Assignment 2

Duc Minh Le (Tom Le)

ID: 3561723

Athabasca University

COMP 456 – Artificial Intelligence

Dr. Sidi Esmahi

Steve Leung

# Question 1

# 1. A problem may not have an exact solution because of inherent ambiguities in the problem statement or available data. Medical diagnosis is an example of this. A given set of symptoms may have several possible causes; doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment. Vision is another example of an inexact problem. Visual scenes are often ambiguous, allowing multiple interpretations of the connectedness, extent, and orientation of objects. Optical illusions exemplify these ambiguities. Vision systems often use heuristics to select the most likely of several possible interpretations of as scene.

# 2. A problem may have an exact solution, but the computational cost of finding it may be prohibitive. In many problems (such as chess), state space growth is combinatorically explosive, with the number of possible states increasing exponentially or factorially with the depth of the search. In these cases, exhaustive, brute-force search techniques such as depth-first or breadth-first search may fail to find a solution within any practical length of time. Heuristics attack this complexity by guiding the search along the most “promising” path through the space. By eliminating unpromising states and their descendants from consideration, a heuristic algorithm can (its designer hopes) defeat this combinatorial explosion and find an acceptable solution.

# Question 2

Written in **a2q2.pl**

This program return an acyclic path (list of nodes) from node A to Z if found

Graph: syntax graph(Nodes, Edges)

Nodes: list of nodes of the graph

Edges: list of edges, syntax: edge(startNode, endNode), of the graph

Path: the result acyclic path from A to Z if found

Path1: a path in the graph, used for constructing the resulting Path

Usage: path(A, Z, Graph, Path).

E.g: path(a, d, graph([a, b, c, d], [edge(a, b), edge(b, c), edge(b, d), edge(c, d), edge(d, b)]), Path). should return Path = [a, b, d]

# Question 3

Written in **a2q3.pl**

This program checks all the possibility of paths from node A to node Z using breadth-first search, returns the most cost-effective path and its cost if exist.

path(): recursively search through the graph by breadth-first algorithm, and calculate the cost fron node A to Z using the given edges data

1. Line 23, initial call to path()
2. Line 25-34: breath-first search

shortest\_path():

1. Line 38, call the recursive path()
2. Line 39-41: making sure that the 1st call is the most cost-effective path by comparing all the possible paths

Usage: shortest\_path(A, Z, Path, Cost).

E.g: shortest\_path(a, d, Path, Cost). should return Path = [a, b, c, d], Cost = 6

shortest\_path(c, a, Path, Cost). should fail as there are no path from c to a

# Question 4

Written in **a2q4.pl**

This program recursively checks and splits the given word into appropriate syllabi that satisfy the following 2 cases below

split(): recursively advance the list of characters, check for possible syllabus.

1. Line 19, empty input, end of char list
2. Lines 21-25, vowel – consonant – vowel case
3. Lines 27-31, vowel - consonant – consonant – vowel case
4. Line 33-35, no matching case, advance the list by one char

syllable():

1. Line 39, split the input into a list of chars
2. Line 40, call the recursive split(), display the split result if syllable, otherwise display the initial given input

Usage: Syllable(Input, Result).

E.g: Syllable(analog, R). should return ana-log

Syllable(bumper, R). should return bum-per

Syllable(calculator, R). should return cal-cul-ato-r