Expanding Functional Object Oriented Network and Searching Task Trees

Alec Martin

***Abstract*—In this report I will talk about what we did for Project 2. First I will open with my motivation/related work and then I'll talk about the video annotation, the merging, and the searching. Finally I will end with a discussion section.**

I. Introduction

MY motivation for this project was because I enjoy learning about robotics. I think it's very cool to see how much robots have progressed over the years and how they will continue to progress.

Some related work that I have done was in a previous class called robot vision. In that class, all though it did not directly relate to robot movement, we coded some algorithms to conduct things like edge detection and object recognition using AlexNet. This was similar to Project 2 because while we did not assist in this aspect, the robot that will be using FOONs to make kitchen items will also need to be able to recognize items and the different states that they're in. This first introduction to robot and robot vision inspired me to learn more and take this class.

II. Video Annotation

The first step for project 2 was to watch a cooking tutorial video and create a functional unit tree for the recipe. This was accomplished by first being given a text file that listed the various object/states and motions for each unit and then we went and formatted it properly and added start/end times for the motions if they were in the video.

My two recipes I annotated were Fried Salmon, shown in Fig. 1, and Apple-Cinnamon Cake, shown in Fig. 2.

III. Merging

Th next step for the project was to code a merging algorithm that could take in two files in functional unit format and merge together only the unique ones. I chose to write my algorithm in python and construct the graphs and their subnodes using dictionaires. This allowed me to parse the graph quickly without having to worry about constructing classes to represent the graph.

Next, my algorithm assumes that the files are unique within themselves so it takes the first file and essentially parses all of the functional units and rewrites them to a new merged file. From then on, it parses out the second file and for every functional unit in the parsed out list of units, it compares it to all existing units already written to the file. If the functional

Course Name: Intro to Robotics. The work is supervised by Professor Yu Sun and TA David Paulius.

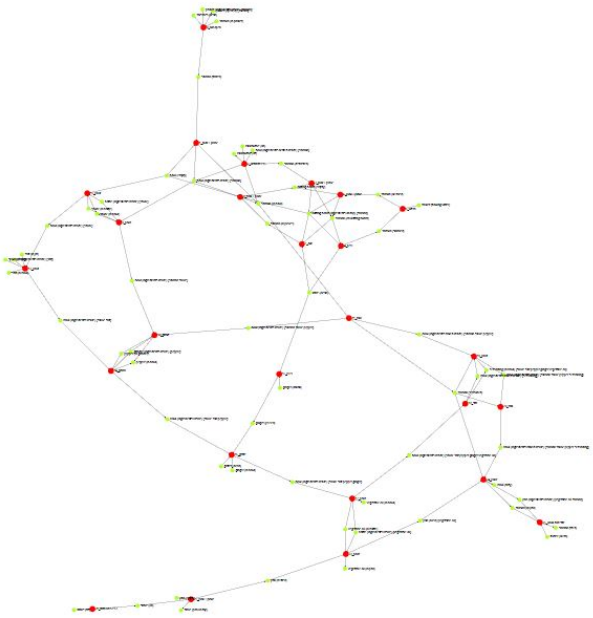

Fig. 1. The visualized tree of the Fried Salmon

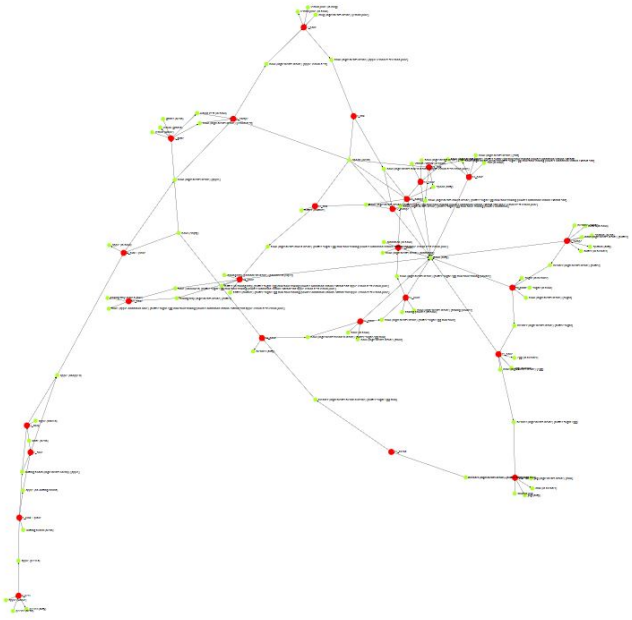

Fig. 2. The visualized tree of the Apple-Cinnamon Cake

unit is new it gets written to the file and if its a duplicate then it is skipped.

Finally, after every functional unit has been processed you are left with a text file containing the unique functional units of each text file.

For this part we were supposed to merge together the two graphs that we made in the previous section and then merge those into the universal graph. Because my two recipes were so different from each other there were no functional units that were the same and the algorithm essentially combined the two files. Additionally when merging into the universal graph, the functional units were to unique to match up with any existing functional units and so they were all added to the universal graph.

The output for Step 1 is shown in Fig. 3, Fig. 4, and Table I

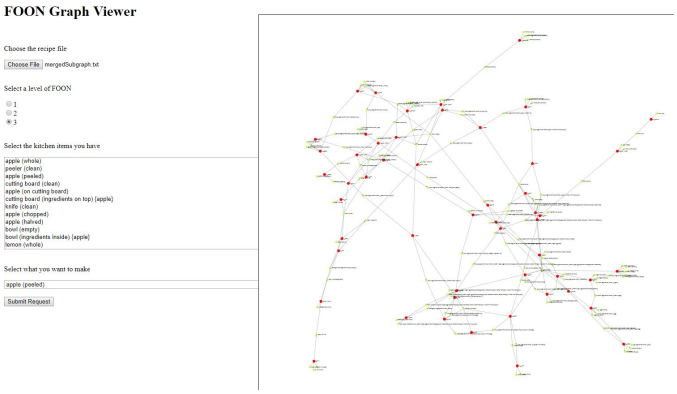

Fig. 3. The graph visualization of step 1 merging

```
(venv) amartin@amartin:~/code/mergeFOON$ python src/mergeFOON.py --file1 resource/0068FOON.txt --file2 resource/P20_04FOON.txt --name mergedSubgraph.txt
# of parsed FOONs from File resource/0068FOON.txt = 26
# of parsed FOONs from File resource/P20_04FOON.txt = 26
# of total parsed FOONs = 52
# of newly added FOONs = 26
# of duplicate FOONs = 0
All Functional Units merged written to mergedSubgraph.txt
New Functional Units added written to mergedSubgraph_added.txt
```

Fig. 4. The terminal output of step 1 merging

TABLE I
STEP 1: MERGING 2 VIDEO RECIPES

# of Parsed FOONs from File 1	26
# of Parsed FOONs from File 2	26
Total # of Parsed FOONs	52
Newly added FOONs	26

The output for Step 2 is shown in Fig. 5, Fig. 6, and Table II

TABLE II
STEP 2: MERGING COMBINED VIDEO RECIPES INTO UNIVERSAL

# of Parsed FOONs from File 1	996
# of Parsed FOONs from File 2	52
Total # of Parsed FOONs	1048
Newly added FOONs	52

IV. SEARCHING

Finally for the last part of Project 2 we were asked to create an algorithm that, given a list of goal nodes, kitchen items, and a graph to search in, could construct the subgraph required to create each goal item.

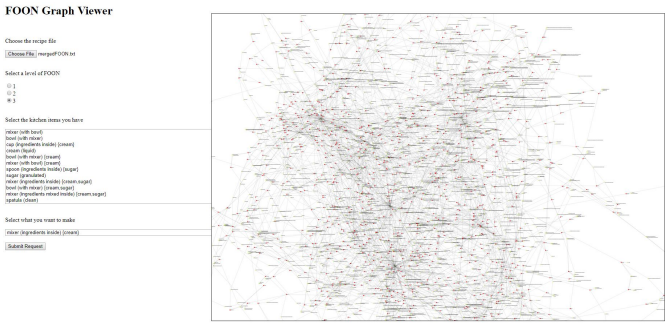

Fig. 5. The graph visualization of step 1 merging

```
(venv) amartin@amartin:~/code/mergeFOON$ python src/mergeFOON.py --file1 resource/FOON_22.6.2018.txt --file2 mergedSubgraph.txt --name mergedFOON.txt
# of parsed FOONs from File resource/FOON_22.6.2018.txt = 996
# of parsed FOONs from File mergedSubgraph.txt = 52
# of total parsed FOONs = 1048
# of newly added FOONs = 52
# of duplicate FOONs = 0
All Functional Units merged written to mergedFOON.txt
New Functional Units added written to mergedFOON_added.txt
```

Fig. 6. The terminal output of step 2 merging

For this part of the project I used the parsing aspect from the previous part to parse out the main graphs functional units. After that I modified the parsing function to parse out the kitchen and goal item files as they didn't include motions. In doing so I encountered issues where the goal items didn't have items listed even though they may have items associated with them in the main graph. Because of this, I decided to edit my goal file and add the items to the goal nodes.

Once I have parsed out the goal node/kitchen nodes and the main graph file I begin a search loop for every goal node in my list of nodes. I start by adding the goal node to a queue. Then, while the queue is not empty I dequeue the first node and find every functional unit that has this node in it's output. Then for every functional unit that I found I see if I can make it with my current list of kitchen items. If I can then I add the unit to the goal tree and the outputs to my kitchen list. If any of the outputs were the goal then I break from the loop and return the tree, if not then I continue the loop. However, if I couldn't make the desired functional unit then I return the missing items and add them to my queue.

This main loop continues while there are items in the queue. To prevent the loop from continuing forever I added logic to track whether new kitchen items have been added and only consider functional units again if there has been new items added since the last time I considered them.

For this step I used the goal and kitchen set 16 and I managed to find 4 of the 10 goal items. These items are shown in Table III

TABLE III
FOUND SEARCHING RESULTS

Goal Item	Length of Tree	Figure
Fried Bacon	5	Fig. 7
Beaten Egg	2	Fig. 8
Seasoned Ribs	7	Fig. 9
Chopped Scallions	1	Fig. 10

I was unable to find trees for 6 of the 10 items however. These items are shown in Table IV

TABLE IV
MISSING SEARCHING RESULTS

Goal Item	Missing Items Count
Plate Pancake	9
Boiling Water	3
Hash Browns	9
Garlic Rice	18
Chipotle-Avocado Sandwich	16
Protein Shake	6

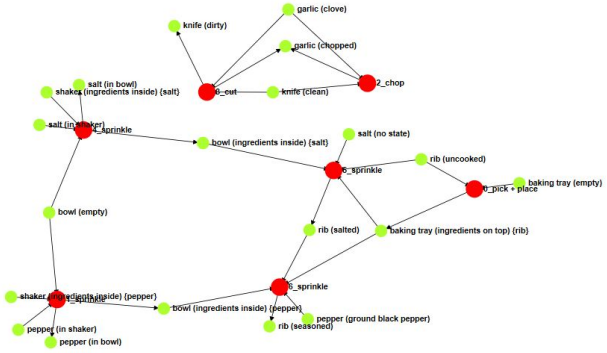

Fig. 9. The tree of goal node Seasoned Ribs

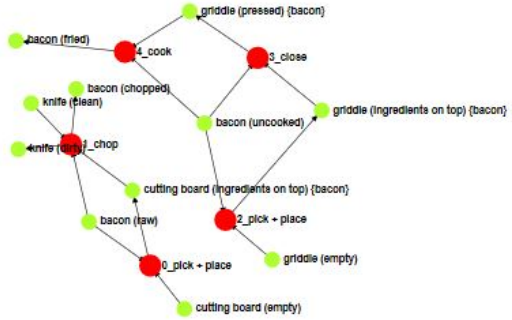

Fig. 7. The tree of goal node Fried Bacon

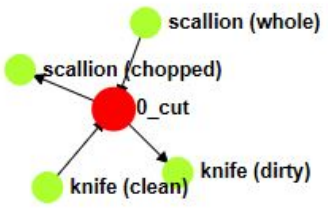

Fig. 10. The tree of goal node Chopped Scallions

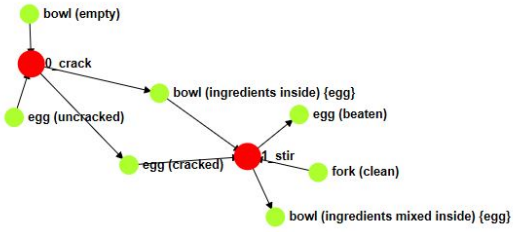

Fig. 8. The tree of goal node Beaten Egg

Fig. 11. The terminal output of merging two of the same file

TABLE V
TEST CASE 1: MERGE TWO OF SAME FILE

# of Parsed FOONs from File 1	26
# of Parsed FOONs from File 2	26
Total # of Parsed FOONs	52
Newly added FOONs	0

V. DISCUSSION

One thing that I thought was interesting at first was that there were no matches in either of my merging steps. This led me to at first believe that the merging wasn't working properly so I set out to create some test cases that would further test this merge. I will discuss some of these tests here.

My first test was to merge two of the same graph together. For this I just used one of my recipe graphs that I made, in this case the Cinnamon-Apple Cake. The outputs can be seen in Fig. 11 and Table V

Finally, I tested merging randomly picked existing sub-graphs back in to the universal graph. Since they are already existing in the universal graph none of the functional units should be added. The outputs can be seen in Fig. 12 and

Table VI

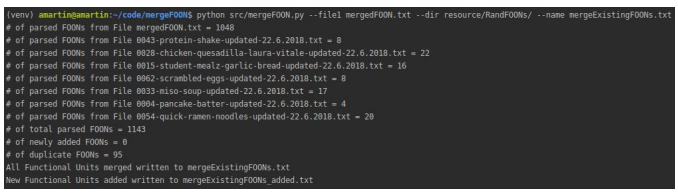

Fig. 12. The terminal output of merging a bunch of random existing subgraphs into the universal graph

TABLE VI
TEST CASE 2: MERGE EXISTING INTO UNIVERSAL

# of Parsed FOONs from File 1	1048
# of Parsed FOONs from File 2	8
# of Parsed FOONs from File 2	22
# of Parsed FOONs from File 2	16
# of Parsed FOONs from File 2	8
# of Parsed FOONs from File 2	17
# of Parsed FOONs from File 2	4
# of Parsed FOONs from File 2	20
Total # of Parsed FOONs	1143
Newly added FOONs	0
Number of duplicate FOONs	95

As can be seen through the two test cases, the merging is working and properly detects duplicates. This made me realize how many unique functional units there are and how difficult it is to find a match. This is because in order for a functional unit to match it had to have every start object/state/item, motion, and end object/state/item match up. This is very unlikely and thus in most cases, the new recipes were all unique.

In order to troubleshoot my searching, I began to step through and debug the searching algorithm only to discover a few inconsistencies that I think may be causing the search to not find as many matches as it could. As mentioned before I made the goal nodes have items because I think in order for something to be determined if it can be made it needs to match the items of the other unit exactly. This meant that for 2 of the goal items I didn't get matches because I was strictly comparing items too.

Additionally, I had to add a hard stop on searching as the loop could potentially get stuck forever because of nodes that appear in both the output and input of many places. Things like empty bowls are in a lot of functional unit and thus if the goal item requires an empty bowl the algorithm may go down a very distant path and start adding all sorts of random items to the missing items list because it wants to make a empty bowl. I think if the universal graph had some object/states that were guaranteed to be hard-stop (i.e. appear in no outputs) then I think the search would produce more accurate missing items and not be as prone to get stuck in an endless loop.

Finally, the biggest concern of mine when it comes to working with the universal graph is how much variability there is and how subjective creating these subgraphs are. Depending on who was the one who created the subgraph could drastically effect the overall standardization of the graph. This can be seen in some of the object/states where there are a lot of object/states that seem to be essentially synonyms of each other. This causes issues on the grand scale because then units are not matching up when it seems as though they logically should. Take for example the object ladle; it has some units that go *clean ladle* in to *dirty ladle* out (lines 1999-2010), some that go just *clean ladle* to nothing in the output (lines 2413-2422), and finally some that go *no state ladle* to no ladle mentioned in the output (lines 9165-9176). I think logically no state shouldn't be an option as the ladle should be either clean or dirty and additionally I think that if an object appears in the premotion section then it should also appear in the output section in a different, but related state, (i.e. clean – dirty). I think if the amount of occurrences of these issues were lowered then there would be a lot more matches between functional units and a theoretical robot would have an easier time transitioning along the different units to get to the final goal node.

ACKNOWLEDGMENTS

Thanks to the Professor Yu Sun and the T.A. David Paulius for the help with the project/understanding the functional object oriented notation language and for writing the following papers on functional units for me to look at for even more additional information [1], [2], [3], [4], [5], [6], [7].

REFERENCES

[1] D. Paulius, A. B. Jelodar, and Y. Sun, "Functional object-oriented network: Construction expansion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5935–5941.

[2] D. Paulius and Y. Sun, "A survey of knowledge representation and retrieval for learning in service robotics," *arXiv preprint arXiv:1807.02192*, 2018.

[3] A. B. Jelodar, D. Paulius, and Y. Sun, "Long activity video understanding using functional object-oriented network," *arXiv preprint arXiv:1807.00983*, 2018.

[4] D. Paulius, Y. Huang, R. Milton, W. D. Buchanan, J. Sam, and Y. Sun, "Functional object-oriented network for manipulation learning," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2655–2662.

[5] Y. Sun and Y. Lin, "Modeling paired objects and their interaction," in *New Development in Robot Vision*. Springer, 2015, pp. 73–87.

[6] Y. Sun, S. Ren, and Y. Lin, "Object–object interaction affordance learning," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 487–496, 2014.

[7] S. Ren and Y. Sun, "Human-object-object-interaction affordance," in *2013 IEEE Workshop on Robot Vision (WORV)*. IEEE, 2013, pp. 1–6.