**ARTIFICIAL INTELLIGENCE - FINAL PROJECT**

**CAR CHOOSER**

**by CHANDAN PARAMESWARAIAH**

**Net ID - cbp140230**

**Problem Description**

To implement a logical agent that will accept inputs from the user for their preferred choices in car attributes and hence provide best available car to them which matches all the choices as requested by the user. If a car matching requirements of the user is not available in the knowledge base then appropriate message is shown to the user.

**Example**

Preferred Choice of a car from the user

1) Car should have automatic A/C

2) It should have a petrol engine

3) It should have Sedan Body Style.

**Knowledge Base is represented in OWL**

1) ABC is a car of Type Audi

2) XYZ is a car of Type Chevrolet

3) All Audi cars have petrol engines

4) Chevrolet cars have both petrol engines and diesel engines.

5) Engines are either petrol or diesel.

6) Some cars of Chevrolet have Sedan Body Style.

**Inferred Facts from knowledge base**

Audi cars do not have diesel engines.

Some of the Chevrolet cars have petrol engines and some have diesel engines.

Based on the existing facts and inferred facts the logical agent may suggest one of Chevrolet Cars to user.

**Proposed Solution**

The act of selecting a car by logical agent involves satisfying all the constraints. This process of satisfying constraints is achieved by implementing A\* algorithm and Greedy Best First Search with the following details.

1) A\* - g(n) - estimated minimum cost of the node n from the starting node. In the current implementation this value is chosen as number of instances in the node satisfying particular constraint.

h(n) - heuristics chosen is remaining unsatisfied constraint. In this the approach nodes are generated from the initial state and put in the priority queue and they are processed based on their least f(n) value.

2) Greedy Best First Search - In this approach the variable is chosen in a particular order i.e each variable is given a weight, the variable with the least weight is chosen for expansion. That means the algorithm selects the most promising node which it considers as the nearest node to goal node.

The variables chosen in this problem according to the order is listed below.

1) Make - Car manufacturer company

2) Engine - Petrol or Diesel Engine

3) AC - Automatic AC or Manual AC

4) Color - Light Color or Dark Color

5) Body Style - Body style may be Sedan, Coupe or SUV.

For Example If a user gives Car make, AC, Bodystyle as his preference then this algorithm first expands node satisfying make constraint, then AC and then body style.

**Knowledge Base**

Knowledge Base consists of the following.

**Classes**

1) AirConditioner - Contains all types of AC

a) AutomaticAC - subclass of AirConditioner.

b) ManualAC - subclass of AirConditioner.

2) BodyStyle - Contains different types of Style Cars

a) Coupe - subclass of BodyStyle

b) Sedan - subclass of BodyStyle

c) SUV - subclass of BodyStyle

3) Car - Contains different types of Cars.

a) Audi - subclass of Car

b) Benz - subclass of Car

c) BMW - subclass of Car

d) Chevrolet - subclass of Car

e) Ford - subclass of Car

f) Honda - subclass of Car

4) Color - Contains Light Colors and Dark Colors

DarkColor - subclass of Color

LightColor - subclass of Color.

5) Engine - Contains two types of Engines.

PetrolEngine - subclass of Engine

Diesel Engine - subclass of Engine.

**Object Properties**

1) hasAutomaticAC - has domain as Car and Range as AutomaticAC class.

2) hasColor - has domain as Car and Range as Color class.

3) hasCoupeBodyStyle - has domain as Car and Range as Coupe class.

4) hasDieselEngine - has domain as Car and Range as DieselEngine class.

5) hasManualAC - has domain as Car and Range as ManualAC class.  
6) hasPetrolEngine - has domain as Car and Range as PetrolEngine class.

7) hasSedanBodyStyle - has domain as Car and Range as Sedan class.

8) hasSUVBodyStyle - has domain as Car and Range as SUV class.

**Data Properties**

1) acname - holds ac name

2) bodystylename - holds body style name

3) carname - holds car name

4) enginename - holds name of the engine

5) engineWeight - holds weight of the engine in KG.

6) partnumber - holds part number of the engine.

**Implementation Details**

**Programming Tools used**

**1) Protege**

It is a Stanford GUI tool used to represent Knowledge Base in RDF/OWL ontology. Version used is 5.0.0-beta-15.

**2) Eclipse**

IDE for developing source code .java files which implements Jena APIs**.**

**Queries**

**SPARQL -** Query used to retrieve facts from the knowledge base. SPARQL does not have inference engine, it just retrieves existing facts from the knowledge base.

**JENA** - API used to query OWL ontology. Unlike from SPARQL, JENA has inference engine which includes results also as part of inference including the query involved. Java uses JENA API to obtain facts from knowledge base.

**Examples & Results**

Consider user gives five properties of the car say Sedan Body Style, Car make Audi, Petrol Engine.

**1) Using A\***

g(n) - Number of instances generated for that node for a satisfying constraint(cost increases along the search path)

h(n) - Remaining unsatisfied constraint

AudiA1,AudiA2

AudiA3,AudiS3

[1,0,0]

AudiS3,MayBach\_S600

BMW501,BMW502,Ford,Audi S2

[0,1,0]

AudiA1,MercedezAMG,AudiA2,Cruze,Malibu,AudiA3,Ikon,AudiS3,CivicSport

MayBach\_S600.

[0,0,1]

AudiS3

[1,1,0]

AudiA1,Audi A2,AudiS2,AudiS3

[1,1,0]

AudiS3

[1,1,1]

f(n) = 4+2(Make) f(n) = 6+2(Body Style) f(n) = 10+2(Petrol Engine)

f(n) = 1+1(Make) f(n) = 5+1

Goal Node h(n) = 0. All constraints satisfied.

At first expansion, 3 nodes are generated with respect to 3 choices given by the user. Their f(n) values are computed are put in the priority queue. In the next step least f(n) is taken from the queue and its nodes are expanded. This step is continued till until a node from priority queue is taken such that all its constraints are satisfied.

Since the estimated heuristic is always less than the actual cost of reaching the goal node, heuristic chosen is admissible. In the above example total cost involved is 9 in reaching the goal node.

**2) Greedy Best First Search**

AudiS3,MayBach\_S600

BMW501,BMW502,Ford,Audi S2

[0,1,0]

AudiA1,MercedezAMG,AudiA2,Cruze,Malibu,AudiA3,Ikon,AudiS3,CivicSport

MayBach\_S600.

[0,0,1]

AudiS3

[1,1,0]

AudiA1,Audi A2,AudiS2,AudiS3

[1,1,0]

AudiS3

[1,1,1]

Consider the user has given Color Blue, Sedan, Diesel Engine as their preferred choices.

Accord,AudiS3

Cruze, Ikon, Malibu

[1,0,0](Color)

AudiS3,MayBach\_S600

BMW501,Malibu,Ford,Audi S2

[0,1,0](Body Style)

BMW501,BMW502,Malibu,Fiesta,

Accord

[0,0,1] (Diesel Engine)

Accord, Malibu

[1,1,0](Color)

Malibu, Malibu501

[1,1,0](Body Style)

Malibu

[1,1,1](Goal)

In the above example search algorithm selects the node with least order i.e. highest priority node to expand states into its successors. Since Engine has the least order among BodyStyle and Color, in the first level Engine node is expanded. Followed by Color Node and finally goal node is achieved and Body Style node remains unexpanded.

**Architectural Diagram**

SPARQL Engine (Inference)

Application

(.java)

JENA API

(Knowledge Base)

Protege

RDF/XML

Return RDF/XML

Search Algorithm

A\* or Greedy Best First Search

Application

(.java)

Goal (All Constraints Satisfied)

Search Space

**Summary Of Problems Encountered**

Sparql queries implementation for multilevel inference which involved nested queries was challenging where Input given from the user has to be inferred to the immediate level of facts in the knowledge base and from the inferred facts the next level of inference has to be done.

The query did yield the undesired output for multilevel inference, to overcome this issue one of the measures taken were changing the inference level of JENA API. And the other measure was to implement the nested query i.e. joining two or more queries to obtain the desired output.

**Pending Issues**

In A\* algorithm implementation f(n) should be increasing along the search path, If a child node has f(n) value less than its parent, then its parent nodes f(n) value is chosen according to path max equation.

In the search algorithm implemented f(n) value sometimes decreases for the child node and though its parent f(n) is considered, more suitable heuristics and cost can be chosen such that f(n) never decreased along its search path.

**Potential Improvements**

1) Knowledge Base is incomplete i.e. Entire domain knowledge is not included in Protege to represent problem domain. One measure for improvement is to include complete Knowledge Base and implement full fledge logical agent

2) GUI can be implemented to accept the user input.

3) One of the heuristic chosen uses greedy algorithm to find the solution. As always a greedy algorithm does not yield optimal solution, better heuristic can be chosen.