Cairo University
Faculty Computers & Artificial Intelligence
Information Systems Department
Spring 2020

Course: CS361 / Artificial Intelligence

Delivery Date: Saturday 11 June 2020

Project ID: PID23868001

Instructor: Dr. Abeer El-Korany

Dr. Khaled Wassif

Final Assessment Project

Search Problem

Topic 1

Name	ID
Mira Mohamed Abd-Elraheem	20170305
Nehal Akram Ahmed	20170318
Nouran Qassem Mohamed	20170322
Nourhan Ihab El-Khodary	20170324

Table of Contents

List of Illustrations

Chapter 1: Introduction

The Abstract Concept of Problem

"Problem-solving is the act of defining a problem, determining the cause of the problem, identifying, prioritizing, and selecting alternatives for a solution, and implementing a solution."

Problem Solving Steps

- Define the Problem.
- Determine the Root Cause(s) of the Problem.
- Develop Alternative Solutions.
- Select a Solution.
- Implement the Solution.
- Evaluate the Outcome.

As shown in **Figure 1**.

Figure 1: Problem Solving Steps

Problem Solving Agents

Problem-solving is identified in computer science field as a part of artificial intelligence that uses a number of known algorithms and heuristic functions to reach a certain goal.

A problem-solving agent is a goal-driven agent and focuses on satisfying the goal by following those steps:

- Goal Formulation: This step is where the agent chooses the goal out of multiple goals and decide the actions that should be taken to reach this goal. This is based on the systems' current state and performance measures.
- Problem Formulation: This step is where the agent decides the actions that should be taken to achieve the goal.
- Initial State: The starting state of the system.
- Actions: A set of possible actions that the agent can take at each state.
- Transition Model: The description of each action outcome.
- Goal Test: It determines if the given state is a goal state.
- Path cost: Based on some heuristic functions each path is assigned a cost that reflects its efficiency and also reflects the whole system performance.

Problem Solving Methods

"Problem solving methods (PSMs) are domain-independent reasoning components, which specify patterns of behaviors which can be reused across applications. "

PSM Importance

- provide strong, model-based frameworks in which to carry out knowledge acquisition.
- supports the rapid development of robust and maintainable applications through component reuse.
- provides a generic reasoning pattern, characterized by iterative sequences of model 'extension' and 'revision', which can be reused quite easily to solve scheduling.

Chapter 2: Different PSM Techniques

Problem solving are mainly designed to help a team through a process of identifying problems, finding a possible solution, and then evaluating the most suitable and appropriate one.

Finding effective solutions to complex problems is not easy process, but by using the right techniques, and methods, you can help your team be more creative and efficient in this process.

So, the techniques mentioned below provide a solid architecture for problem solving that takes the team through all the problem-solving steps.

In-depth problem-solving techniques

It provides a complete end-to-end process for developing effective solutions. If you're looking for a problem-solving model, these processes are a great place to start:

1- **Six Thinking Hats:** is a classic technique for identifying the problems that need to be solved and enables your team to consider them from different angles, whether that is by focusing on facts and data, creative solutions or by considering why a particular solution might not work.

2- **Lightning Decision Jam:** it is a type of meetings; they exercise it to:

- Identify and define the problem quickly.
- Find a lot of solutions.
- Find the best solution, based on identifying the effort and impact of the best solutions.
- The agreed action steps can be started and implemented as soon as the meeting is over

Techniques to identify and analyze problems

It focusses on the problem identification (define the problem you want to solve) and analysis part of the process that sets the foundation for developing effective solutions. For example:

1- **Problem tree** is a tool to clarify the hierarchy of problems addressed by the team members within a design project, the hierarchy could be from most important to least important problems.

2- **SWOT analysis** is a framework used to help an individual or organization identify strengths, weaknesses, opportunities, and threats associated with business competition or project planning and to develop strategic planning.

Problem solving techniques for finding solutions

The end goal of any problem-solving activity and whatever problem-solving technique you use, organizational challenges/problems can only be solved with an appropriate solution. For example:

1- **Mind Spin:** it's a fast and loud method to enhance brainstorming within a team. By doing multiple rounds, your team begin to generate possible solutions before moving on to developing those solutions and encouraging further ideation.

2- **Four-step sketch:** is an exercise that helps people to create well-formed concepts through a structured process that includes:

- Review key information
- Start design work on paper
- Consider multiple variations
- Create a detailed solution

This exercise is preceded by a set of other activities allowing team members to clarify the challenge they want to solve.

Chapter 3: Travel Agent (Search Problem)

System Components

Any expert System must have knowledge base, interface engine and user interface as shown in **Figure 2**.

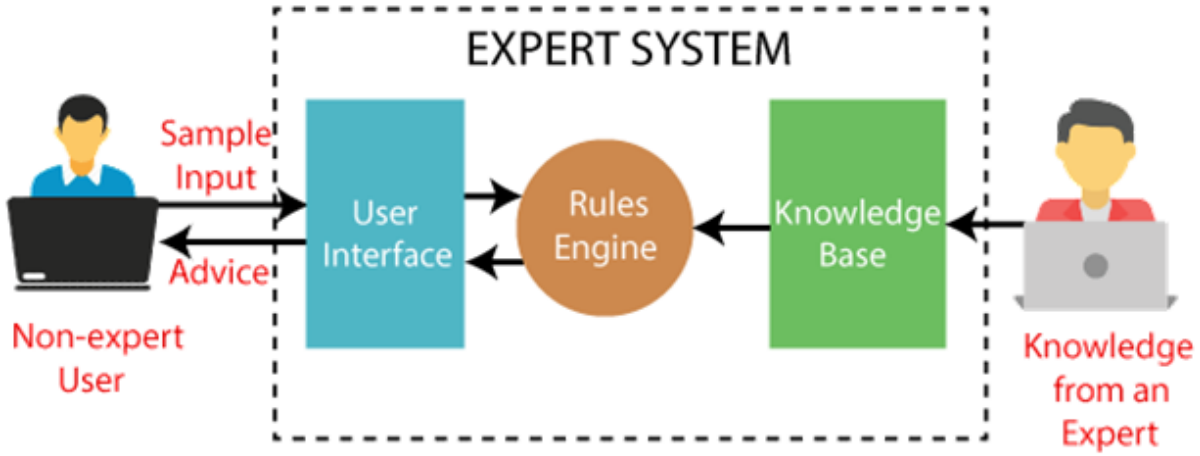

Figure 2: Expert System Components

Knowledge Base

It is the repository of facts. It stores all the knowledge needed in the problem domain. It is the container of the facts.

In the travel agent problem, we have many facts:

- 150 facts about the flight information, it represents the starting city, destination city, departure time, arrival time, flight number and the days that this flight take off.
  Example: flight ("London", "Liverpool", time (21, 00), time (22, 00), "BA133", [sun, mon, thu, fri]).
- 20 facts about the city position, it represents the city and its latitude and longitude.
  Example: city ("Alexandria", 31.2, 29.95).
- 7 facts about the day and its next day.
  Example: nextDay (mon,tue).

Interface (Rules) Engine

It is the brain of the expert system. It contains a set of rules that solves a specific problem, using the knowledge base and facts and apply it on the user's query.

In the program, we are using A* algorithm. Its rules are:

- Gets the best flights within a small-time range which the user requests and the shortest path.
- Gets all the options of flight from the knowledge base of the flights.

- Get the cost of each path which is G (x) function: the cost of previous flight plus the waiting time between the previous flight and the chosen flight, H (x) function: getting the distance between the traveling city and destination and getting the estimate time the flight will take by the average velocity of a commercial airplane and dividing it by the distance. Then add the G (x) and H (x).
- Finally compare the costs getting the lowest path cost and represent it to the user.
- If no solution was found within the range, the range would be expanded 2 days: one before first day within the range and one next to the last day within the range.
- If still no solution found, tell the user that no solution is found.

The code will be found in the Appendix.

User Interface

It takes the query that the user enters and passes it to the interface engine. Then takes the results and represent it to the user.

We are using prolog as user interface. The user enters the query which is the traveling city, the destination city and the range he wants to reach his destination within it.

Example: travel ("Rome", "New York", [sun, mon]).

Test Cases

First Case: A flight that is direct from the traveling city to the destination within the given range

EX: travel **("Alexandria", "Aswan", [mon]) & travel ("Chicago", "New York", [sat])** as shown in **Figure 3**.

```
?- travel("Alexandria", "Aswan",[mon]).
A solution is found
Flight number: MS005 From Alexandria to Aswan. 11:0 and arrival time 12:15.
true.

?- travel("Chicago", "New York",[sat]).
A solution is found
Flight number: DL044 From Chicago to New York. 9:0 and arrival time 11:18.
true.
```

Figure 3: Directly Within the Range

Second Case: Flights are found but indirectly with transits flight in between the travel city and the destination within the given range

EX: **travel ("Alexandria", "Port Said", [sun, mon])** as shown in **Figure 4**.

```
?- travel("Alexandria", "Port Said",[sun,mon]).
A solution is found
Flight number: MS003 From Alexandria to Cairo. 9:15 and arrival time 10:0.
Flight number: MS013 From Cairo to Port Said. 11:0 and arrival time 11:20.
true.
```

Figure 4: Indirect Within the Range

Third Case: No flights found within the range that the user gave whether in direct or indirect flights. So, the range is expanded as mentioned before. The flight is found directly.

EX: **travel ("Alexandria", "Aswan", [sun])** as shown in **Figure 5**.

```
?- travel("Alexandria", "Aswan",[sun]).
 No flights found in this range, so we expand the range
A solution is found
Flight number: MS005 From Alexandria to Aswan. 11:0 and arrival time 12:15.
true.
```

Figure 5: Direct Flight with Expanded Range

Fourth Case: No flights found within the range that the user gave whether in direct or indirect flights. So, the range is expanded as mentioned before. The flights are indirect with transits.

EX: **travel ("Alexandria", "Port Said", [mon])** as shown in **Figure 6**.

```
?- travel("Alexandria", "Port Said",[mon]).
A solution is found
Flight number: MS003 From Alexandria to Cairo. 9:15 and arrival time 10:0.
Flight number: MS013 From Cairo to Port Said. 11:0 and arrival time 11:20.
true.
```

Figure 6: Indirect Flights with Expanded Range

Fifth Case: No flights are found even after expansion of the range the user gave.

EX: **travel ("Cairo", "Chicago", [sun, mon, tue])** as shown in **Figure 7**.

```
?- travel("Cairo", "Chicago",[sun,mon,tue]).
 No flights found in this range, so we expand the range
No solution
true.
```

Figure 7: No Solution Found

Sixth Case: the user does not enter a range, or the cities entered are in correct.

EX: **travel ("Cairo", "Paris", [])** & **travel ("Cairo", "san", [sun])** as shown in **Figure 8**.

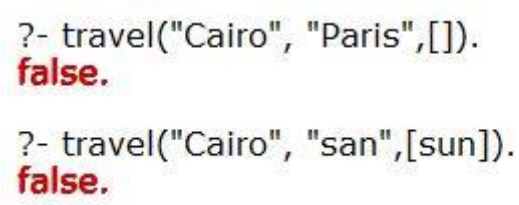

Figure 8: Wrong Entries

References

- Dieter Fensel and Enrico Motta (2001). Structured Development of Problem-Solving Methods.
  http://ksi.cpsc.ucalgary.ca/KAW/KAW98/fensel2/

- Martin Molina and Yuval Shahar (1996). Problem-Solving Method Reuse and Assembly: From Clinical Monitoring to Traffic Control.
  http://ksi.cpsc.ucalgary.ca/KAW/KAW96/molina/kaw_psm.html

- Tutorial and Example (2017). Problem-Solving in Artificial Intelligence.
  https://www.tutorialandexample.com/problem-solving-in-artificial-intelligence/

- Joost Breuker (2005). Components of Problem-Solving and Types of Problems.
  https://link.springer.com/chapter/10.1007/3-540-58487-0_7

- James Smart (2020). 35 Problem-Solving Techniques and Activities to Create Effective Solutions.
  https://www.sessionlab.com/blog/problem-solving-techniques/

- Michael Smart (2019). Running a Remote Lighting Decision Jam: How to Run One with a Remote Team.
  https://miro.com/blog/lightning-decision-jam-remote-teams/

- Mitchell Grant (2020), Gordon Scott (reviewed). Strength, Weakness, Opportunity and Threat SWOT Analysis.
  https://www.investopedia.com/terms/s/swot.asp

Appendix