NemoProfile Network Motif Analysis Parallelization with MASS Library

CSS499 Term Report, Spring 2016

Drew Andersen

Introduction

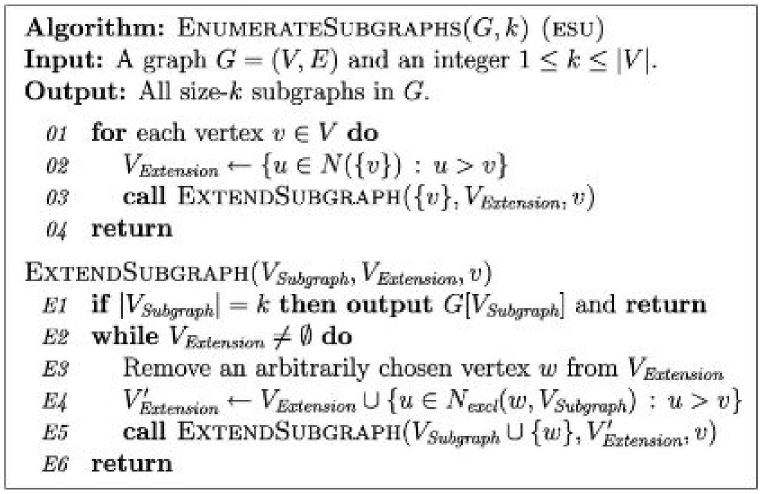
This quarter I began work on adding a feature called NemoProfile to a series of parallel big data programs called Bionet, developed by former student Matthew Kipps. The NemoProfile feature adds additional, relevant information to the programs' output, but with added computational and memory costs. My goal was to efficiently implement the feature and measure these penalties.

Background

Recent advances in technology have led to an increase in the amount of data available to researchers in many fields, including biology. Some of this information, such as protein-protein interactions, is best presented in a network or graph data structure. Researchers often analyze these biological networks by attempting to detect unique, recurring subgraph patterns called *network motifs*. Many algorithms and tools are available to aid in this process, including ESU, randESU, and AllCollect, but most tools detect and enumerate these motifs, but do not map the motifs to specific nodes. This mapping is valuable to researchers, but has traditionally be considered too expensive to collect.

Professor Wooyoung Kim and a former student developed a novel technique to efficiently map network motif instances to nodes. This technique uses the ESU algorithm (Figure 1) to detect and enumerate instances of all subgraphs of a given size in a target network. The resulting enumeration of subgraphs is then tested against a random pool of similar networks to determine which subgraph patterns appear more frequently in the target network than in a typical network (Figure 2). The resulting patterns are considered network motifs. By retaining the mapping of nodes to subgraphs, the NemoProfile technique can also allow for efficient collection of all instances of network motifs by executing ESU on an induced

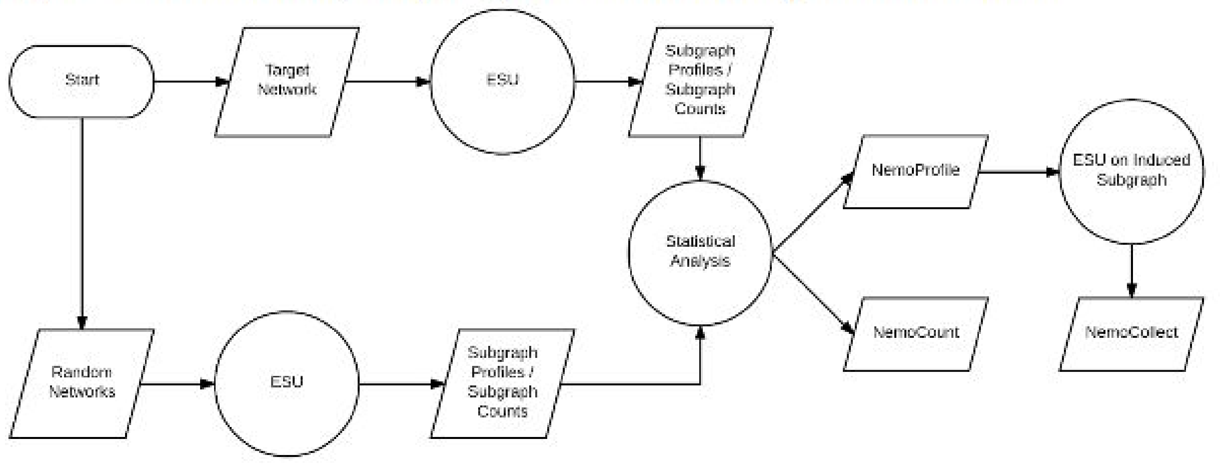
Figure 1: Pseudocode of the ESU algorithm utilized by the NemoProfile technique.



```
Algorithm: EnumerateSubgraphs($G, k$) (ESU)
Input: A graph $G = (V, E)$ and an integer $1 \leq k \leq |V|$.
Output: All size-$k$ subgraphs in $G$.
  01 for each vertex $v \in V$ do
  02     $V_{Extension} \leftarrow \{u \in N(\{v\}) : u > v\}$
  03     call ExtendSubgraph($\{v\}, V_{Extension}, v$)
  04 return
ExtendSubgraph($V_{Subgraph}, V_{Extension}, v$)
  E1 if $|V_{Subgraph}| = k$ then output $G[V_{Subgraph}]$ and return
  E2 while $V_{Extension} \neq \emptyset$ do
  E3     Remove an arbitrarily chosen vertex $w$ from $V_{Extension}$
  E4     $V'_{Extension} \leftarrow V_{Extension} \cup \{u \in N_{excl}(w, V_{Subgraph}) : u > v\}$
  E5     call ExtendSubgraph($V_{Subgraph} \cup \{w\}, V'_{Extension}, v$)
  E6 return
```

subgraph consisting of the nodes belonging to network motifs, as identified through NemoProfile. This process of network motif collection is called NemoCollect.

Figure 2: A flowchart depicting the processes of *NemoProfile* and *NemoCollect*.



Problem

While NemoProfile adds minimal overhead to a sequential implementation of an ESU-based implementation of the network motif finding process, there is potential for additional overhead when parallelizing the NemoProfile technique. The additional overhead steps primarily from three sources:

1. Merging of node-motif frequency mappings. In a sequential, shared-memory model, instances are enumerated in a single map. However, in a parallel, distributed-memory model, the instances are enumerated in *n* maps, where *n* is the number of computing nodes. When these maps are collected, they must be sequentially combined at the master node.
2. The amount of data to transfer between nodes is higher, leading to larger latency.
3. The amount of data stored at each node is higher, leading to higher memory usage.

Our goal is to determine how much additional overhead is added to a typical ESU-based parallel network motif detection program compared to such a program with NemoProfile and NemoCollect implemented.

The programs created by Kipps (called Bionet) parallelize the ESU algorithm through three different methods: MPI, MASS agents-based, and MASS places-based. He also implemented a sequential version of the algorithm for comparison. These implementations enumerated subgraphs for a target network, but did not map subgraphs to nodes or perform the comparison against random networks to detect network motifs. Also, the agents-based implementation suffers from severe memory overhead issues for larger network and motif sizes, to the point that remoted nodes run out of memory and throw exceptions.

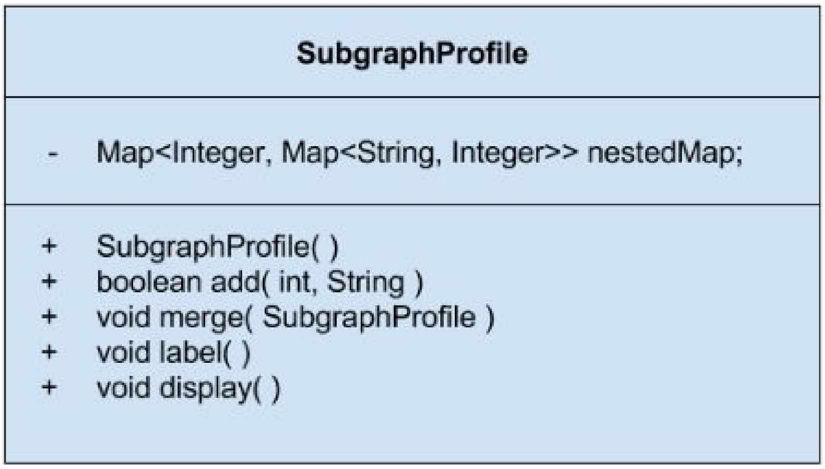
Implementation

In the original implementations of Bionet, the subgraph pattern enumerations are stored in hash maps. I considered three options to replace this structure:

1. Two-dimensional matrix array. While this implementation would have been the easiest to implement, it would also lead to relatively high memory complexity because there is no way to determine how large the matrix would need to be at each node.
2. Two-tuples. Under this implementation, the two-tuples would essentially perform a naive NemoCollect. I originally attempted this implementation due to a misunderstanding of how NemoProfile functions, but soon discovered my mistake as evidenced by prohibitively expensive overhead.
3. Nested hashmaps. This was the version we ultimately chose. The outer hashmap maps vertices to maps like the ones utilized in the original implementation, mapping patterns to frequencies of occurrence.

I designed and implemented a new class called SubgraphProfile to manage the complexity of the new data structure (Figure 3). The class was designed to be used across all four implementations of Bionet (with minor modifications necessary for MPI). In addition to moving the complexity of the add and merge processes into a class, I also moved the label and display processes into this class. The final design turned out to be rather simple, but was the result of a great deal of trial and error.

Figure 3: The SubgraphProfile class, created to abstract away the complexity of the data structure being implemented as part of this project.



```
SubgraphProfile

-   Map<Integer, Map<String, Integer>> nestedMap;

+   SubgraphProfile( )
+   boolean add( int, String )
+   void merge( SubgraphProfile )
+   void label( )
+   void display( )
```

Results

I am still in the process of gathering experimental data. So far, the overhead appears to be significant, due almost entirely to the merging process. Surprisingly, this is true even for the sequential portion of the process, which would seem to contradict the results of the unpublished paper by Kim and Haukup.

Future Work

My next step is to finish gathering and analyzing the experimental data. Following that, I will re-examine the sequential NemoProfile implementation created by Kim and Haukup to see if there are any significant differences between their implementation and mine. Next, I will implement the remaining elements of NemoProfile and NemoCollect into Bionet and report the results. Finally, I plan to begin work on a production version of NemoProfile/NemoCollect that could be released to the public.