**Question 2:**

Examine the AI literature to discover whether the following tasks can currently be

solved by computers:

**Solution:**

**a**. Playing a decent game of table tennis (Ping-Pong).

* Yes, tech company Omron has a bot named “Orpheus”
* <https://www.theverge.com/2017/10/9/16448488/table-tennis-playing-robot-ai-forpheus-omron>

**b**. Driving in the center of Cairo, Egypt.

* Yes, it should be theoretically possible thanks to Google’s Self-Driving Car project, Waymo but practically there might be issues due to the traffic issues in cairo
* <https://www.google.com/selfdrivingcar/>
* <https://www.citymetric.com/transport/cairo-s-traffic-problems-are-costing-egypt-around-4-cent-its-gdp-369>

**c**. Driving in Victorville, California.

* This should be easier than driving in Cairo since the traffic is much more organized and has less density of vehicles than Cairo.
* Google’s self-driving car is the solution here as well.

**d**. Buying a week’s worth of groceries at the market.

* This is possible thanks to Amazon’s grocery store
* <https://venturebeat.com/2018/01/21/amazon-set-to-open-doors-on-ai-powered-grocery-store/>
* Keeping security issues, such as shoplifting at bay is a challenge
* Scaling this when the number of customers increase is a challenge.

**e**. Buying a week’s worth of groceries on the Web.

* Yes, this can be done easily by using software -based AI bots.
* Amazon’s dash button is an example.
* <https://www.amazon.com/Dash-Buttons/b?ie=UTF8&node=10667898011>

**f**. Playing a decent game of bridge at a competitive level.

* This is possible by using AI using planning based algorithms
* <http://www.cs.umd.edu/~nau/papers/smith1998computer.pdf>
* The planning based bridge player won Baron Barclay Bridge Challenge in’97.

**g**. Discovering and proving new mathematical theorems.

* This has been proposed but not done yet, though professors are doing research in this area
* <http://www.slate.com/articles/health_and_science/science/2015/03/computers_proving_mathematical_theorems_how_artificial_intelligence_could.html>
* A challenge would be to think creatively like a human, because the human brain has not been understood 100% well yet.

**h**. Writing an intentionally funny story.

* <https://www.youtube.com/watch?v=CnMStLTsZ3M>
* This is possible to a limited degree, but humor requires creativity and finesse which are hard for a robot to quantify or assess.

**i**. Giving competent legal advice in a specialized area of law.

* Giving legal advice is not possible as of today, since it requires logical reasoning and natural language processing.
* https://www.lawsociety.org.nz/lawtalk/lawtalk-archives/issue-887/can-artificial-intelligence-ever-give-legal-advice

**j**. Translating spoken English into spoken Swedish in real time.

* Yes, this is possible thanks to Google translate and compute capabilities of today.
* <https://techcrunch.com/2015/01/14/amaaaaaazing/>

**k**. Performing a complex surgical operation.

* Yes, this is possible today as robotics becomes more powerful.
* <https://spectrum.ieee.org/the-human-os/biomedical/devices/in-fleshcutting-task-autonomous-robot-surgeon-beats-human-surgeons>
* In tasks which require more accuracy and precision such as flesh cutting, a robotic arm can be used to provide better incision.

**Question 3:**

Provide a formulation of the wolf, the sheep, the cabbage, and the astronaut problem discussed in class. What needs to be known(perceived)? What are the higher-level decisions to make? What control actions would be driven once the higher-level decisions are made? (10 points)

**Solution:**

This is a state-space search problem.

What needs to be perceived? The state. The agent needs to know

* Initial state
* The set of possible actions
* The transition models
* Check if goal has been attained
* A metric such as a path cost which needs to be optimized or satisfied to define the termination of the problem.

The higher-level decisions to make are:

* Which is the strategy that is going to be used: E.g., Search or Motion Planning using sampling or if it linear programming based approach
* The approach to take depends on the computational capability at our disposal as well as granularity of solution we seek.
* We need different approaches when we try to perform the task as fast as possible vs performing the task with shortest route.

The set of states are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **State** | **Shore 1** | **Shore 2** | *Actions* | *Boat* |
| S0 |  | Sheep, Cabbage, Wolf | A0 |  |
| S1 |  | Wolf, Cabbage | A1 | Man, Sheep |
| S2 | Sheep | Wolf, Cabbage | A2 | Man |
| S3 | Sheep | Wolf | A3 | Man, Cabbage |
| S4 | Sheep, Cabbage | Wolf | A4 | Man |
| S5 | Cabbage | Wolf | A5 | Man, Sheep |
| S6 | Cabbage | Sheep, Wolf | A6 | Man |
| S7 | Cabbage | Sheep | A7 | Man, Wolf |
| S8 | Cabbage, Wolf | Sheep | A8 | Man |
| S9 | Cabbage, Wolf, Sheep |  | A9 | Man |

Problem is a tuple P = <S,A,P,G,r>

**Set of States (S):** The possible set of states is the (Shore1, Shore2) values which correspond to who are present in the shore (this can be represented in 3 bits, 1 for each cabbage, wolf or wolf)

**Actions (A)**: The action is the boat (Boat) occupancy values

**Transition Model(P)** can be represented as shown in the table above. It can also be drawn as a finite state machine.

**Goal(G)** 🡪 State S9

**The Path cost (r)** 🡪 can be set as a function of the number of steps.

**Question 4:**

Formulate the problem of a team of robots autonomously vacuuming a home in the least amount of time. What needs to be known (perceived)? what are the higher-level decisions to make? What control actions would be driven once the higher-level decisions are made? (10 points)

**Solution**:

In line with the previous problem, the problem is a tuple <S,A,P,G,r>. Let us consider the simplest case wherein each robot stores the positions of the rest of the robots in its memory. We will also assume that the area to be cleaned is sub-divided as grids.

**Set of States(S):** The possible set of states is the n-tuple *locations* = (pos­1, pos2, pos3, …, posn).

The status of each of the grids is also stored as a Boolean variable say, *is\_clean* = (is\_clean1, is\_clean2, is\_clean3..., is\_cleann).

True meaning that the grid is clean & likewise, false means that the grid needs cleaning.

**Actions (A)**: The action can be one of the following: move north, move south, move east or move west.

**Goal(G):** It is the state where *is\_clean* = (T, T,…, T)

**Path Cost(r):**  can be a function of the distance moved by each of the bots. It can also be some form of cost such as power consumed by the bot.

**Transition Model(P)** can be represented as shown in the code below for a bot. It can also be drawn as a finite state machine.

|  |  |
| --- | --- |
| Percept Sequence | Actions |
| North\_is\_clean= T  South\_is\_clean=T  East\_is\_clean=T  West\_is\_clean=F | Move west |
| North\_is\_clean= T  South\_is\_clean=T  East\_is\_clean=F  West\_is\_clean=T | Move east |
| North\_is\_clean= F  South\_is\_clean=T  East\_is\_clean=T  West\_is\_clean=T | Move north |
| North\_is\_clean= T  South\_is\_clean=F  East\_is\_clean=T  West\_is\_clean=T | Move south |

We need to also track the position of each of the bot and move only when the corresponding cell is empty.

What needs to be perceived? The state. The agent needs to know

* Initial state
* The set of possible actions
* The transition models
* Check if goal has been attained
* A metric such as a path cost which needs to be optimized or satisfied to define the termination of the problem.

The higher-level decisions to make are:

* Which is the strategy that is going to be used: E.g., Search or Motion Planning using sampling or if it linear programming based approach
* The approach to take depends on the computational capability at our disposal as well as granularity of solution we seek.
* We need different approaches when we try to perform the task as fast as possible vs performing the task with shortest route.
* In this case, we need to keep in mind the is\_clean status of each of the grids and the locations of the each of the bots so that they do not end up colliding into one another.

**Question 5:**

Prove that KL divergence is always non-negative. Hint: Use Jensen’s inequality (10 points)

**Solution:**

A close up of text on a whiteboard

Description generated with high confidence

**Question 8:**

**Solution:** Python code is attached.

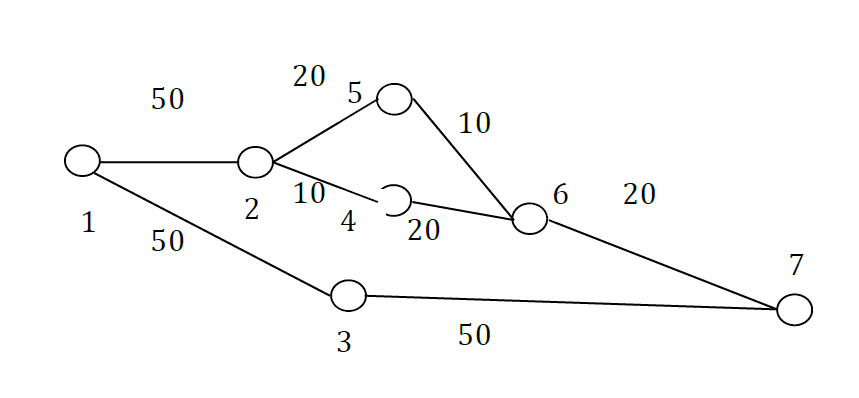
**Breadth First Search** takes 25 iterations to reach (5,5) from (1,1). **Depth First Search** should take 10 iterations theoretically (but I am getting different results for my code) for the same. This is expected since breadth first expands in the nearest neighbor’s layer of nodes and depth first goes through the layers.

The most optimal path is a sequence of 8 steps. Neither BFS nor DFS seems to give it for me.

I believe we will need an algorithm such as Djikstra’s algorithm for the same.

**Question 7:**

Implement breadth first and depth first search to find the optimal path from node 1 to node 7 in the following graph search problem:



The numbers 1 to 7 denote the index of the vertex. The numbers on the edges denote the cost to traverse from one vertex to the other.

1. Present your answer as a sequence of vertices and the total cost of the entire transition.
2. What is the optimal sequence and cost?
3. Does your algorithm achieve it?
4. Report the number of total computation time and calculations (you can count the number of times the search function was called).

**Solution:**

graph = {

'node1' : set(['node2', 'node3']),

'node2' : set(['node1', 'node4', 'node5']),

'node3' : set(['node1', 'node7']),

'node4' : set(['node2', 'node6']),

'node5' : set(['node2', 'node6']),

'node6' : set(['node4', 'node5', 'node7']),

'node7' : set(['node3', 'node6'])}

def bfs\_paths(graph, start, goal):

queue = [(start, [start])]

while queue:

(vertex, path) = queue.pop(0)

for next in graph[vertex] - set(path):

if next == goal:

yield path + [next]

else:

queue.append((next, path + [next]))

print "Breadth First Search Paths"

print list(bfs\_paths(graph, 'node1', 'node7'))

def dfs\_paths(graph, start, goal):

stack = [(start, [start])]

while stack:

(vertex, path) = stack.pop()

for next in graph[vertex] - set(path):

if next == goal:

yield path + [next]

else:

stack.append((next, path + [next]))

print "Depth First Search Paths"

print list(dfs\_paths(graph, 'node1', 'node7'))

Ref: <http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/>

**Breath First Search** gives the paths in the following order:

[['node1', 'node3', 'node7'], ['node1', 'node2', 'node5', 'node6', 'node7'], ['node1', 'node2', 'node4', 'node6', 'node7']]

Cost of Path1 = 100

Cost of Path2 = 100

Cost of Path3 = 100

Optimal Cost = 100. All paths are equally optimal. Yes, the algorithm achieves it.

**Depth First Search** gives the paths in the following order:

[['node1', 'node2', 'node4', 'node6', 'node7'], ['node1', 'node2', 'node5', 'node6', 'node7'], ['node1', 'node3', 'node7']]

Cost of Path1 = 100

Cost of Path2 = 100

Cost of Path3 = 100

Optimal Cost = 100. All paths are equally optimal. Yes, the algorithm achieves it.

BFS and DFS are exploration based algorithms. It returns all possible paths between two nodes in a graph. However, in terms of number of hops, BFS gives a shorter path first with only 3 vertices, whereas DFS gives it along with the all the paths.

**Question 10:**

**A close up of text on a whiteboard

Description generated with high confidence**

Entropy & KL divergence have been calculated as above.

A close up of text on a whiteboard

Description generated with very high confidence

Mutual Information can be calculated by knowing the joint pmf.