Nelson Batista, Max Inciong, and Francesca Truncale

Senior Project II — Fall 2017

Professor Jianting Zhang

Report for Oct. 10

This week, we completed almost the entirety of the CPU implementation. The only remaining tweak is to modify the `find_isomorphism` function to return the *number* of matches of the subgraph in the larger graph, rather than simply whether there is a match. We are currently debugging this particular modification, and it should be complete by the middle of this week.

We have looked into timing considerations, and used Python's built-in profiling tools to analyze the performance of the CPU code. On the Facebook graph we are using, finding a single instance of subgraph isomorphism takes around 12-15 seconds. Below is the result of one sample run.

```
Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000   15.085   15.085 <string>:1(<module>)
   492    0.000    0.000    0.001    0.000 graph.py:22(<lambda>)
117612   13.864    0.000   13.864    0.000 graph.py:69(has_edge)
     1    0.000    0.000   15.085   15.085 isomorphism.py:43(find_isomorphism)
     1    1.218    1.218   15.085   15.085 isomorphism.py:63(update_possible_assignments)
     1    0.000    0.000   15.085   15.085 isomorphism.py:7(search)
   492    0.000    0.000    0.000    0.000 {len}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
   242    0.000    0.000    0.000    0.000 {method 'remove' of 'list' objects}
   488    0.002    0.000    0.002    0.000 {range}
```

Figure 1: Sample run of the CPU implementation of subgraph isomorphism.

We see that the overwhelming majority of the time is spent calling the `has_edge` method of the graph class. This method, shown below, calls the built-in `in` function of Python lists.

```python
def has_edge(self, vert1, vert2):
    """ Checks if edge conecting vert1 and vert2 is in the graph
    """
    return ({vert1, vert2} in self.adjacencies) #if adjacent, there's an edge
```

Listing 1: `has_edge` method of our graph class.

This method has a runtime complexity of O(n) (see the table at `https://wiki.python.org/moin/TimeComplexity`), and is called *117,612* times in the sample run. This is an obscene number of calls, and is adequately explained by looking at the `update_possible_assignments` function, called by `find_isomorphism`:

```python
any_changes = True
```

```python
while any_changes:
    any_changes = False
    for i in range(0, subgraph.n_vertices()):
        for j in possible_assignments[i]:
            for adj in subgraph.adjacencies(i):
                match = False
                for vert in range(0, graph.n_vertices()):
                    # graph.has_edge gets called once for every vertex in the graph
                    # for every item in the subgraph's adjacencies
                    # for every possible assignments
                    # that is a huge number of calls to has_edge
                    # which is in itself an O(n) operation
                    # definitely room for improvement.
                    if adj in possible_assignments[adj] and graph.has_edge(j, vert):
                        match = True
                if not match:
                    possible_assignments[i].remove(j)
                    any_changes = True
```

Listing 2: Contents of the update_possible_assignments function.

We see that has_edge is called once per each iteration of the innermost loop, which runs for as many iterations as the graph has vertices (which is many if the graph is large, as ours is), which is itself contained within another for loop with many iterations. Suffice to say, the algorithm's performance would benefit greatly from being able to parallelize this particular loop, since it gets called so many times.