KNOWLEDGE RETRIEVAL

Sai Sivasankaran Chittoor Madhusudhan

Abstract— **The Functional Object-Oriented Network (FOON), presented in the paper, is a structured knowledge representation. This representation is obtained in the form of graphs which contains several functional units. These functional units are made up of three elements called Motion, State and Object. These functional units and elements must be represented in such a way that a robot can easily manipulate the graph and can smoothly traverse the corresponding tree, there by solving problems using novel knowledge gathered from different sources and online instructional videos.**

I. Introduction

Most of the times you start wondering how an Artificial Intelligence (AI) powered device or robot functions. The ease with which the robot starts to understand leaves everyone baffled. In this paper, we are going to discuss about how one such AI powered robot operates from the inside. The proposed innovative FOON focuses on task manipulation, which is influenced by both object states and functional motions, which are represented as connected nodes in a FOON. The links between these nodes reflect two-way dependencies, in which functional motions are influenced by the states of the objects, and the consequent state is influenced by the functional motion. Not only does a FOON provide structured knowledge about objects and their states, but it also provides structured knowledge about the relationships between functional motions and states. From a manipulation goal, a FOON can be searched to find the objects involved, their desired states, and the functional motions to achieve those states.

II. Functional Object-Oriented Network

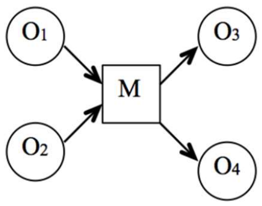
The Functional Object-Oriented Network (FOON) is a 2-way network that contains object, state, and motion nodes. We give input prior to the motion nodes, and we get output after the motion nodes. A rule is that an object node must be connected with motion nodes and that motion node must be connected to another new object node. No two motion nodes or no two object nodes must not be connected to each other.

A. FOON nodes

Nodes basically have two states, Object (O) and Motion(M). We additionally have State Nodes(S), which represents the states of objects. N_o represents an object in certain state which is either used by the manipulator or is interacting with another object. For example, while breaking egg in a cooking task, a person breaks the egg using a knife by striking it in between. Here both the egg and knife represent objects. Initially the egg is in the state whole, and knife's state in clean. After the beating motion, which is motion node. N_M is the type of motion node. No two graphs are same in FOON. No two object nodes are also same. They are different in terms of attributes and settings. Every Functional unit basically starts with an object node, followed by its state. These are inputs. After inputs there is a Motion node, which is the transition from input to output. And there are output object nodes and states. This is the basic structure of a FOON functional unit. No two objects are linked to each other. Objects before the Motion nodes are the inputs and objects after the motion nodes are the outputs. Basically, inputs and outputs are separated by a motion node.

B. *FOON Edges*

A FOON is characterized as a directed network since certain nodes are the consequence of interactions between other nodes. An edge, represented by the letter E, connects two nodes. Although edges are formed from an object node to a motion node or vice versa, it's important to keep in mind that no two objects or motions are related.



Here, M is the Motion node. O1 & O2 are input object nodes, O3 & O4 are output object nodes.

C. *Functional Unit:*

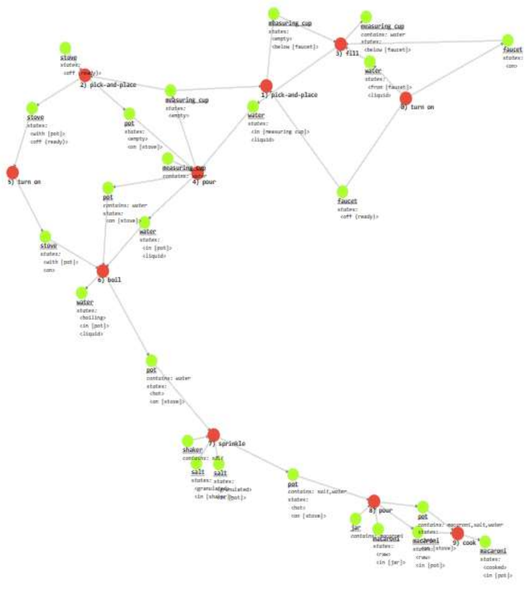
The smallest learning unit in a FOON is the functional unit. It displays the connection between one or more items and a single functional motion to which they are all tied. In other words, each unit symbolizes a single, atomic action that takes place as part of a broader activity. As shown in above figure, input object nodes are connected to the edges leading to the functional motion node, while output object nodes are connected to the edges heading away from the functional motion node.

D. *Network:*

FOONs are expressed using traditional graph representations such as adjacency matrices and adjacency lists. We use an adjacency matrix to represent the network in order to make it easier to construct a digraph and perform network analysis. Each node in the matrix is represented by a row, with its connections to other nodes represented by columns. An edge from node Ni to node Nj has a value of 1, which preserves edge directional properties; an index has a value of 0 if two nodes are not connected. A node list is included with the adjacency matrix, which keeps track of all the objects and motion nodes identified in the graph.

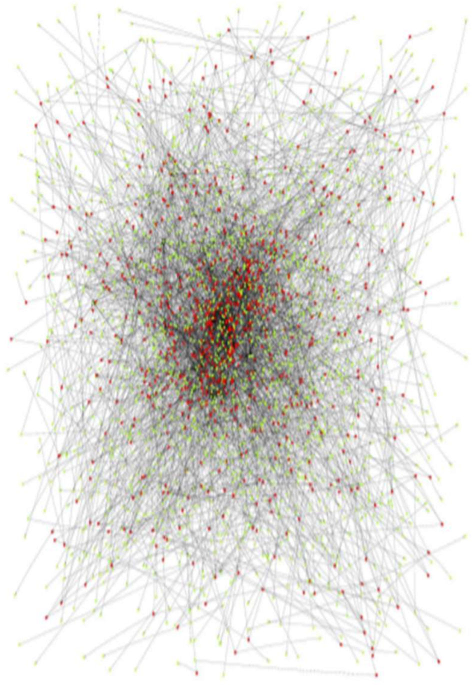
III. FOON Learning From Video:

In an ideal world, a FOON could be automatically trained by monitoring human behavior. We presently build many tiny sets of functional units by labeling instructional movies due to the intricacy of object, state, and motion identification. These functional units were typed in manually from a set of videos. These videos are the recipes of some common household foods.



FOON Graph

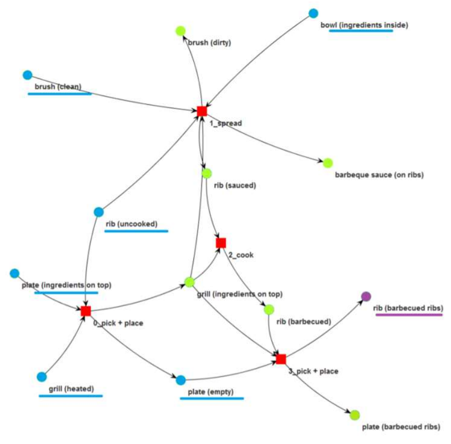
Then, for each video, we automatically integrate them into a single subgraph. The subgraphs from each video are then integrated into a massive network, which we refer to as a universal FOON. Even though the functional units are manually labeled, the process of combining the information into a universal FOON is done algorithmically. As a result, creating a FOON can be thought of as a semi-automated process.



FOON graph with all the videos

IV. Foon Analysis And Methodology:

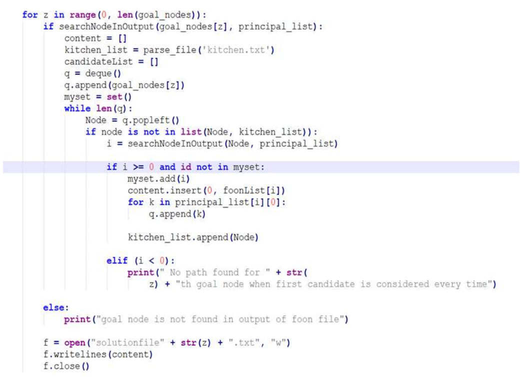
In the FOON for motion creation, motion nodes are utilized to identify motion type. The universal FOON can be thought of as a structured representation of data that can be used to solve information manipulation problems. Given a desired goal and a set of available objects, there are two formal techniques for producing manipulations from the FOON: 1) creating the motions required to finish the assignment, and 2) extracting a task tree. The retrieval algorithm used is a combination of the breadth-first search and the depth-first search as a specific application of the branch-and-bound algorithm in C that we can execute in its entirety. This is determined by the availability of objects in each functional unit as listed in K. Weights that can act as heuristics or constraints on the development



Task Tree

of a task tree can also be added to the search operation. These heuristics can take the form of a cost value linked with each motion, influencing the functional units that are added to T.

To increase the quality of our search, we can use cost values that reflect the complexity of motions or by determining the quickest path to a goal's completion. The path we take is entirely contingent on the objects in the robot's environment being available. Our network's capacity to mix and blend knowledge into a single network makes it extremely strong and valuable. Many alternative task trees for different contexts can be found within a universal FOON. These options could be a whole new way of carrying out a task, like how there could be multiple ways to prepare a specific meal. The best path is chosen by taking into account all the possible paths that the robot can take before arriving at a conclusion based on many external factors like availability of ingredients etc., Our job sequences will not always follow the complete procedure from a single video source. For example, there are numerous methods to make a meat sauce, and by knowing how to make sauces with a range of ingredients,



```
for z in range(0, len(goal_nodes)):
    if searchNodeInOutput(goal_nodes[z], principal_list):
        content = []
        kitchen_list = parse_file('kitchen.txt')
        candidateList = []
        q = deque()
        q.append(goal_nodes[z])
        myset = set()
        while len(q):
            Node = q.popleft()
            if node is not in list(Node, kitchen_list)):
                i = searchNodeInOutput(Node, principal_list)

                if i >= 0 and id not in myset:
                    myset.add(i)
                    content.insert(0, foonList[i])
                    for k in principal_list[i][0]:
                        q.append(k)

                    kitchen_list.append(Node)

                elif (i < 0):
                    print(" No path found for " + str(
                        z) + "th goal node when first candidate is considered every time")

    else:
        print("goal node is not found in output of foon file")

    f = open("solutionfile" + str(z) + ".txt", "w")
    f.writelines(content)
    f.close()
```

Task Tree Retrieval Algorithm

we can compensate for the lack of specific ingredients that would be required if we followed a single recipe. The novelty stems not just from the variety of task sequences available, but also from the freedom with which we prepare the meals.

Methodology:

Initially we parse the goal node file into a list. We take this goal nodes and iterate one after the other in a for loop. Similarly, we parse the universal FOON file and put it in a list called master list. We also have a kitchen file called kitchen.txt, which contains kitchen items. Goal node is pushed into the queue. Start a while loop when the queue has items. Pop the first element of queue so that we can find the desired input in the FOON file. But before that check whether the file is in the kitchen. If it is present over there, then the file is ready. If the item is not there, check the master list where all the functional units are present. If we get the desired element, insert that array index into a separate list called FOON output. Pick inputs from the desired goal nodes, insert into queue, and repeat the while loop till the condition is satisfied. This will continue till all the inputs are present in the kitchen. We can also modify our search in terms of number of inputs given, i.e., add heuristics so that least possible steps are followed to prepare the same food item while saving time. For example, if you find that scrambled eggs can be prepared with either {egg, oil, cheese, onion} or {egg, oil, salt}, take the path that requires {egg, oil, salt} because the robot takes fewer input objects.

Objects	**Without Heuristic**	**With Heuristic**
Greek Salad	71	67
Ice	23	39
Macaroni	7	7
Sweet Potato	14	89
Whipped Cream	1	1

These are the number of Functional Units found with and without Heuristics

We could find all the items in the given kitchen. If objects are not in the kitchen, then we have to use the FOON file, but we could find all the objects in the kitchen and we could prepare the FOON graph effectively.

V. Conclusion:

This study presents the functional object-oriented network, or FOON, as a representation of manipulation tasks that links interactive objects to their functional motions. A FOON organizes information regarding the relationship between object states and functional object motions, which is useful for learning manipulation tasks and understanding human actions. We devised a system for building functional units based on abstracted knowledge obtained from online instructional videos, especially cookery courses, and manually annotated by human users. To get insight into the network's structure, centrality measurements were used. Given a manipulation goal, our searching technique can recover manipulation information from a FOON. To store manipulation knowledge, a task tree sequence containing a series of concerned objects, manipulation motions, and impending goals is utilized. These task trees will not always follow the same precise technique as a single recipe or movie, making them a flexible and creative approach of modifying objects based on information from multiple sources.

VI. References

[1] Dr. Yu Sun, Professor for Artificial Intelligence, University of South Florida, Tampa, Florida, USA.

[2] David Paulius, Yongqiang Huang, Roger Milton, William D. Buchanan, Jeanine Sam, and Yu Sun: Functional Object-Oriented Network for Manipulation Learning,