BFS/DFS/DLS/IDS) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** Comparison and analysis of uninformed search algorithms

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO 2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russell and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**](http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf)
4. [**http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**](http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html)
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf** [**http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf**](http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf)
6. [**http://wiki.answers.com/Q/Informed\_search\_techniques\_and\_uninformed\_search\_techniques**](http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques)
7. **www.cs.swarthmore.edu/~eeaton/teaching/cs63/.../**UninformedSearch**.ppt**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:** Problem solving, state-space trees, problem formulation, goal based agent architecture

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

The AI researchers have come up many algorithms those operate on state space tree to give the result. Goal based agent architectures solve problems through searching or planning. Depending on availability of more information other than the problem statement decides if the solution can be obtained with uninformed search or informed search.

Its fact that not all search algorithms end up in giving the optimal solution. So, it states the need to have a better and methodological approach which guarantees optimal solution.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:**  Uninformed (blind) search, iterative deepening, greedy best first search, A\* search

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Uninformed searching techniques:**

* Breadth first search
* Depth first search
* Iterative deepening search
* Depth limit search

**Chosen Problem statement: 8 Puzzle problem**

The 8 Puzzle problem involves a 3x3 grid containing 8 numbered tiles and one blank tile. The objective is to rearrange the tiles from an initial state to a goal state using legal moves. The legal moves are swapping the blank tile with its adjacent (horizontal or vertical) numbered tile. The goal state is typically defined as the sorted arrangement of tiles, such as:

* **States**: Different configurations of the puzzle.
* **Initial State**: Any solvable configuration of the puzzle.
* **Transition Model**: Legal moves of swapping the blank tile with an adjacent numbered tile.
* **Actions**: Legal moves defined by the transition model.
* **Goal Test**: Check if the current state is the sorted arrangement of tiles.
* **Path Cost**: Each move has a uniform cost.

**State-space tree :**

**A notebook with writing on it

Description automatically generated**

**A notebook with writing on it

Description automatically generated**

**Code:  
BFS**

*from* collections *import* deque

def print\_board(board):

*for* row *in* board:

print(row)

def get\_blank\_position(board):

*for* i *in* range(3):

*for* j *in* range(3):

*if* board[i][j] == 0:

*return* (i, j)

def is\_valid\_move(i, j):

*return* 0 <= i < 3 and 0 <= j < 3

def swap\_tiles(board, blank\_pos, new\_pos):

i1, j1 = blank\_pos

i2, j2 = new\_pos

board[i1][j1], board[i2][j2] = board[i2][j2], board[i1][j1]

def is\_goal\_state(board, goal\_state):

*return* board == goal\_state

def get\_neighbors(board):

blank\_pos = get\_blank\_position(board)

neighbors = []

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

*for* move *in* moves:

new\_pos = (blank\_pos[0] + move[0], blank\_pos[1] + move[1])

*if* is\_valid\_move(\*new\_pos):

new\_board = [row.copy() *for* row *in* board]

swap\_tiles(new\_board, blank\_pos, new\_pos)

neighbors.append(new\_board)

*return* neighbors

def bfs\_8\_puzzle(initial\_state, goal\_state):

visited = set()

queue = deque([(initial\_state, [])])

*while* queue:

current\_state, path = queue.popleft()

*if* is\_goal\_state(current\_state, goal\_state):

*return* path + [current\_state]

*if* tuple(map(tuple, current\_state)) not in visited:

visited.add(tuple(map(tuple, current\_state)))

neighbors = get\_neighbors(current\_state)

*for* neighbor *in* neighbors:

queue.append((neighbor, path + [current\_state]))

*return* None

initial\_state = [

[8, 1, 3],

[0, 2, 4],

[7, 6, 5]

]

goal\_state = [

[1, 2, 3],

[8, 0, 4],

[7, 6, 5]

]

solution = bfs\_8\_puzzle(initial\_state, goal\_state)

*if* solution:

print("Solution found:")

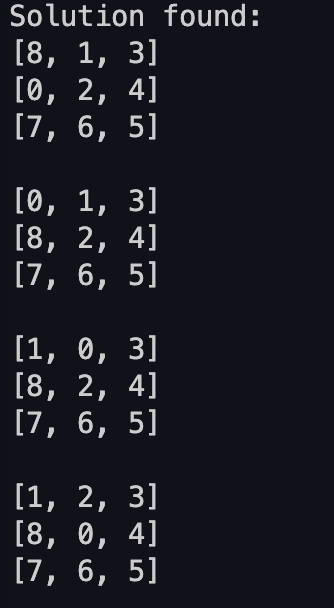
*for* step *in* solution:

print\_board(step)

print()

*else*:

print("No solution found.")

****

**DFS:**

*from* collections *import* deque

def print\_board(board):

*for* row *in* board:

print(row)

def get\_blank\_position(board):

*for* i *in* range(3):

*for* j *in* range(3):

*if* board[i][j] == 0:

*return* (i, j)

def is\_valid\_move(i, j):

*return* 0 <= i < 3 and 0 <= j < 3

def swap\_tiles(board, blank\_pos, new\_pos):

i1, j1 = blank\_pos

i2, j2 = new\_pos

board[i1][j1], board[i2][j2] = board[i2][j2], board[i1][j1]

def is\_goal\_state(board, goal\_state):

*return* board == goal\_state

def get\_neighbors(board):

blank\_pos = get\_blank\_position(board)

neighbors = []

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

*for* move *in* moves:

new\_pos = (blank\_pos[0] + move[0], blank\_pos[1] + move[1])

*if* is\_valid\_move(\*new\_pos):

new\_board = [row.copy() *for* row *in* board]

swap\_tiles(new\_board, blank\_pos, new\_pos)

neighbors.append(new\_board)

*return* neighbors

def dfs\_8\_puzzle(initial\_state, goal\_state):

visited = set()

stack = [(initial\_state, [])]

*while* stack:

current\_state, path = stack.pop()

*if* is\_goal\_state(current\_state, goal\_state):

*return* path + [current\_state]

*if* tuple(map(tuple, current\_state)) not in visited:

visited.add(tuple(map(tuple, current\_state)))

neighbors = get\_neighbors(current\_state)

*for* neighbor *in* neighbors:

stack.append((neighbor, path + [current\_state]))

*return* None

initial\_state = [

[8, 1, 3],

[0, 2, 4],

[7, 6, 5]

]

goal\_state = [

[1, 2, 3],

[8, 0, 4],

[7, 6, 5]

]

solution = dfs\_8\_puzzle(initial\_state, goal\_state)

*if* solution:

print("Solution found:")

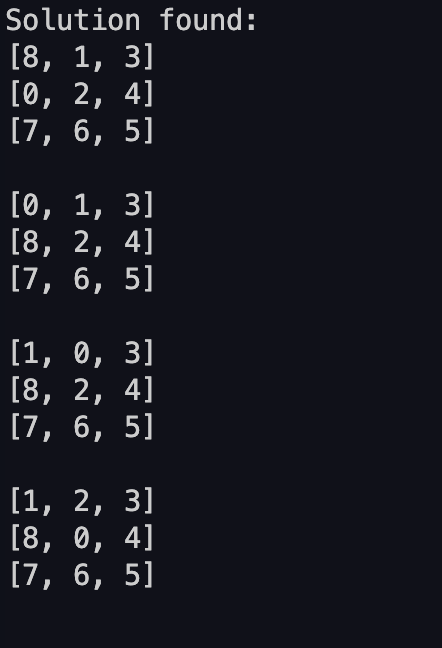
*for* step *in* solution:

print\_board(step)

print()

*else*:

print("No solution found.")

****

**DLS:**

*from* collections *import* deque

def print\_board(board):

*for* row *in* board:

print(row)

def get\_blank\_position(board):

*for* i *in* range(3):

*for* j *in* range(3):

*if* board[i][j] == 0:

*return* (i, j)

def is\_valid\_move(i, j):

*return* 0 <= i < 3 and 0 <= j < 3

def swap\_tiles(board, blank\_pos, new\_pos):

i1, j1 = blank\_pos

i2, j2 = new\_pos

board[i1][j1], board[i2][j2] = board[i2][j2], board[i1][j1]

def is\_goal\_state(board, goal\_state):

*return* board == goal\_state

def get\_neighbors(board):

blank\_pos = get\_blank\_position(board)

neighbors = []

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

*for* move *in* moves:

new\_pos = (blank\_pos[0] + move[0], blank\_pos[1] + move[1])

*if* is\_valid\_move(\*new\_pos):

new\_board = [row.copy() *for* row *in* board]

swap\_tiles(new\_board, blank\_pos, new\_pos)

neighbors.append(new\_board)

*return* neighbors

def dls\_8\_puzzle(initial\_state, goal\_state, max\_depth):

visited = set()

stack = [(initial\_state, [], 0)]

*while* stack:

current\_state, path, depth = stack.pop()

*if* depth > max\_depth:

*continue*

*if* is\_goal\_state(current\_state, goal\_state):

*return* path + [current\_state]

*if* tuple(map(tuple, current\_state)) not in visited:

visited.add(tuple(map(tuple, current\_state)))

neighbors = get\_neighbors(current\_state)

*for* neighbor *in* neighbors:

stack.append((neighbor, path + [current\_state], depth + 1))

*return* None

initial\_state = [

[8, 1, 3],

[0, 2, 4],

[7, 6, 5]

]

goal\_state = [

[1, 2, 3],

[8, 0, 4],

[7, 6, 5]

]

solution = dls\_8\_puzzle(initial\_state, goal\_state,3)

*if* solution:

print("Solution found:")

*for* step *in* solution:

print\_board(step)

print()

*else*:

print("No solution found.")

**A screenshot of a computer

Description automatically generated**

**Comparison of performance of uninformed Algorithm:**

Given the characteristics of the 8 Puzzle problem, which has a relatively small state space, BFS is generally the preferred choice because:

* The state space of the 8 Puzzle problem is not excessively large, so the memory-intensive nature of BFS is less of a concern.
* BFS guarantees the optimal solution, which is desirable for solving puzzles.
* BFS will explore the shallowest nodes first, which can be advantageous in situations where the solution is closer to the initial state.

Therefore, in this specific case, BFS is considered the best algorithm. However, if memory constraints were a significant concern or if the state space were much larger, DFS or DLS might be more practical despite their lack of optimality.

Top of Form

Bottom of Form

**Post Lab Objective questions**

1. **Which search algorithm imposes a fixed depth limit on nodes?**
   1. Depth-limited search
   2. Depth-first search
   3. Iterative Deepening search
   4. Only (a) and (b)
   5. Only (a), (b) and (c).

**Answer: a**

1. **Optimality of BFS is**
   1. When all step costs are equal
   2. When all step costs are unequal
   3. When there is less number of nodes
   4. Both a & c

**Answer: a**

**Post Lab Subjective Questions:**

1. **Mention the criteria for the evaluation of search Algorithm.**Criteria for the evaluation of search algorithms:

* Completeness: Does the algorithm guarantee finding a solution if one exists?
* Optimality: Does the algorithm find the optimal solution, i.e., the one with the lowest cost?
* Time complexity: How long does the algorithm take to find a solution?
* Space complexity: How much memory does the algorithm require?
* Admissibility: Does the algorithm always underestimate the cost to reach the goal?
* Heuristic accuracy: How accurate are the heuristic estimates used in informed search algorithms?
* Any domain-specific considerations: Certain problems may have additional criteria specific to their characteristics.

1. **State the properties of BFS,DFS, DLS and IDS**

Properties of BFS, DFS, DLS, and IDS:

* **Breadth-First Search (BFS)**:
  + Completeness: Guaranteed to find a solution if one exists.
  + Optimality: Finds the optimal solution when step costs are equal.
  + Time complexity: �(��)*O*(*bd*), where �*b* is the branching factor and �*d* is the depth of the shallowest goal node.
  + Space complexity: �(��)*O*(*bd*) due to storing all nodes at each level.
* **Depth-First Search (DFS)**:
  + Completeness: Not guaranteed to find a solution; can get stuck in infinite loops.
  + Optimality: Not guaranteed to find the optimal solution.
  + Time complexity: �(��)*O*(*bm*), where �*b* is the branching factor and �*m* is the maximum depth of the search tree.
  + Space complexity: �(��)*O*(*bm*) due to storing a single path from the root to a leaf node.
* **Depth-Limited Search (DLS)**:
  + Similar to DFS but with a depth limit imposed.
  + Completeness: Depends on the depth limit; may not find a solution if the limit is too shallow.
  + Optimality: Not guaranteed to find the optimal solution.
* **Iterative Deepening Search (IDS)**:
  + A variant of DFS with incrementally increasing depth limits.
  + Combines the benefits of BFS and DFS.
  + Completeness: Guaranteed to find a solution if one exists.
  + Optimality: Optimal when step costs are equal.
  + Time complexity: �(��)*O*(*bd*), similar to BFS, but with less memory overhead.

1. **Explain why BFS is worst approach when the branching factor and solution depth in state-space tree is large (value =10 or more)**

Explanation of why BFS is the worst approach when the branching factor and solution depth in the state-space tree are large:

* BFS explores all nodes at each depth level before moving to the next level.
* With a large branching factor (�*b*) and solution depth (�*d*), the number of nodes at each level can grow exponentially.
* BFS needs to store all these nodes in memory, leading to a significant memory overhead.
* The memory consumption of BFS becomes prohibitively large as the branching factor and solution depth increase.
* Therefore, BFS becomes impractical and inefficient for large state-space trees with high branching factors and depths.

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No.**

|  |
| --- |
| **Title:** Implementation of Informed search algorithm( GBFS/A\*) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** Comparison and analysis of informed search algorithms

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russell and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**](http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf)
4. [**http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**](http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html)
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf** [**http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf**](http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf)
6. [**http://wiki.answers.com/Q/Informed\_search\_techniques\_and\_uninformed\_search\_techniques**](http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:** Problem solving, state-space trees, problem formulation, goal based agent architecture

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

The AI researchers have come up many algorithms those operate on state space tree to give the result. Goal based agent architectures solve problems through searching or planning. Depending on availability of more information other than the problem statement decides if the solution can be obtained with uninformed search or informed search.

Its fact that not all search algorithms end up in giving the optimal solution. So, it states the need to have a better and methodological approach which guarantees optimal solution.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Heuristic, Informed search, greedy best first search, A\* search

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Informed searching techniques**

* Greedy best first search
* A\*

**Chosen Problem statement: Traveling Salesman**

**State-space tree :**

**A diagram of a tree

Description automatically generated**

**Code:**

*import* numpy *as* np

*import* heapq

class TSP:

def \_\_init\_\_(self, num\_cities, distances):

*self*.num\_cities = num\_cities

*self*.distances = distances

def greedy\_best\_first\_search(self):

visited = [0] *# Start from city 0*

total\_distance = 0

*while* len(visited) < *self*.num\_cities:

current\_city = visited[-1]

min\_distance = float('inf')

next\_city = None

*for* city *in* range(*self*.num\_cities):

*if* city not in visited:

*if* *self*.distances[current\_city][city] < min\_distance:

min\_distance = *self*.distances[current\_city][city]

next\_city = city

visited.append(next\_city)

total\_distance += min\_distance

total\_distance += *self*.distances[visited[-1]][visited[0]]

visited.append(visited[0])

*return* visited, total\_distance

def a\_star(self):

start\_node = (0, [0], 0)

pq = [start\_node]

heapq.heapify(pq)

*while* pq:

total\_distance, visited, current\_city = heapq.heappop(pq)

*if* len(visited) == *self*.num\_cities:

total\_distance += *self*.distances[visited[-1]][visited[0]]

visited.append(visited[0])

*return* visited, total\_distance

*for* next\_city *in* range(*self*.num\_cities):

*if* next\_city not in visited:

next\_visited = visited + [next\_city]

next\_distance = total\_distance + *self*.distances[current\_city][next\_city] + *self*.distances[next\_city][0]

heapq.heappush(pq, (next\_distance, next\_visited, next\_city))

num\_cities = 4

distances = np.array([

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]

])

tsp = TSP(num\_cities, distances)

*# Greedy Best-First Search*

route\_greedy, distance\_greedy = tsp.greedy\_best\_first\_search()

print("Greedy Best-First Search:")

print("Route:", route\_greedy)

print("Total Distance:", distance\_greedy)

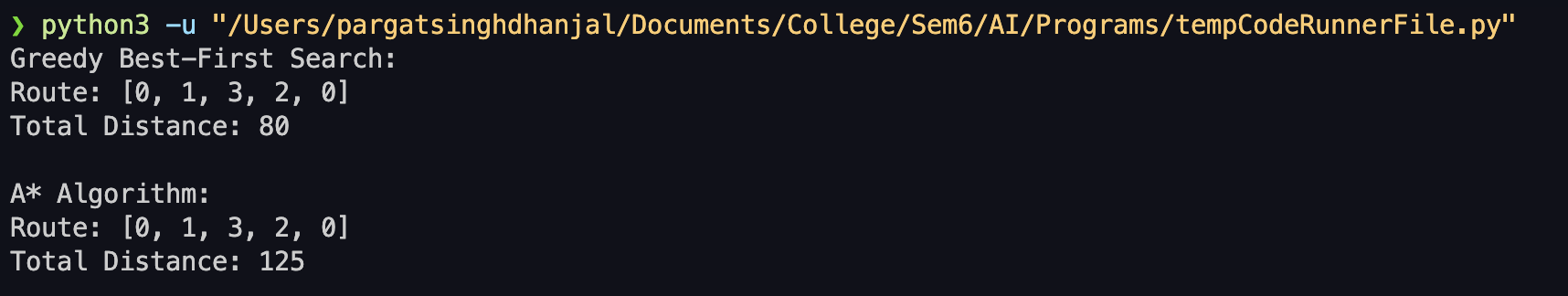
*# A\* Algorithm*

route\_a\_star, distance\_a\_star = tsp.a\_star()

print("\nA\* Algorithm:")

print("Route:", route\_a\_star)

print("Total Distance:", distance\_a\_star)

****

**Comparison of performance of Greedy and A\* Algorithm:**

A screenshot of a white sheet with black text

Description automatically generated

**Properties of A\* algorithm:**

1. **Completeness**: A\* finds a solution if one exists in a finite search space.
2. **Optimality**: A\* guarantees the shortest path if the heuristic is admissible and consistent.
3. **Memory Efficiency**: A\* uses a priority queue to explore promising paths first.
4. **Time Complexity**: A\* depends on heuristic quality but typically behaves like *O*(*bd*).
5. **Heuristic Function**: Guides search by estimating the cost to reach the goal.
6. **Adaptability**: Customizable for different problem domains.
7. **Node Expansion Order**: Expands nodes with lowest total cost (f-value).
8. **Potential Pitfalls**: Inefficiency or failure if heuristic isn't admissible or consistent, and high memory usage in large search spaces.

**Post lab Objective questions**

1. **A heuristic is a way of trying**
   1. To discover something or an idea embedded in a program
   2. To search and measure how far a node in a search tree seems to be from a goal
   3. To compare two nodes in a search tree to see if one is better than the other
   4. Only (a) and (b)
   5. Only (a), (b) and (c).

**Answer: e**

1. **A\* algorithm is based on** 
   1. Breadth-First-Search
   2. Depth-First –Search
   3. Best-First-Search
   4. Hill climbing.
   5. Bulkworld Problem.

**Answer: c**

1. **What is a heuristic function?** 
   1. A function to solve mathematical problems
   2. A function which takes parameters of type string and returns an integer value
   3. A function whose return type is nothing
   4. A function which returns an object
   5. A function that maps from problem state descriptions to measures of desirability.

**Answer: e**

**Post Lab Subjective Questions:**

1. **How best-first-search algorithm supports heuristic evaluation function?**

Best-first search algorithm supports heuristic evaluation function by using it to determine the "best" node to expand next. Instead of exploring all possible paths, it selects the node that is most likely to lead to the goal based on the heuristic evaluation function. This function provides an estimate of the cost from the current node to the goal, guiding the search towards the most promising paths.

1. **Find a good heuristic function for the following:**

**a. Monkey and Banana problem**

For the Monkey and Banana problem, a good heuristic function could be the Manhattan distance between the monkey and the banana. This heuristic estimates the number of moves required for the monkey to reach the banana by only considering the straight-line distance in each direction, ignoring any obstacles or walls.

**b. Traveling Salesman problem**

For the Traveling Salesman problem, a good heuristic function could be the minimum spanning tree (MST) heuristic. This heuristic calculates the minimum spanning tree of the graph representing the cities and then adds the minimum edge weight not in the MST to the total cost. Although not always optimal, it provides a good estimate of the shortest tour length.

1. **Define the heuristic search. Discuss benefits and shortcomings.**

Heuristic search is a search algorithm that uses a heuristic evaluation function to guide the search towards the most promising paths. It estimates the cost from the current state to the goal state and selects the next state to explore based on this estimation.

Benefits:

* Heuristic search can significantly reduce the search space by focusing on promising paths, leading to faster search times.
* It can be applied to large problem spaces where exhaustive search is not feasible.
* Heuristic search algorithms are often relatively simple to implement.

Shortcomings:

* The quality of the solution depends heavily on the quality of the heuristic function. A poorly designed heuristic can lead to suboptimal solutions or even failure to find a solution.
* Heuristic search algorithms are not guaranteed to find the optimal solution, as they may get stuck in local optima.
* It may be challenging to design a heuristic function that accurately estimates the true cost to reach the goal, especially in complex problem domains.

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 6**

|  |
| --- |
| **Title:** Implementation of Alpha Beta Pruning. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Implementation of Alpha-Beta Pruning algorithm

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf**](http://www.cs.sfu.ca/CourseCentral/310/oschulte/mychapter5.pdf)
4. [**http://cs.lmu.edu/~ray/notes/asearch/**](http://cs.lmu.edu/~ray/notes/asearch/)
5. **www.cs.cornell.edu/courses/cs4700/2011fa/.../06\_adversarialsearch.pdf**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pre Lab/ Prior Concepts:** Two/Multi player Games and rules, state-space tree, searching algorithms and their analysis properties

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile: -** The game playing has been integral part of human life. The multiplayer games are competitive environment in which everyone tries to gain more points for himself and wishes the opponent to gain minimum.

The game can be represented in form of a state space tree and one can follow the path from root to some goal node, for either of the player.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Adversarial search, minmax algorithm, minmax pruning,

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Adversarial Search:-**

Adversarial search is a fundamental technique in artificial intelligence used for determining the best move or strategy in a two-player game where both players have opposing goals. It involves exploring the game tree to ascertain the optimal move for one player while considering the potential counter-moves of the opponent.

In adversarial search, each possible move is evaluated based on a heuristic evaluation function, which estimates the utility of the move for the player. The algorithm selects the move that leads to the most favorable outcome for the player, assuming the opponent will also make their best moves.

The minimax algorithm is one of the most commonly used techniques in adversarial search. It operates by recursively traversing the game tree, evaluating each node by alternating between maximizing the player's score and minimizing the opponent's score. This process continues until a terminal state, where the game ends, is reached.

Another crucial algorithm in adversarial search is alpha-beta pruning. It is an enhancement of the minimax algorithm designed to reduce the number of nodes explored in the game tree. Alpha-beta pruning eliminates irrelevant nodes by maintaining bounds on the possible scores that can be achieved. This pruning process significantly reduces the computational effort required to find the optimal move.

**Alpha-beta pruning algorithm:**

Alpha-beta pruning is an optimization technique used in adversarial search algorithms, such as minimax, to reduce the number of nodes that need to be evaluated in the game tree. The basic idea of alpha-beta pruning is to eliminate nodes in the game tree that will never be reached because there is no need to evaluate them.

The alpha-beta pruning algorithm maintains two values, alpha and beta, which represent the best values found so far for the maximizing player and the minimizing player, respectively. Initially, alpha is set to negative infinity, and beta is set to positive infinity. As the search progresses, these values are updated based on the best possible scores found so far.

When the algorithm explores a node in the game tree, it evaluates the node and updates the alpha and beta values accordingly. If the alpha value becomes greater than or equal to the beta value, then the algorithm can prune the subtree rooted at that node, as it will never be reached in the game because the opponent would never make such a move.

By eliminating these irrelevant nodes, the alpha-beta pruning algorithm reduces the number of nodes that need to be evaluated, resulting in faster search times and more efficient game- playing strategies.

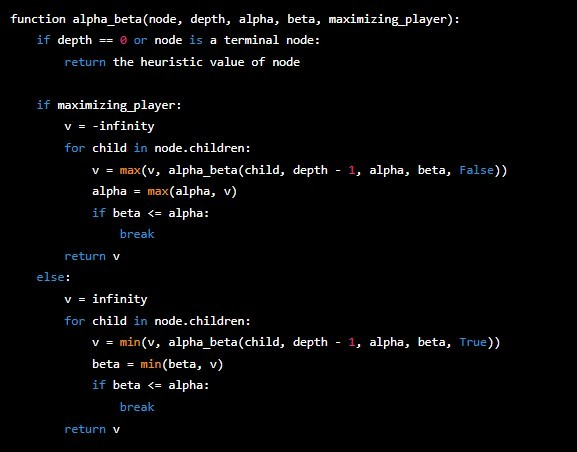
**Chosen Problem:**

Alpha-beta pruning is a technique used in game tree search algorithms to improve the efficiency of the search. It is particularly useful in games like Tic-Tac-Toe where the game tree is relatively small and can be fully searched.

In Tic-Tac-Toe, the game tree can be represented as a tree of all possible moves from the initial state. Each node in the tree represents a game state, and the edges represent possible moves. The goal of the algorithm is to find the best move from the current state, which is the move that leads to the highest score for the current player.

To perform alpha-beta pruning in Tic-Tac-Toe, the algorithm starts at the root node and evaluates all possible moves. It then recursively evaluates the resulting states, keeping track of alpha and beta values along the way. If the current player is trying to maximize their score, the algorithm updates alpha with the maximum score seen so far, and if the opponent is trying to minimize the score, it updates beta with the minimum score seen so far.

If the algorithm encounters a node where the beta value is less than or equal to the alpha value, it knows that the opponent can force a score that is worse than the current alpha value, so it can prune the rest of the subtree rooted at that node. This reduces the number of nodes that need to be evaluated, which improves the efficiency of the algorithm.

The pseudocode for the alpha-beta pruning algorithm in Tic-Tac-Toe would look something like this:

**Solution tree for chosen Problem:**

A diagram of a game

Description automatically generated

**Implementation:**

*import* copy

EMPTY\_SPACE = ' '

AI\_MARKER = 'O'

PLAYER\_MARKER = 'X'

WIN = 1

DRAW = 0

LOSS = -1

START\_DEPTH = 0

winning\_states = [

[(0, 0), (0, 1), (0, 2)],

[(1, 0), (1, 1), (1, 2)],

[(2, 0), (2, 1), (2, 2)],

[(0, 0), (1, 0), (2, 0)],

[(0, 1), (1, 1), (2, 1)],

[(0, 2), (1, 2), (2, 2)],

[(0, 0), (1, 1), (2, 2)],

[(2, 0), (1, 1), (0, 2)]

]

def print\_game\_state(*state*):

*if* *state* == WIN:

print("AI wins!")

*elif* *state* == DRAW:

print("It's a draw!")

*elif* *state* == LOSS:

print("Player wins!")

def print\_board(*board*):

print()

*for* row *in* *board*:

print(row[0], '|', row[1], '|', row[2])

print("----------")

print()

def get\_legal\_moves(*board*):

legal\_moves = []

*for* i *in* range(3):

*for* j *in* range(3):

*if* *board*[i][j] == EMPTY\_SPACE:

legal\_moves.append((i, j))

*return* legal\_moves

def position\_occupied(*board*, *pos*):

*return* *board*[*pos*[0]][*pos*[1]] != EMPTY\_SPACE

def get\_occupied\_positions(*board*, *marker*):

occupied\_positions = []

*for* i *in* range(3):

*for* j *in* range(3):

*if* *marker* == *board*[i][j]:

occupied\_positions.append((i, j))

*return* occupied\_positions

def board\_is\_full(*board*):

*for* row *in* *board*:

*if* EMPTY\_SPACE in row:

*return* False

*return* True

def game\_is\_won(*occupied\_positions*):

*for* win\_state *in* winning\_states:

*if* all(pos *in* *occupied\_positions* *for* pos *in* win\_state):

*return* True

*return* False

def get\_opponent\_marker(*marker*):

*return* PLAYER\_MARKER *if* *marker* == AI\_MARKER *else* AI\_MARKER

def get\_board\_state(*board*, *marker*):

occupied\_positions = get\_occupied\_positions(*board*, *marker*)

opponent\_marker = get\_opponent\_marker(*marker*)

*if* game\_is\_won(occupied\_positions):

*return* WIN

occupied\_positions = get\_occupied\_positions(*board*, opponent\_marker)

*if* game\_is\_won(occupied\_positions):

*return* LOSS

*if* board\_is\_full(*board*):

*return* DRAW

*return* DRAW

def minimax\_optimization(*board*, *marker*, *depth*, *alpha*, *beta*):

best\_move = (-1, -1)

best\_score = LOSS *if* *marker* == AI\_MARKER *else* WIN

*if* board\_is\_full(*board*) or DRAW != get\_board\_state(*board*, AI\_MARKER):

best\_score = get\_board\_state(*board*, AI\_MARKER)

*return* best\_score, best\_move

legal\_moves = get\_legal\_moves(*board*)

*for* curr\_move *in* legal\_moves:

*board*[curr\_move[0]][curr\_move[1]] = *marker*

*if* *marker* == AI\_MARKER:

score, \_ = minimax\_optimization(copy.deepcopy(*board*), PLAYER\_MARKER, *depth* + 1, *alpha*, *beta*)

best\_score = max(best\_score, score - *depth*)

*if* best\_score > *alpha*:

*alpha* = best\_score

best\_move = curr\_move

*if* *beta* <= *alpha*:

*break*

*else*:

score, \_ = minimax\_optimization(copy.deepcopy(*board*), AI\_MARKER, *depth* + 1, *alpha*, *beta*)

best\_score = min(best\_score, score + *depth*)

*if* best\_score < *beta*:

*beta* = best\_score

best\_move = curr\_move

*if* *beta* <= *alpha*:

*break*

*board*[curr\_move[0]][curr\_move[1]] = EMPTY\_SPACE

*return* best\_score, best\_move

def game\_is\_done(*board*):

*return* board\_is\_full(*board*) or DRAW != get\_board\_state(*board*, AI\_MARKER)

def main():

board = [[EMPTY\_SPACE] \* 3 *for* \_ *in* range(3)]

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\tTic Tac Toe AI\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print("Player = X\t AI Computer = O\n")

print\_board(board)

*while* not game\_is\_done(board):

row = int(input("Row (0-2): "))

col = int(input("Column (0-2): "))

print()

*if* row < 0 or row > 2 or col < 0 or col > 2:

print("Invalid input. Row and column should be between 0 and 2.")

*continue*

*if* position\_occupied(board, (row, col)):

print(f"The position ({row}, {col}) is occupied. Try another one...\n")

*continue*

board[row][col] = PLAYER\_MARKER

*if* game\_is\_done(board):

print("Game Over")

print\_board(board)

*break*

best\_score, best\_move = minimax\_optimization(copy.deepcopy(board), AI\_MARKER, START\_DEPTH, -float('inf'), float('inf'))

board[best\_move[0]][best\_move[1]] = AI\_MARKER

print\_board(board)

result = get\_board\_state(board, AI\_MARKER)

print\_game\_state(result)

*if* \_\_name\_\_ == "\_\_main\_\_":

main()

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of numbers and symbols

Description automatically generated**

**Post Lab objective Questions:**

1. **Which search is equal to minmax search but eliminates the branches that can’t influence the final decision?**
   1. Breadth-first search
   2. Depth first search
   3. Alpha-beta pruning
   4. None of the above

**Answer:**

1. **Which values are independent in minmax search alogirthm?**
   1. Pruned leaves x and y
   2. Every states are dependant
   3. Root is independent
   4. None of the above

**Answer:**

**Post Lab Subjective Questions:**

* + - 1. **Explain the concept of adversarial search**

Adversarial search is a type of search algorithm used in artificial intelligence that is specifically designed to find the optimal decision or strategy for a player in a competitive game or scenario. In adversarial search, the algorithm attempts to evaluate the possible outcomes of the game by taking into account the moves of both players and predicting their future moves based on the current state of the game.

The basic idea behind adversarial search is to model the game as a tree, where each node represents a possible game state and each edge represents a possible move by a player. The algorithm then uses various search techniques, such as the minmax algorithm, to traverse the tree and evaluate the potential outcomes of the game.

One important aspect of adversarial search is that it assumes that the opposing player is trying to win the game as well, and is making moves that are intended to block or interfere with the other player's strategy. Therefore, the algorithm must be able to anticipate and account for these moves in its evaluation of the game state.

Adversarial search is commonly used in a variety of games, such as chess, poker, and Go, as well as in other competitive scenarios, such as military strategy and cybersecurity. The goal of adversarial search is to find the best possible strategy or decision that a player can make in order to maximize their chances of winning the game, while also taking into account the actions of their opponent.

Adversarial search is a game-playing technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search. Here, game-playing means discussing those games where human intelligence and logic factor is used, excluding other factors such as luck factor. Tic-tac-toe, chess, checkers, etc., are such type of games where no luck factor works, only mind works.

Mathematically, this search is based on the concept of ‘Game Theory.’ According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.

* + - 1. **Explain how alpha-beta pruning improves memory efficiency of algorithm**

Alpha-beta pruning is a search algorithm used in adversarial search to improve the memory efficiency of the search algorithm. The main idea behind alpha-beta pruning is to reduce the number of nodes that are evaluated during the search process by eliminating those nodes that are guaranteed to lead to a worse outcome.

During the search process, the alpha-beta algorithm maintains two values, alpha and beta, which represent the bounds of the best score found so far for the maximizing player and the minimizing player, respectively. As the search progresses, the algorithm compares the current score of each node with the alpha and beta values, and if the score falls outside the bounds, the algorithm eliminates the node and all its descendants.

This pruning process reduces the number of nodes that need to be evaluated, which improves the memory efficiency of the algorithm. By eliminating nodes that are guaranteed to lead to a worse outcome, the algorithm is able to focus on the more promising nodes and avoid wasting resources on nodes that will not impact the final decision.

Overall, alpha-beta pruning is an effective technique for improving the memory efficiency of search algorithms in adversarial search problems, and is widely used in game playing applications such as chess and checkers.

* + - 1. **Explain how a game of chess may benefit from min-max and alpha-beta pruning algorithms.**

The game of chess is a classic example of a competitive, two-player game that can be solved using adversarial search algorithms such as the min-max and alpha-beta pruning algorithms. Here are a few ways in which these algorithms can benefit the game of chess:

Efficient Search: Chess has a very large search space, with a typical game consisting of around 35 possible moves at each turn. The min-max and alpha-beta pruning algorithms are designed to efficiently search through large decision trees, and can quickly eliminate branches that are unlikely to lead to a good outcome. This allows the algorithm to focus on the most promising moves, and can greatly reduce the number of nodes that need to be evaluated.

Evaluation Function: In order to search through the decision tree, the min-max and alpha- beta pruning algorithms require an evaluation function that assigns a score to each possible game state. In chess, an evaluation function might take into account factors such as the number of pieces on the board, the relative strength of each player's position, and the potential for future moves. By using an evaluation function, the algorithm can quickly eliminate moves that are unlikely to lead to a good outcome, and focus on moves that are more promising.

Pruning: Alpha-beta pruning can be particularly effective in reducing the search space for chess. Because chess is a zero-sum game, where one player's gain is the other player's loss, it is possible to prune large sections of the decision tree by eliminating moves that are guaranteed to be worse than other moves that have already been evaluated. This can greatly reduce the number of nodes that need to be evaluated, and can lead to significant improvements in the efficiency of the algorithm.

Overall, the min-max and alpha-beta pruning algorithms can be very effective in solving the game of chess, and have been used to develop some of the strongest chess-playing computer programs in the world. These algorithms can help players to quickly evaluate potential moves and develop winning strategies, and can help to push the boundaries of what is possible in this classic game.

**Conclusion:** Thus, we have understood and performed the Implementation of Alpha Beta Pruning.

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No.**

|  |
| --- |
| **Title:** Implementation of Constraint Satisfaction concepts |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Implementation of Local search algorithm

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms). |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pre Lab/ Prior Concepts:** Informed, uninformed search,Local search

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:** Constraint Satisfaction, CSP with backtracking

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Definition:-** (Define CSP)

**The sequence in which variable-constraint assignments are considered by CSP algorithms to improve the backtracking efficiency:-**

**Problem chosen: Cryptarithmetic**

**DAYS + TOO = SHORT**

**Step by step solution to the problem:**

**Explanation:**  
**F R E E**  
**+ B I R D**  
**———–**  
**C A G E**

**Step 1:**

Start with the rightmost column, E + D = E.

This implies D = 0

**Step 2:**

Moving to the left, E + R + carry (from Step 1) = G.

Since G can’t be 0, the minimum value for E + R + carry is 10. So, E + R + carry = 10 + G.

**Step 3:**

Now, we have R + I + carry (from Step 2) = A.

This implies R + I + carry – A = 0

**Step 4:**

Moving further left, F + B + carry (from Step 3) = C.

This implies F + B + carry – C = 0.

**Step 5:**

Finally, we have a carry in the leftmost column.

Now, we want to find the values of F, R, B, I, C, A such that A + B + C + D + E + F + G equals 22.

Let’s solve through an example:

Suppose we set F = 1 and R = 9. Then, from Step 2:

**E + 9 + carry = 10 + G**  
**E + G + carry = 1**

**Since no two letters can have the same value,** we can’t have E = G = 0. So, the carry must be 1.

Now, from Step 3:

**9 + I + 1 – A = 0**  
**I + 10 – A = 0**  
**I – A = -10**

This implies A must be 0 (to avoid negative values).

Now, from Step 4:

1 + B + 1 – C = 0  
B + 2 – C = 0  
B – C = -2

This implies C must be greater than B, and since B can’t be 0 (as D is 0), C must be at least 3. Let’s choose C = 3 and B = 1.

Now, we’ve found values for F, R, B, I, C, A such that A + B + C + D + E + F + G = 22:

F = 1, R = 9, B = 1, I = 0, C = 3, A = 0

Now, let’s calculate E and G:

**E + 9 + 1 = 10 + G**  
**E + 10 = 10 + G**

This implies E = G.

So, E = G = 5.

Now, we add up A + B + C + D + E + F + G:

0 + 1 + 3 + 0 + 5 + 1 + 5 = 15

Therefore, A + B + C + D + E + F + G = 22

**Code:**

*import* itertools

def get\_value(word, substitution):

s = 0

factor = 1

*for* letter *in* reversed(word):

s += factor \* substitution[letter]

factor \*= 10

*return* s

def solve(equation):

left, right = equation.lower().replace(' ', '').split('=')

left = left.split('+')

letters = set(right)

*for* word *in* left:

*for* letter *in* word:

letters.add(letter)

letters = list(letters)

digits = range(10)

*for* perm *in* itertools.permutations(digits, len(letters)):

sol = dict(zip(letters, perm))

*if* sum(get\_value(word, sol) *for* word *in* left) == get\_value(right, sol):

values\_left = [str(get\_value(word, sol)) *for* word *in* left]

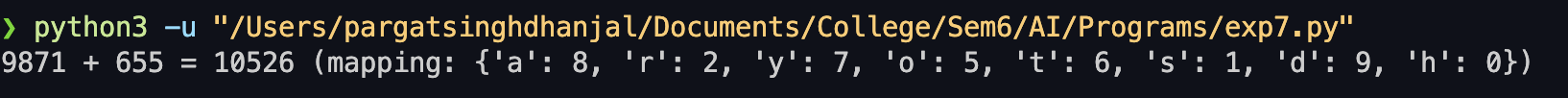
value\_right = get\_value(right, sol)

result = ' + '.join(values\_left) + " = {} (mapping: {})".format(value\_right, sol)

print(result)

*if* \_\_name\_\_ == '\_\_main\_\_':

solve('D A Y S + T O O=S H O R T ')

****

**Post Lab objective Questions:**

**To overcome the need to backtrack in constraint satisfaction problem can be eliminated by \_\_\_\_\_\_\_\_\_\_\_\_**

**a) Forward Searching**

**b) Constraint Propagation**

**c) Backtrack after a forward search**

**d) Omitting the constraints and focusing only on goals**

**Consider a problem of preparing a schedule for a class of student. What type of problem is this?**

**a) Search Problem**

**b) Backtrack Problem**

**c) CSP**

**d) Planning Problem**

**Q1. How do you solve a CSP Problem?**

To solve a Constraint Satisfaction Problem (CSP), you typically use techniques such as:

* **Backtracking:** Enumerate possible assignments of values to variables and backtrack when a constraint violation is encountered.
* **Constraint Propagation:** Use local constraints to reduce the domain of variables, thus making it easier to find a solution.
* **Forward Checking:** After making an assignment, check the remaining domains of variables to see if any are reduced to empty, which would indicate a dead-end.
* **Arc Consistency:** Ensure that for every variable, there is a consistent assignment for every other variable with which it shares a constraint.
* **Heuristic Methods:** Use heuristics to guide the search process, such as variable ordering or value ordering heuristics.

**Q2. Explain CSP with an example.**

Let's consider a classic example of a CSP: the map coloring problem.

**Problem Statement:** Given a map of regions (such as countries) where adjacent regions cannot have the same color, assign a color to each region such that no adjacent regions share the same color.

**Variables:** Each region on the map is a variable.

**Domains:** The domain of each variable is the set of available colors.

**Constraints:** The constraint is that no two adjacent regions can have the same color.

**Example:**

In this example, suppose we have a map with four regions: A, B, C, and D. The constraints are that adjacent regions cannot have the same color. Let's say we have three colors available: red, blue, and green.

We could start by assigning colors to variables and applying constraint propagation and backtracking to find a valid solution. For instance, we might start by assigning red to region A, then check the adjacent regions to see which colors are available for them, and so on until a valid assignment for all regions is found. If at any point a region cannot be assigned a color without violating the constraints, we backtrack and try a different color for a previous region.

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 8**

|  |
| --- |
| **Title:** IEEE Case study on learning/planning techniques |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Case study on AI System (learning/planning techniques) based paper published in IEEE/ACM/Springer

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO4** | Analyse applications of AI and understand planning & learning processes in advanced AI applications |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

A learning agent is **a tool in AI that is capable of learning from its experiences**. It starts with some basic knowledge and is then able to act and adapt autonomously, through learning, to improve its own performance.

Artificial intelligence is an important technology in the future. Whether it is intelligent robots, self-driving cars, or smart cities, they will all use different aspects of artificial intelligence!!! But Planning is very important to make any such AI project.

Even Planning is an important part of Artificial Intelligence which deals with the tasks and domains of a particular problem. Planning is considered the logical side of acting.

Everything we humans do is with a definite goal in mind, and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.

**For example**, Planning is required to reach a particular destination. It is necessary to find the best route in Planning, but the tasks to be done at a particular time and why they are done are also very important.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***Note: A minimum of two papers should be selected based on the AI application of learning or planning concepts and summarized as follows.***

**Title :** *Deep Reinforcement Q-Learning for Intelligent Traffic Signal Control with Partial Detection*

**Abstract of the paper :**

This paper introduces a deep reinforcement Q-learning model designed to optimize traffic signal control at an isolated intersection in environments where only a portion of vehicles are detected, thanks to the emergence of connected vehicles. Unlike most literature that assumes complete vehicle detection at intersections, this model works under the realistic scenario of partial detection. It incorporates a novel state representation for partially observable environments, a new reward function tailored for traffic signal control, and presents a detailed network architecture with tuned hyper-parameters.

**Proposed Architecture / System:**

The proposed architecture includes a detailed deep reinforcement Q-learning model with several innovative components:

* **Partial DTSE (Discrete Traffic State Encoding)**: A new state representation that captures the partially observable environment of the intersection, focusing on connected vehicles.
* **Total Squared Delay Reward Function**: A new reward function designed to minimize the total travel time through the intersection by emphasizing fairness and efficiency.
* **Convolutional Dueling DQN (Deep Q-Network)**: The network architecture, utilizing convolutional neural networks (CNN) for handling the state representations, is specifically tuned for the task of traffic signal control under partial vehicle detection.

**Results:**

The model was tested against traditional actuated traffic signal controllers like Max Pressure and SOTL in various simulation scenarios. It demonstrated superior performance, particularly in scenarios with more complex intersection configurations and varying traffic demands. Notably, the model showed robustness and adaptability to different levels of vehicle detection, with significant improvements in traffic flow and reduced delays, even at low detection rates of connected vehicles.

**Conclusion:**The study concludes that the deep reinforcement Q-learning model significantly outperforms existing actuated traffic signal controllers in environments with partial vehicle detection. It sets new performance thresholds for acceptable and optimal operations based on the rates of connected vehicle detection, advocating for the deployment of such models at intersections to improve traffic flow and reduce congestion. The paper also outlines several avenues for future research, including exploring probabilistic estimation methods for uncertain vehicle detections and extending the model to decentralized multi-agent systems for coordinated traffic light control across multiple intersections.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Title:**

LLM-ASSIST: Enhancing Closed-Loop Planning with Language-Based Reasoning

**Abstract:**

This study investigates the integration of Large Language Models (LLMs) like GPT4 and Llama2 with rule-based planning for autonomous driving to address the limitations of existing learning- and rule-based planners in handling diverse driving scenarios. A novel hybrid planner is proposed, which combines a conventional rule-based planner with an LLM-based planner. By leveraging the commonsense reasoning of LLMs, the approach navigates complex scenarios with well-reasoned outputs, achieving state-of-the-art performance on the nuPlan benchmark across various metrics. The code for this project will be made available online.

**Proposed Architecture / System:**

The architecture introduces a hybrid planning method that utilizes a state-of-the-art rule-based planner, PDM-Closed, for common scenarios, and an innovative LLM-based planner for high-uncertainty scenarios. The system dynamically engages the LLM-based planner based on the base planner's performance metrics. It involves two modes of LLM assistance: **LLM-ASSISTUNC**, which allows the LLM to directly generate a future trajectory for the ego vehicle, and **LLM-ASSISTPAR**, where the LLM suggests parameters for the base planner to safely navigate the scenario.

**Results:**

Extensive evaluations on the nuPlan benchmark demonstrated the superior performance of the LLM-assisted planner in both reactive and non-reactive settings, outperforming existing learning- and rule-based methods. The approach notably improves safety, reducing the number of dangerous driving scenarios by 11%. Ablation studies and qualitative analyses further underscore the effectiveness of leveraging LLMs for planning in complex driving environments.

**Conclusion:**

LLM-ASSIST represents a significant advancement in autonomous driving planning, showcasing the potential of combining traditional rule-based planners with the cognitive and reasoning capabilities of LLMs. The approach not only sets new benchmarks in performance but also offers insights into future research directions, emphasizing the need for further improvements in LLM grounding, multimodality, and processing speeds to fully harness their potential in real-time, safety-critical applications like autonomous driving.

**Batch: Roll No.:**

**Experiment / assignment / tutorial No. 9**

|  |
| --- |
| **Title:** Virtual Lab 1 |

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective: Self-Learning Experiment using Virtual Lab in NLP to introduce concepts like word analysis, word generation and building POS Tagger

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO4** | Analyse applications of AI and understand planning & learning processes in advanced AI applications |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pre Lab/ Prior Concepts:** Natural Language Processing

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory: Discuss/Add the Language Model, Text Classification in NLP.**

**Reference Link:** [**https://nlp-iiith.vlabs.ac.in/exp/word-analysis/**](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/)

**Batch: B1 Roll No.: 16010121045**

**Experiment / assignment / tutorial No. 10**

|  |
| --- |
| **Title:** Virtual Lab 2 |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Objective:** Understand the Virtual Lab Experiment for tagging Parts of Speech using Hidden Markov Model.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO4** | Analyse applications of AI and understand planning & learning processes in advanced AI applications |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russel and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pre Lab/ Prior Concepts:** Natural Language Processing

Hidden Markov Models (HMMs) are statistical models used to model sequences of observable events that are assumed to depend on unobservable, or "hidden," states. One common application of HMMs is in part-of-speech (POS) tagging, where each word in a sentence is assigned a grammatical category (e.g., noun, verb, adjective) based on the context of the sentence.

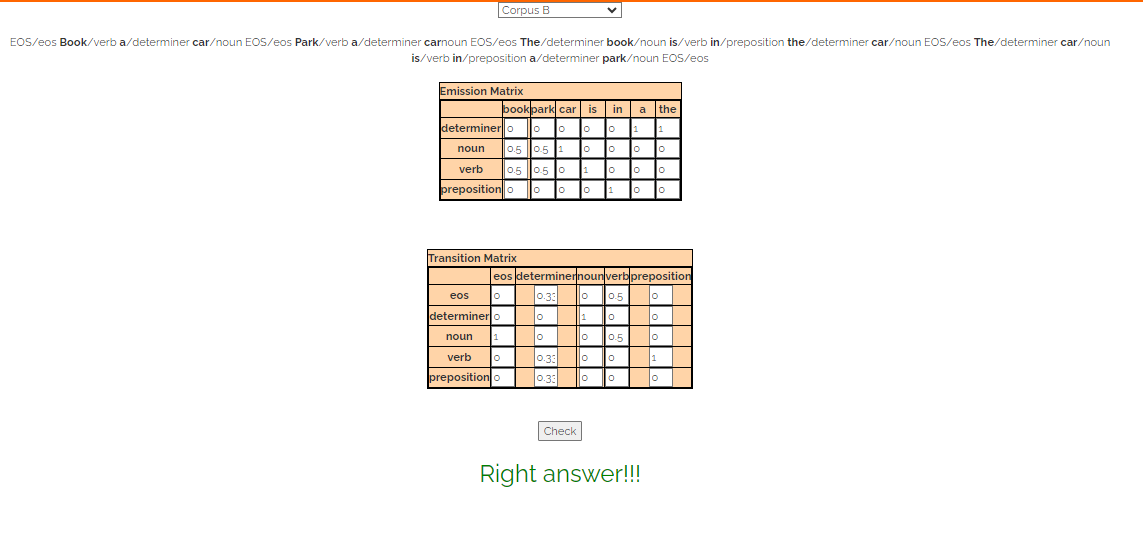
Here's a general procedure for POS tagging using a Hidden Markov Model:

1. **Define States**: The first step is to define the states of the model. In POS tagging, each state typically represents a part-of-speech tag (e.g., noun, verb, adjective).
2. **Define Observations**: Next, define the observations. These are the observable events associated with each state. In POS tagging, the observations are the words in the sentence.
3. **Transition Probabilities**: Calculate the transition probabilities between states. These probabilities represent the likelihood of transitioning from one POS tag to another. They are typically estimated from a labeled corpus of text data.
4. **Emission Probabilities**: Calculate the emission probabilities of observing each word given each POS tag. These probabilities represent the likelihood of observing a particular word given a certain POS tag. They are also estimated from a labeled corpus of text data.
5. **Initial State Probabilities**: Calculate the initial probabilities of starting in each state. These probabilities represent the likelihood of starting a sentence with each POS tag.
6. **Forward Algorithm**: Use the forward algorithm (or other methods like Viterbi algorithm) to compute the probability of a sequence of observations given the model. This involves calculating the probability of being in a particular state at each step of the sequence and the probability of observing each observation given the current state.
7. **Backward Algorithm (Optional)**: In some cases, the backward algorithm is used to calculate the probability of ending a sequence with a particular state. This can be useful for certain applications or for refining the tagging process.
8. **Decoding**: Once the probabilities are calculated, decode the sequence of POS tags that best explains the observed sequence of words. This is typically done using the Viterbi algorithm, which finds the most likely sequence of states given the observations.
9. **Tagging**: Finally, assign the most likely POS tag to each word in the sentence based on the decoding results.

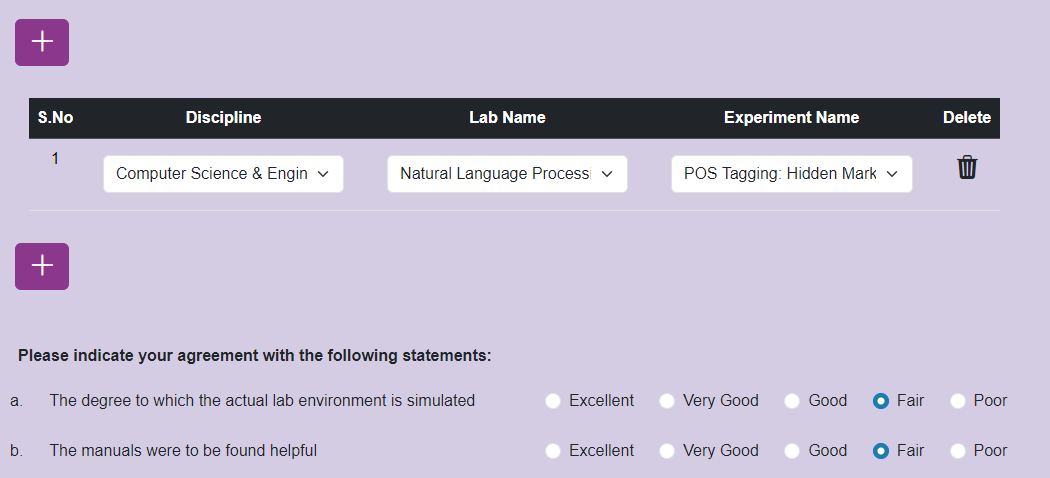
By following this procedure, a Hidden Markov Model can be trained and used for part-of-speech tagging, which is a fundamental task in natural language processing.

**Reference Link:** [**https://nlp-iiith.vlabs.ac.in/exp/markov-model/**](https://nlp-iiith.vlabs.ac.in/exp/markov-model/)

**Simulation Results:**



**Feedback:**

****

**Postlab Questions:**

1. **How is Hidden Markov Model different from Markov Model?**

A Markov model is a stochastic model that describes a sequence of possible events where the probability of each event depends only on the state attained in the previous event. These models are memoryless, meaning that the future state depends only on the current state and not on the sequence of events that preceded it.

A Hidden Markov Model (HMM) extends the concept of a Markov model by introducing hidden states, which are not directly observable. Instead, the model emits observable symbols or events based on the hidden states. The transitions between hidden states and the emissions of symbols from each state are governed by probabilities. In HMMs, the goal is to infer the sequence of hidden states given the observed sequence of symbols.

1. **What is the basic design for HMM for finding out POS?**

* **States**: In a POS tagging HMM, the states represent the different parts of speech (e.g., noun, verb, adjective).
* **Observations**: The observations are the words in the text.
* **Transition Probabilities**: These represent the probability of transitioning from one part of speech to another. For example, the probability of transitioning from a noun to a verb or from a verb to an adjective.
* **Emission Probabilities**: These represent the probability of observing a word given a particular part of speech. For example, the probability of observing the word "dog" given that the current part of speech is a noun.
* **Initial State Probabilities**: These represent the probability of starting with each part of speech at the beginning of a sentence.
* **Decoding Algorithm**: Use algorithms like the Viterbi algorithm to find the most likely sequence of POS tags given the observed sequence of words.

The design of a POS tagging HMM involves estimating these probabilities from a labeled corpus of text data, where each word is annotated with its corresponding part of speech. These probabilities are then used to infer the most likely sequence of POS tags for unseen text data.

Top of Form

Bottom of Form