Rakesh Gopal Kavodkar
rgopalk
200066020

NOTE: The source code for the three problems is included as a zip file in the attachments.

Problem 1.

1. To run the profiler, we need to compile the source code with the –pg option, so that it generates the `gmon.out` file once we run the program. To run the profiler, we need to use the command `gprof ./executable` . This will give us the following result if piped to `less`.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 39.17      0.09     0.09                             main
 26.11      0.15     0.06  5975248     0.00     0.00  mapreduce::Worker::enter_into_map(long)
 13.06      0.18     0.03        1    30.03    30.03  mapreduce::perform_reducing(mapreduce::Worker**,
  8.70      0.20     0.02  2987624     0.00     0.00  mapreduce::Worker::add_to_vertices(long, long)
  8.70      0.22     0.02        8     2.50    10.01  mapreduce::Worker::perform_mapping()
  4.35      0.23     0.01        8     1.25     1.25  mapreduce::Worker::Worker(long, long)
  0.00      0.23     0.00       16     0.00     0.00  __gnu_cxx::__normal_iterator<std::string*, std::
  0.00      0.23     0.00       16     0.00     0.00  std::vector<std::string, std::allocator<std::str
  0.00      0.23     0.00       14     0.00     0.00  std::_Vector_base<std::string, std::allocator<st
```

   a. The above result corresponds to the `youtube.graph.original` file which is attached in the moodle page.
      Here we see that the mapping function, viz, `perform_mapping()` which in turn calls `enter_into_map(..)` take 26.11% and 8.70% of the total time.
      The reducing process takes about 13% of the execution time `perform_reducing(..)`
      The majority of the execution time is taken is the main function. The main function has lines of code that writes the output into a file (about 30 million lines for the above mentioned graph).
      The cost of IO here is almost comparable to the cost of the actual computation.
      Note that `add_to_vertices(..)` populates each of the *worker nodes*.
      There is no function that takes up more than 80% of the computational time.

Problem 2.

1. We see that the Hadoop program takes more time that the C/C++ implementation.
   This can be attributed to a couple of reasons.
   - Hadoop writes the intermediate results to the HDFS file system. It also has a dedicated logging service that logs each and every task that is undertaken. File I/O is a considerably costly operation. In the C/C++ implementation, there are no intermediate files written, nor are there any logs.
   - Even though the Hadoop ran on a single node, the implementation of the framework is for a distributed computation, and hence, it has a lot of different components like the JobTracker, TaskTracker, NameNode, DataNode, etc. Each of these components runs as a separate service. And even though the master and slave are both on the same machine and it uses the `localhost`

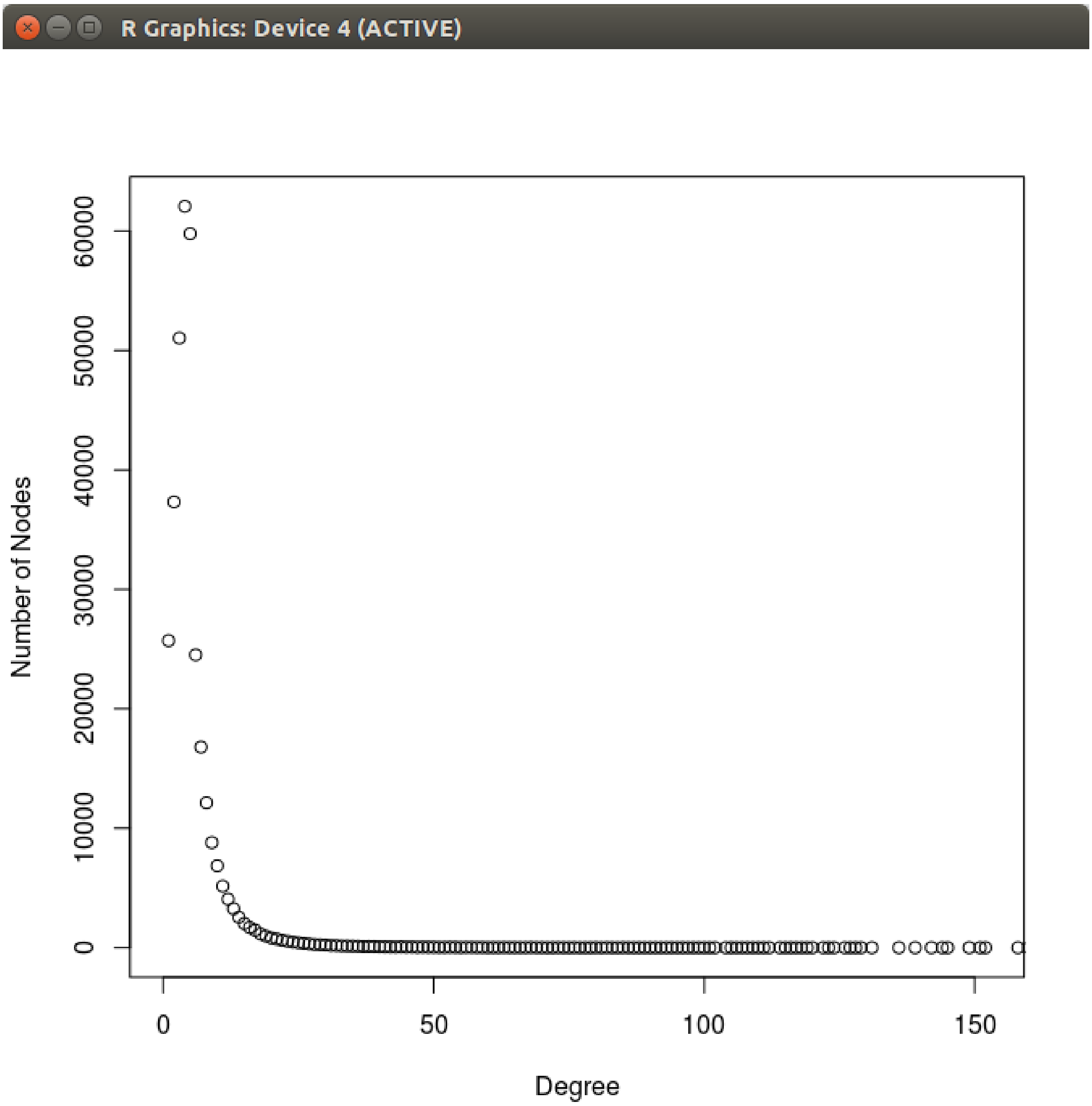
address, a lot of extra computation is done to keep track of the jobs (equivalent to what would be done in a multi node scenario) and to hand over the baton to the next one in line.

- The C/C++ implementation is concise and to the point. It does exactly the vertex degree counting, unlike Hadoop which is a big framework to which we add our code, where the execution moves about through different components in different phases of the computation.
- C/C++ is way faster than Java

Problem 3.1.

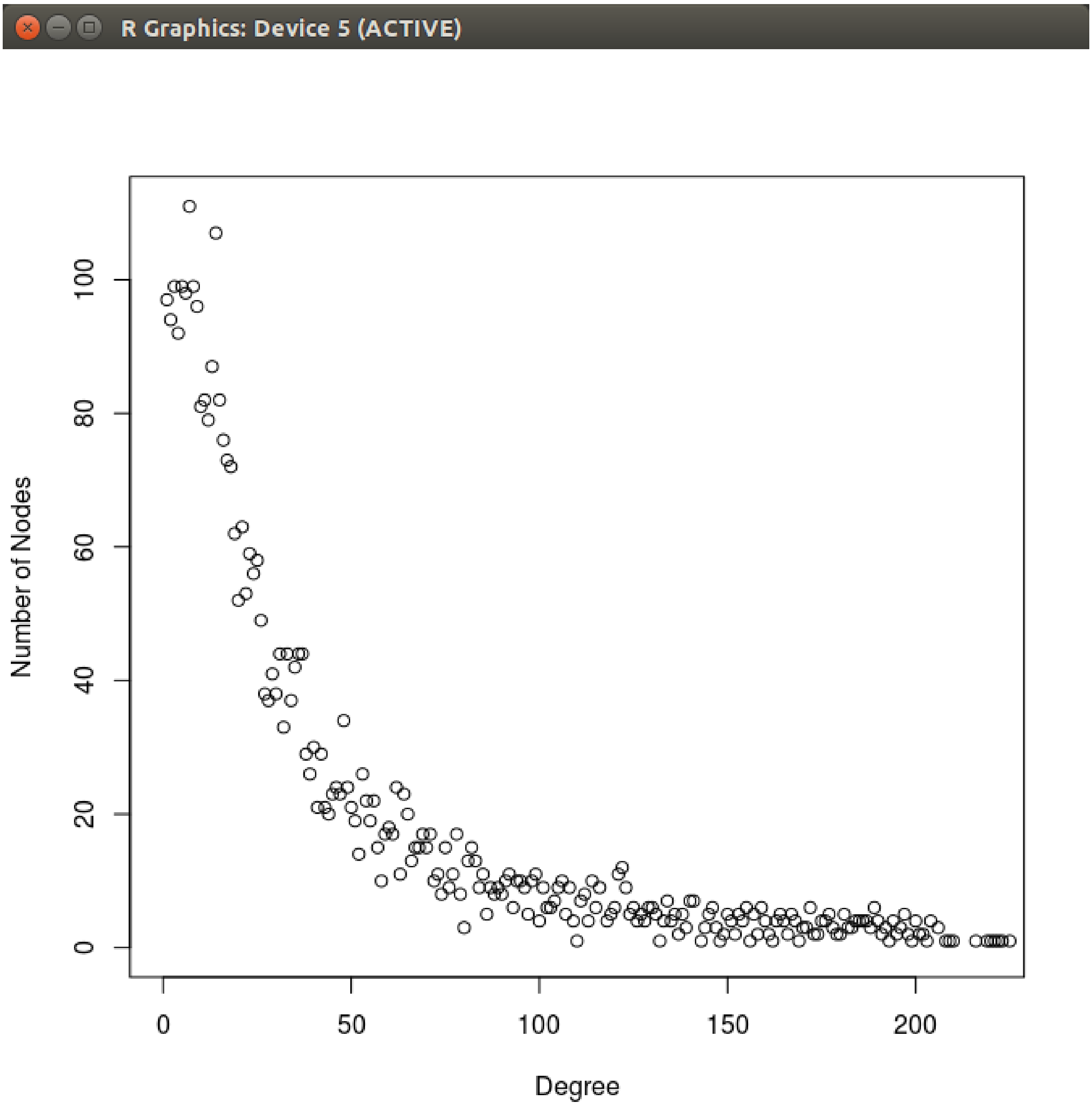
Plotting the degree versus the number of nodes, we get the following graphs for the below mentioned datasets

1. Amazon:



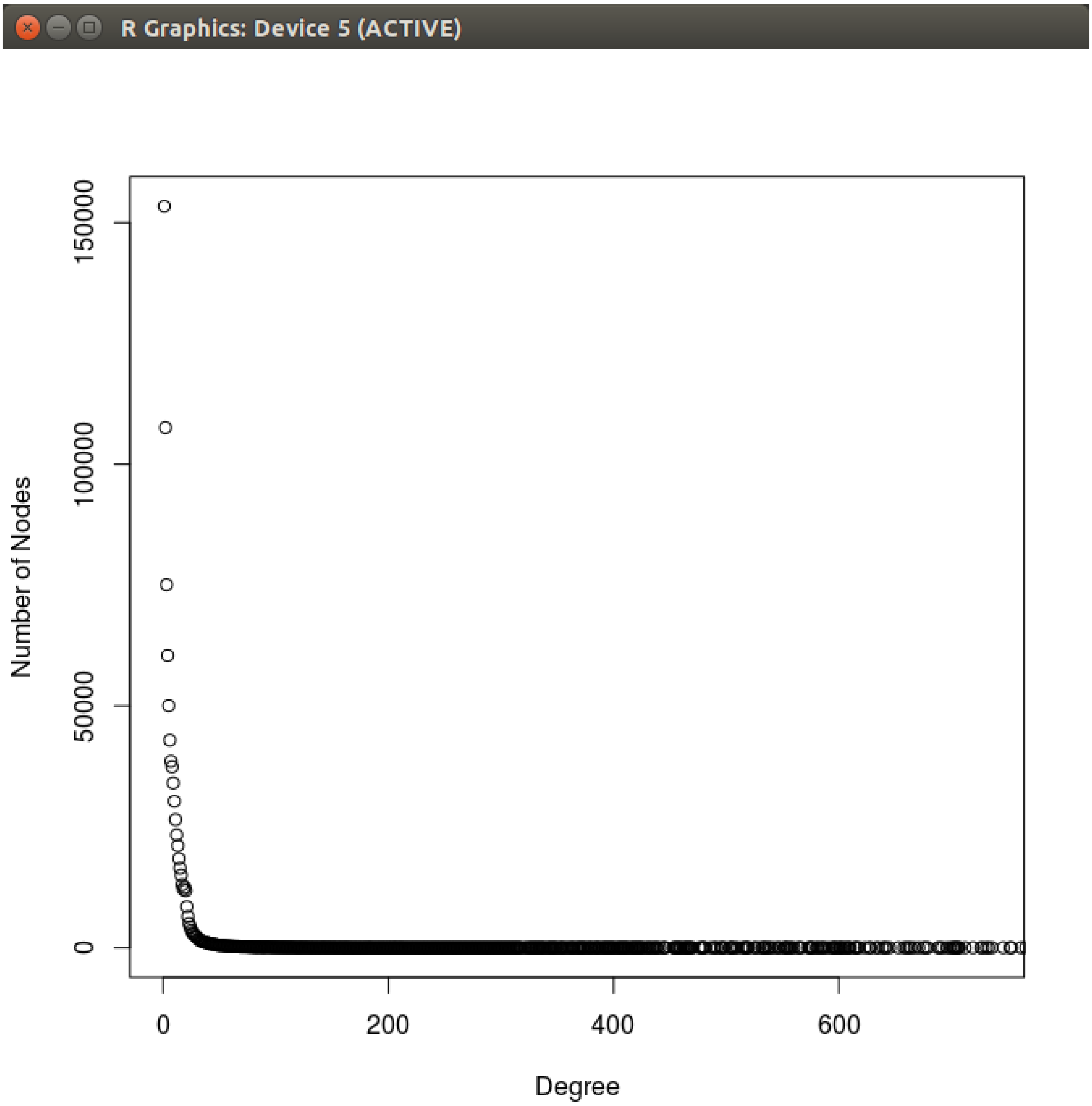
We see that this graph follows the power-law curve, and hence this is a **scale-free graph**

2. Facebook



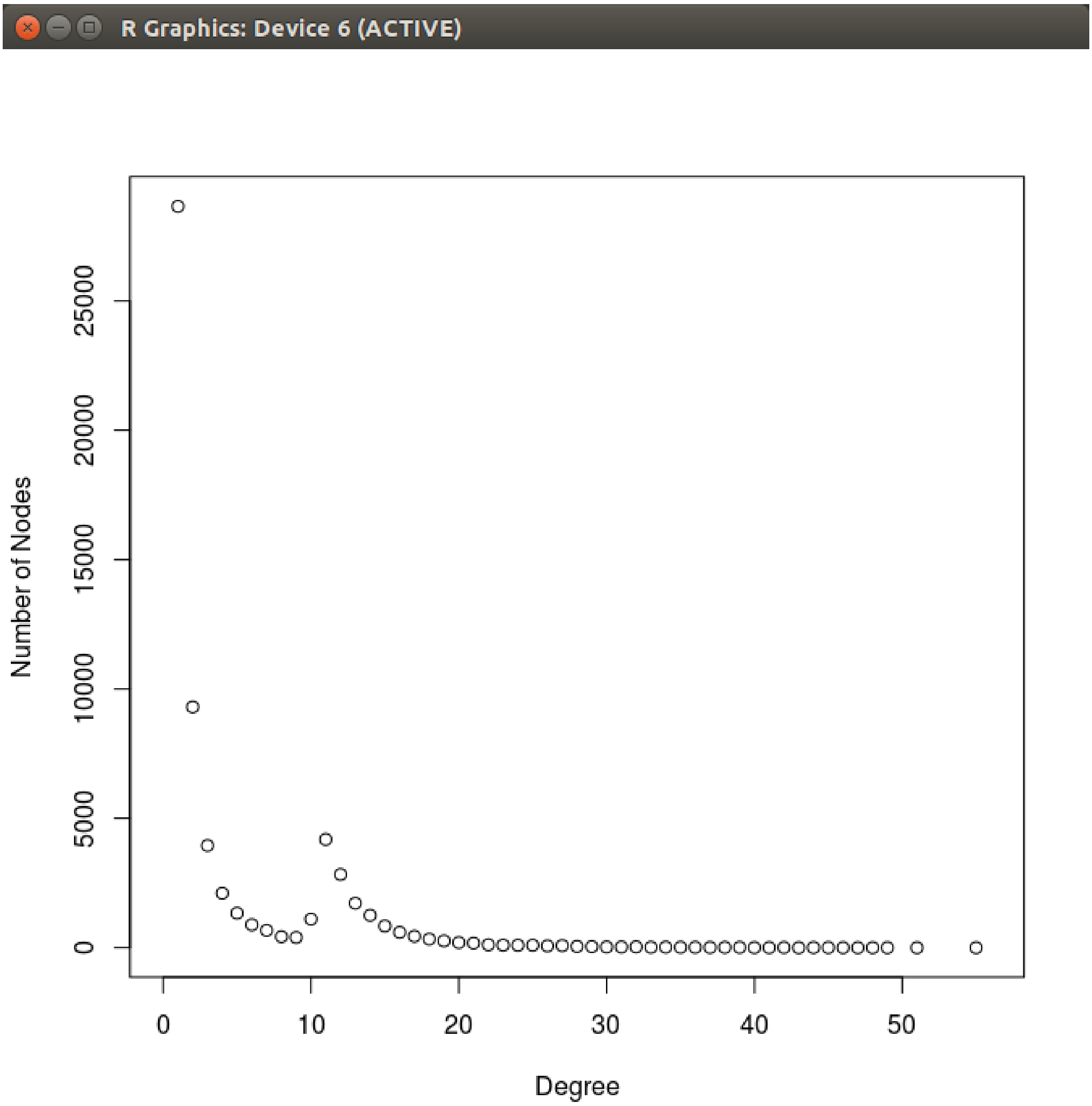
Although the points are a bit scattered, they seem to follow the power-law curve. Hence this is a **scale-free graph**

3. **Google**



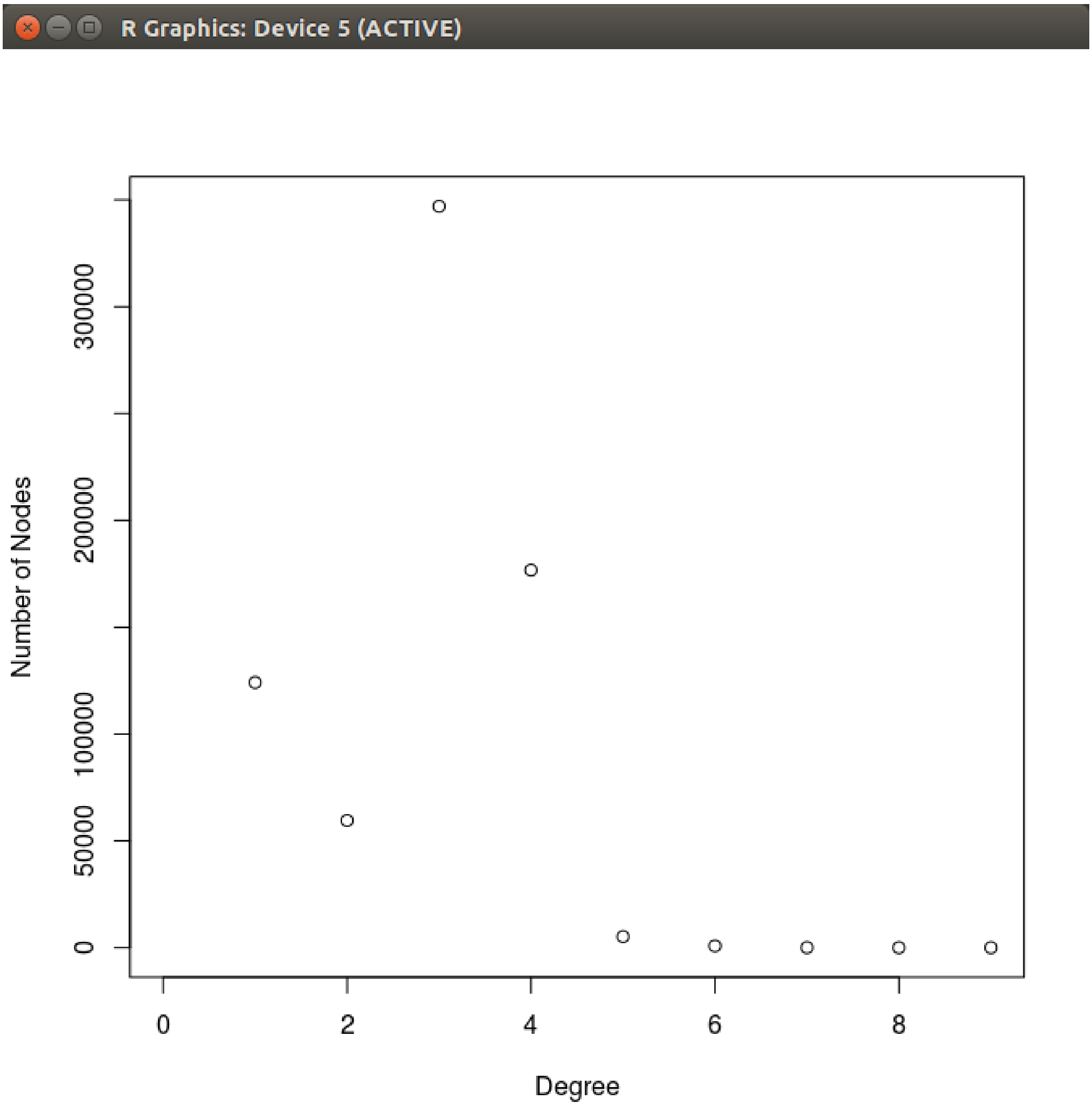
Google’s dataset also follows the power-law curve. It is a **scale-free** graph

4. p2p-Gnutella31



This graph has a few of outliers with respect to the power-law curve, however since it does maintain the shape; this **could** qualify for being a **scale free graph**.

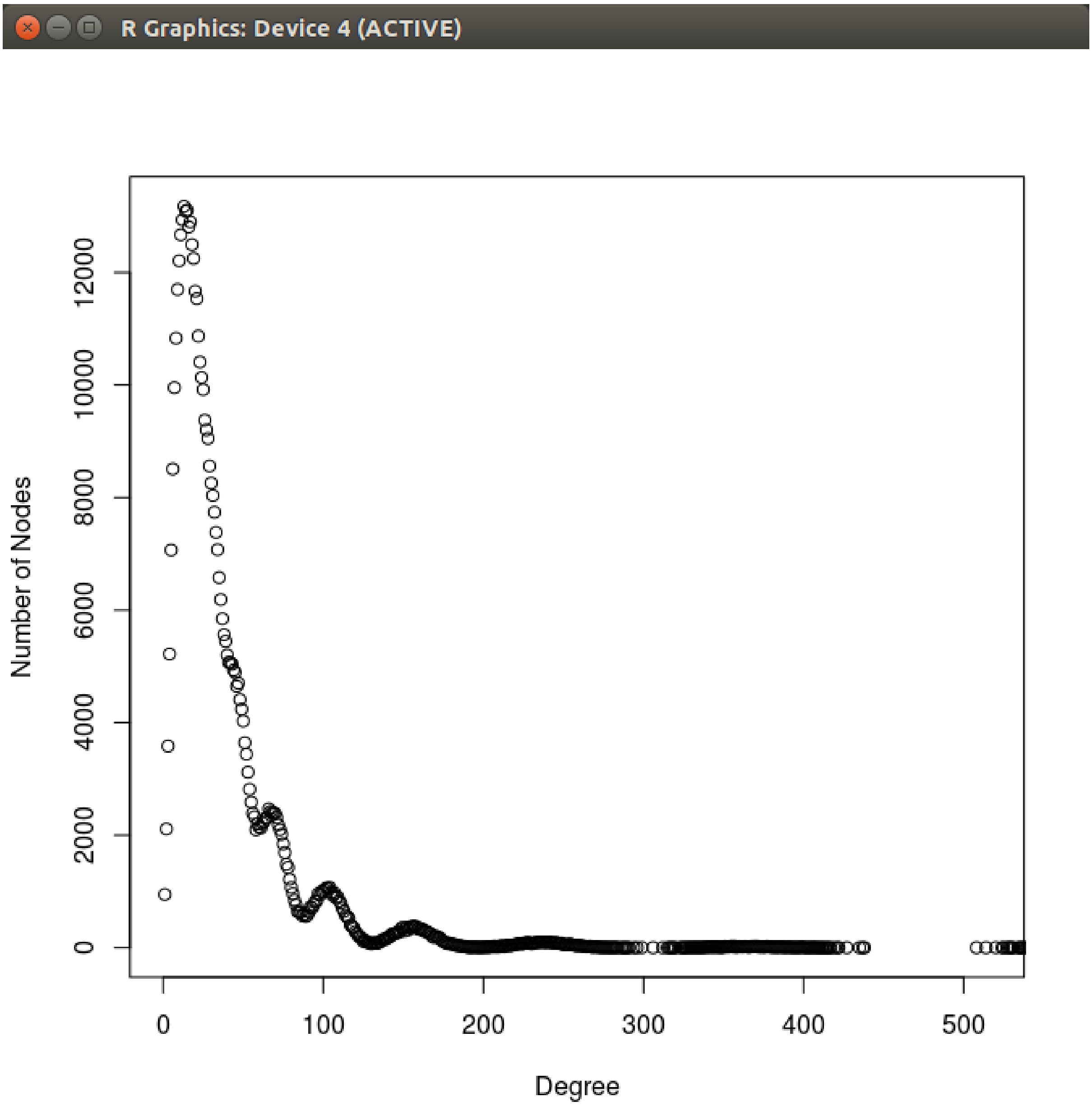
5. roadNet-PA



Although the points to the right are all low compared to the ones on the left, there are quite a lot of points are outliers. But considering that it does follow the power-law curve in the shape, it **could be** called a **scale free graph.**

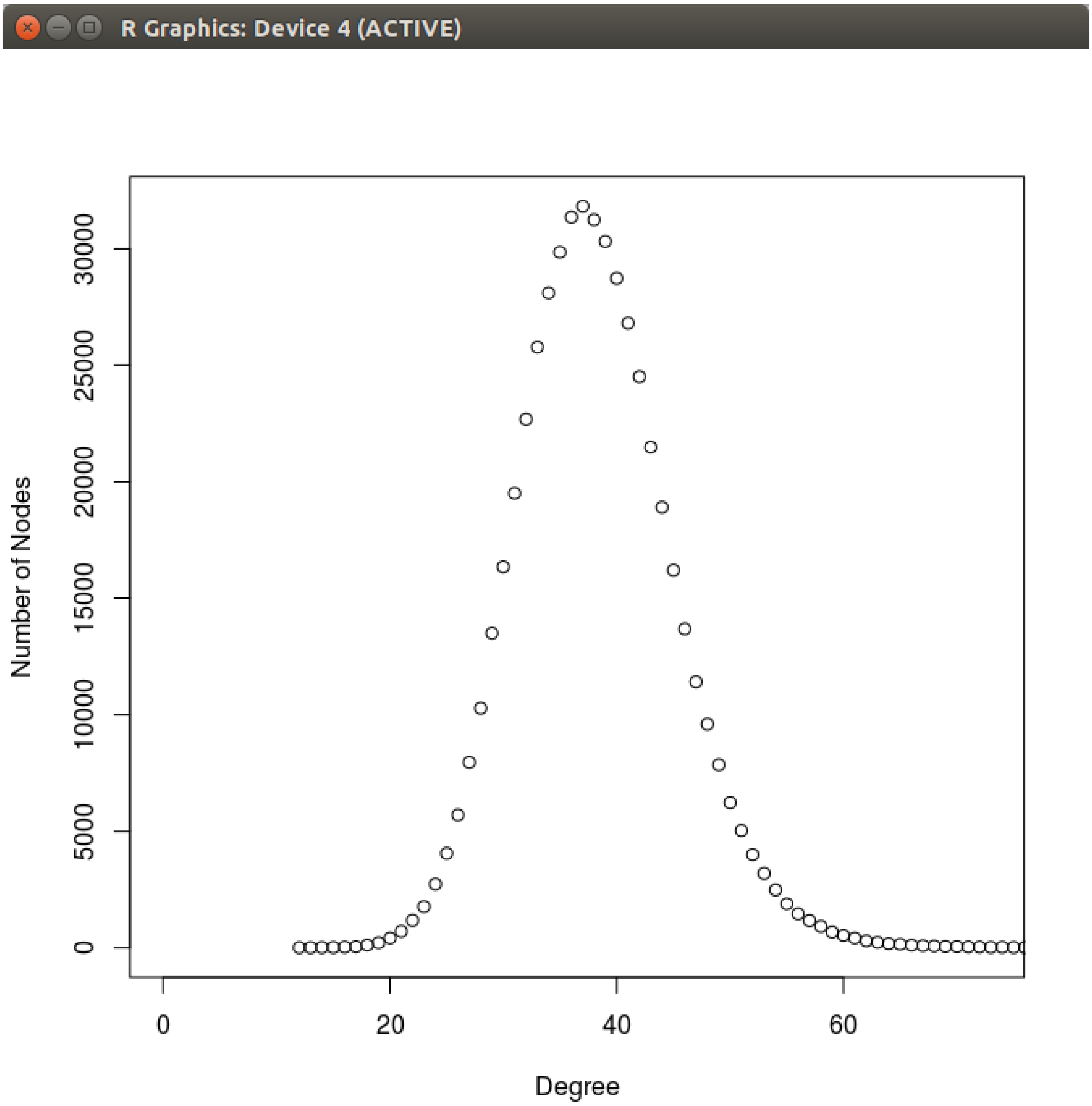
Problem 3.2.

1. Random graph 1



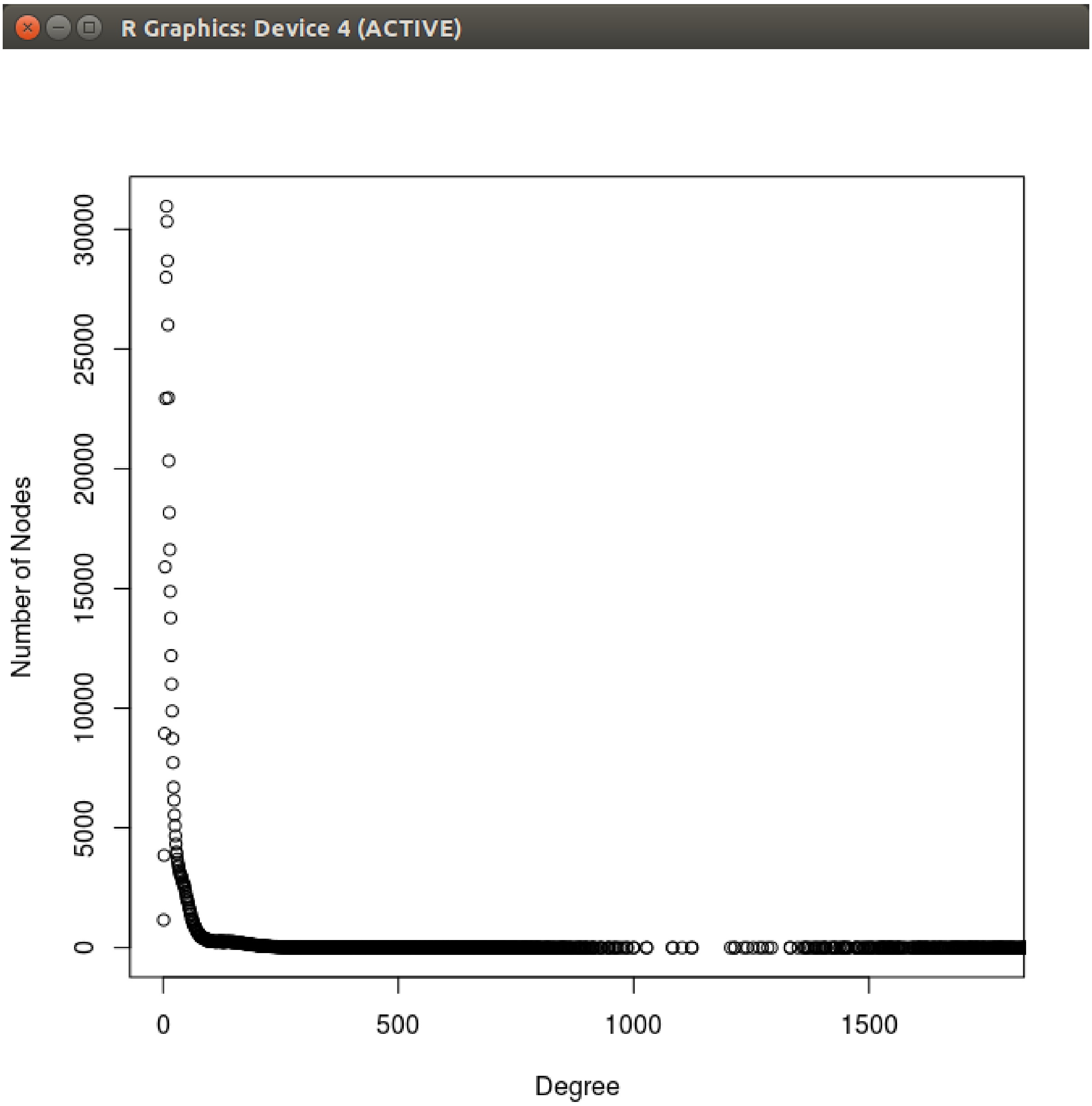
It does have the shape of the power-law curve, however considering the points to the extreme left which are a lot in number; this **cannot** be called a **scale-free graph**

2. Random graph 2



This clearly is not a **scale-free graph.** The shape corresponds to a bell curve

3. Random graph 3



This follows the power-law curve is definitely a **scale-free graph.**