LBA: Hyderabad Restaurant Expert System

Raymundo Gonzalez, Eugene Chan, Frank Looi

CS152 Spring 2018

Introduction

Finding where to eat in Hyderabad is hard. There are thousands of restaurants with different cuisines, distance, price range and other information. An expert system is perfect to solve the problem of suggesting restaurants that can meet the preference of anyone looking for a restaurant to eat at.

Fig 1 -- Triumph after finding the perfect restaurant at Hyderabad to eat (Anandam)

Our expert system is built with Prolog for two reasons:

- Prolog uses predicate logic that can map relationships between objects and properties. This is handy when we are defining a restaurant based of its relationship with different types of cuisines, payment methods and other characteristics
- Prolog has a default inference engine that uses backward chaining. It is efficient because it prunes away parts of the search tree that are not relevant to the goal. Using Prolog's default inference engine, we do not need to build our own inference engine on Python

We implemented a Prolog backend to store our knowledge base and inference rules. Our Python front-end deals with the interaction with the user. Using Python for our front-end allows more user-friendly interaction with our inclusion of the graphic user inference.

Building the Restaurant Knowledge Base

We collected data from our favorite restaurants in Hyderabad, including Anandam, Mainland China, Burger King, KFC, Domino's Pizza, Karachi Cafe, Whiteboard Cafe, Firangi Bake, Little Italy, Mamagoto, Urban Asia, Hitec Nanking, Grill Box, Freshies Gourmet Salad, Mustang, India Bistro, Drunken Monkey, Wich Please, Punjabi Bytes, and Holi.

For each restaurant, we collected its cuisine type, accepted payment method, average price for two people, rating (on the scale of one to five) and distance from our residential hall (measured in km).

Prolog back-end: Askables and Rules[1]

Our askable is *pick_restaurant(X)*. It will return a list of variable(s) that unify with restaurants in the knowledge base that matches the values of the user input.

Our rules include:

- *pick_restaurant*: this rule selects a certain restaurant if the user's cuisine, payment method, price, rating and distance matches with the restaurant
- *memberOfCuisine*: check if *all* of the user's input for cuisine (a list of categories) is a member of each restaurant's cuisine (also a list of categories). For example, this outputs True for a restaurant that has Chinese *and* Seafood cuisines, according to the user input
- *memberOfPay*: similar to memberOfCuisine, but checking if *any* of the preferred payment method exists in the restaurant's payment method. For example, output True if a restaurant accepts cash *or* card.
- *smallerThan*: outputs True if a user input integer is smaller than another value, which can be used for Price, Rating and Distance criteria. For example, users can specify their maximum price, and the restaurant's average price needs to be smaller than this maximum price.

Python front-end: Graphic User Interface

Using the Tkinter Python library, we are able to implement a graphic user interface to ask the user his/her parameters for the restaurants he/she wants to go. Fig. 1 demonstrate the GUI implemented.

[1] **#ailogic** - We designed our knowledge base in a way that allows unification of a variable with a restaurant name, given the user criteria. After obtaining the user input on Python front-end, we assert them into our Prolog expert system to infer a suitable restaurant from the KB through backward-chaining. Our KB design is also elegant because we just need to add one more line for every new restaurant in our KB.

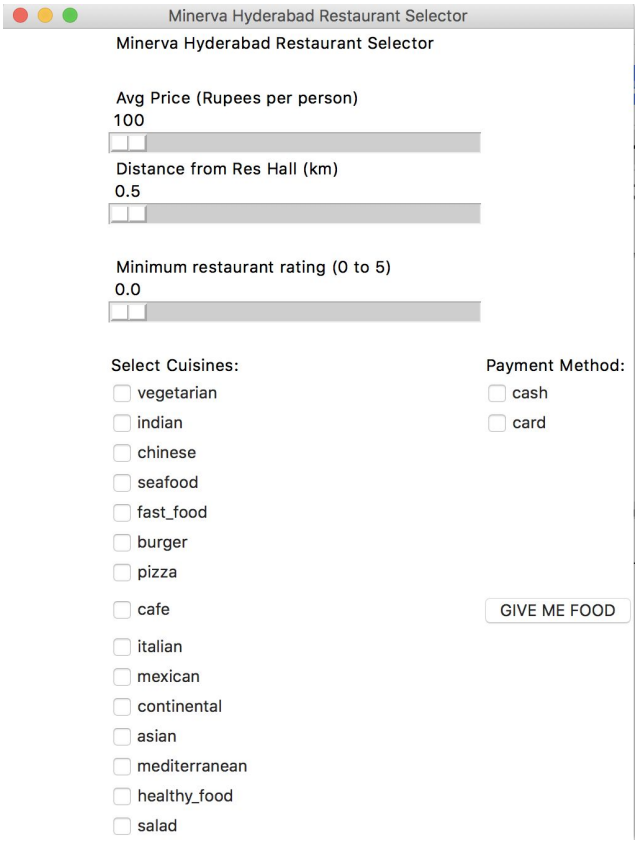

Fig 2 -- GUI for the restaurant expert system; User selects the maximum average price he/she is willing to pay, maximum distance willing to travel, minimum rating acceptable, multiple cuisines and payment method. Clicking "GIVE ME FOOD" will return the name of the restaurant that matches the parameters

Test Cases for the Expert System

	Test Case 1	Test Case 2	Test Case 3
Cuisine	Indian, Vegetarian	Italian	Fast food
Payment Method	Card	Cash OR Card	Cash OR Card
Budget	₹300 per person	₹800 per person	₹200 per person
Minimum Rating	3.0	3.6	3.5
Distance (Up to)	3 km	5 km	4 km
Recommendation	Anandam (Fig 1)	Firangi Bake, Little Italy, Mustang Terrace Lodge	KFC, Wich Please

References

- Expert system algorithm from class session 12.2
- SWI-Prolog documentation: http://www.swi-prolog.org/pldoc/index.html
- TkInter documentation: https://docs.python.org/2/library/tkinter.html

Appendix 1. Contribution of each group member to the assignment.

Group Member	Data Collection and Project Management	Front End Development	Back End Development	Product Improvement and Report
Raymundo	20%	70%	30%	20%
Eugene	20%	15%	50%	40%
Frank	60%	15%	20%	40%

The entire group was involved in deciding the architecture of the expert system, including structure of the knowledge base and degree of integration with Python. Even though each team member ended up focusing on a specific area of expertise, we discussed and debugged different codes along the way.

Raymundo primarily worked on the Graphical User Interface using the Python TkInter library. He also defined the general form of the KB so that we could unify the restaurants that satisfy user requests, while using just one line per restaurant. Eugene focused on improving the KB, so that we could handle multiple cuisine choices and payment methods in a single request, and Frank compiled data and managed the timeline and execution of this project.

Appendix 2. Python Code for the Expert System

```
import pyswip
from pyswip.easy import *
from Tkinter import *

KB = """

% Enter your KB below this line:

pick_restaurant(anandam) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[vegetarian,indian]), payment(Pay), memberOfPay(Pay , [card]), price(Price),
smallerThan(150, Price), rating(Rating), smallerThan(Rating,3.6), distance(Distance),
smallerThan(2.5, Distance).

pick_restaurant(mainland_china) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [chinese,
seafood]), payment(Pay), memberOfPay(Pay , [card,cash]), price(Price), smallerThan(400,
Price), rating(Rating), smallerThan(Rating,3.7), distance(Distance), smallerThan(1.6,
Distance).

pick_restaurant(burger_king) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [fast_food,
burger]), payment(Pay), memberOfPay(Pay , [cash,card]), price(Price), smallerThan(200,
Price), rating(Rating), smallerThan(Rating,3.6), distance(Distance), smallerThan(4.7,
Distance).
```

```
pick_restaurant(kfc) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [fast_food, burger]),
payment(Pay), memberOfPay(Pay , [cash]), price(Price), smallerThan(200, Price),
rating(Rating), smallerThan(Rating,3.7), distance(Distance), smallerThan(3.1, Distance).

pick_restaurant(dominos_pizza) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [fast_food,
pizza]), payment(Pay), memberOfPay(Pay , [cash,card]), price(Price), smallerThan(300,
Price), rating(Rating), smallerThan(Rating,3.7), distance(Distance), smallerThan(1.9,
Distance).

pick_restaurant(karachi_cafe) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [cafe]),
payment(Pay), memberOfPay(Pay , [card]), price(Price), smallerThan(250, Price),
rating(Rating), smallerThan(Rating,4.0), distance(Distance), smallerThan(3.3, Distance).

pick_restaurant(whiteboard_cafe) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[vegetarian,cafe]), payment(Pay), memberOfPay(Pay , [cash,card]), price(Price),
smallerThan(250, Price), rating(Rating), smallerThan(Rating,3.9), distance(Distance),
smallerThan(2.5, Distance).

pick_restaurant(firangi_bake) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[italian,mexican]), payment(Pay), memberOfPay(Pay , [cash,card]), price(Price),
smallerThan(250, Price), rating(Rating), smallerThan(Rating,3.6), distance(Distance),
smallerThan(2.9, Distance).

pick_restaurant(little_italy) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[vegetarian,italian,continental]), payment(Pay), memberOfPay(Pay , [cash,card]),
price(Price), smallerThan(750, Price), rating(Rating), smallerThan(Rating,3.6),
distance(Distance), smallerThan(2.2, Distance).

pick_restaurant(mamagoto) :- cuisine(Cuisine),  memberOfCuisine(Cuisine , [asian]),
payment(Pay), memberOfPay(Pay , [cash,card]), price(Price), smallerThan(600, Price),
rating(Rating), smallerThan(Rating,4.3), distance(Distance), smallerThan(1.0, Distance).

pick_restaurant(urban_asia) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [asian]),
payment(Pay), memberOfPay(Pay , [cash,card]), price(Price), smallerThan(450, Price),
rating(Rating), smallerThan(Rating,4.2), distance(Distance), smallerThan(9.0, Distance).

pick_restaurant(hitec_nanking) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [chinese]),
payment(Pay), memberOfPay(Pay , [cash,card]), price(Price), smallerThan(350, Price),
rating(Rating), smallerThan(Rating,3.7), distance(Distance), smallerThan(3.7, Distance).

pick_restaurant(grill_box) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [mediterranean]),
price(Price), memberOfPay(Pay , [cash,card]), payment(Pay), smallerThan(200, Price),
rating(Rating), smallerThan(Rating,2.7), distance(Distance), smallerThan(4.2, Distance).

pick_restaurant(freshies_gourmet_salad) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[healthy_food, salad]), payment(Pay), memberOfPay(Pay , [cash]), price(Price),
smallerThan(250, Price), rating(Rating), smallerThan(Rating,3.4), distance(Distance),
smallerThan(2.8, Distance).

pick_restaurant(mustang_terrace_lodge) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[mexican,italian,salad,indian]), payment(Pay), memberOfPay(Pay , [cash]), price(Price),
smallerThan(550, Price), rating(Rating), smallerThan(Rating,3.9), distance(Distance),
smallerThan(3.9, Distance).

pick_restaurant(india_bistro) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [indian]),
payment(Pay), memberOfPay(Pay , [cash,card]), price(Price), smallerThan(500, Price),
rating(Rating), smallerThan(Rating,4.5), distance(Distance), smallerThan(1.4, Distance).

pick_restaurant(drunken_monkey) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
```

```
[healthy_food]), payment(Pay), memberOfPay(Pay , [cash,card]), price(Price),
smallerThan(200, Price), rating(Rating), smallerThan(Rating,4.0), distance(Distance),
smallerThan(3.0, Distance).

pick_restaurant(wich_please) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [fast_food]),
payment(Pay), memberOfPay(Pay , [card]), price(Price), smallerThan(100, Price),
rating(Rating), smallerThan(Rating,3.6), distance(Distance), smallerThan(2.9, Distance).

pick_restaurant(punjabi_bytes) :-  cuisine(Cuisine), memberOfCuisine(Cuisine , [indian]),
payment(Pay), memberOfPay(Pay , [cash]), price(Price), smallerThan(100, Price),
rating(Rating), smallerThan(Rating,3.7), distance(Distance), smallerThan(2.7, Distance).

pick_restaurant(holi_restaurant) :-  cuisine(Cuisine), memberOfCuisine(Cuisine ,
[indian,chinese]), payment(Pay), memberOfPay(Pay , [cash,card]), price(Price),
smallerThan(400, Price), rating(Rating), smallerThan(Rating,2.5), distance(Distance),
smallerThan(0.8, Distance).

memberOfCuisine([Xh|Xt], Y) :- member(Xh,Y), (length(Xt,0) -> !; memberOfCuisine(Xt, Y)).
memberOfPay([H|T], P) :- member(H,P); memberOfPay(T,P).
smallerThan(X,Y) :- X =< Y.
"""

with open("KB.pl", "w") as text_file: text_file.write(KB)

cuisine_options = ['vegetarian', 'indian', 'chinese', 'seafood', 'fast_food',
                   'burger', 'pizza', 'cafe', 'italian', 'mexican', 'continental', 'asian',
                   'mediterranean', 'healthy_food', 'salad']

payment_options = ['cash', 'card']

def get_restaurant():
    chosen_cuisines = []
    chosen_cuisines = [key for key, ans in zip(cuisine.keys(),
                                               [cuisine[key].get() for key in
cuisine.keys()]) if ans]
    chosen_payment = []
    chosen_payment = [key for key, ans in zip(payment.keys(),
                                               [payment[key].get() for key in
payment.keys()]) if ans]

    prolog = pyswip.Prolog() # Global handle to interpreter
    prolog.consult("KB.pl") # open the KB
    pick_restaurant = Functor("pick_restaurant",1)

    # declaring dynamic procedures to allow multiple executions
    # to allow asserta and retractall as inputs change
    prolog.dynamic('cuisine/1')
    prolog.dynamic('payment/1')
    prolog.dynamic('price/1')
    prolog.dynamic('rating/1')
    prolog.dynamic('distance/1')

    # assert user input (list object) into the respective predicates
    prolog.asserta("cuisine(" + str(chosen_cuisines) + ")")
    prolog.asserta("payment("+ str(chosen_payment) + ")")
    prolog.asserta("price("+ str(price.get()) + ")")
    prolog.asserta("rating(" + str(rating.get()) + ")")
    prolog.asserta("distance(" + str(distance.get()) + ")")

    q = prolog.query('pick_restaurant(X)')
```

```
    sols = [] # used to find length of solution
    ans = '' # placeholder for answers for printing
    for value in list(q):
        if value not in sols:
            # iterate over solutions and remove repetitions
            sols.append(value)

    ans = ', '.join(d['X'] for d in sols) # convert list of dicts to string

    if not len(sols): ans = "Sorry, couldn't found any restaurant."

    ans_label.config(text = "You can get food from: " + ans )

    # retract all assertions after obtaining results
    prolog.retractall("cuisine(" + str(chosen_cuisines) + ")")
    prolog.retractall("payment("+ str(chosen_payment) + ")")
    prolog.retractall("price("+ str(price.get()) + ")")
    prolog.retractall("rating(" + str(rating.get()) + ")")
    prolog.retractall("distance(" + str(distance.get()) + ")")

# using TkInter for Graphical User Interface
master = Tk()
master.title("Minerva Hyderabad Restaurant Selector")

##Make some spacing on the left
for row in range(13):
    Label(master, text="").grid(row= row, column = 1, sticky=W)

Label(master, text=" Minerva Hyderabad Restaurant Selector").grid(row=0, sticky=W, column =
2)

Label(master, text="\t\t").grid(row=1, sticky=W)
price = DoubleVar()
Scale(master, label = "Avg Price (Rupees per person)", from_= 100, to = 1000,
      variable = price, orient = HORIZONTAL, length = 300).grid(row = 2,  column = 2)

Label(master, text="\t\t").grid(row=2, sticky=W)
distance = DoubleVar()
Scale(master, label = "Distance from Res Hall (km)", from_= 0.5, to = 5,
      variable = distance, resolution =   0.1, orient = HORIZONTAL, length = 300).grid(row =
3, column = 2)

Label(master, text="\t\t").grid(row=4, sticky=W)
rating = DoubleVar()
Scale(master, label = "Minimum restaurant rating (0 to 5)", from_= 0.0, to = 5.0,
      variable = rating, resolution =   0.1, orient = HORIZONTAL, length = 300).grid(row =
5, column = 2)

#Make checklist for cuisines
Label(master, text="\t\t").grid(row=6, sticky=W)
Label(master, text="Select Cuisines: ").grid(row=7, sticky=W, column = 2)
cuisine = {}
row = 8
for key in cuisine_options:
    cuisine[key] = IntVar()
    Checkbutton(master, text= key, variable= cuisine[key]).grid(row = row, sticky=W, column
= 2)
    row += 1

#Add payment methods
```

```
Label(master, text="Payment Method:").grid(row=7, column = 3, sticky=W)
payment = {}
row = 8
for key in payment_options:
    payment[key] = IntVar()
    Checkbutton(master, text= key, variable= payment[key]).grid(row= row, column = 3,
sticky=W)
    row += 1

Button(master, text='GIVE ME FOOD', command=get_restaurant).grid(row= 15, pady=4, column =
3)
ans_label = Label(master, text= "")
ans_label.grid(row=17, column = 3)
mainloop()
```