Application 1: Analysis of Citation Graphs
Algorithmic Thinking (Part1)

Shamsuddin Rehmani

July 17, 2019

Application 1 Description

In the Module 1 Application, we will combine the mathematical analysis that we began in the Homework with the code that you have written in the Project to analyze a real-world problem: How do scientific papers get cited? This part of the module will probably be much more unstructured than you are accustomed to in an on-line class. Our goal is to provide a more realistic simulation of how the concepts that you are learning are actually used in practice. Your key task in this part of the module is to think about the problem at hand as you answer each question.

Citation graphs

Our task for this application is to analyze the structure of graphs generated by citation patterns from scientific papers. Each scientific paper cites many other papers, say 20-40, and sometimes (e.g., review papers) hundreds of other papers. But, let's face it: It is often the case that the authors of a paper are superficially familiar with some (many?) of the papers they cite. So, the question is: Are the cited papers chosen randomly (from within the domain of the paper) or is there some "hidden pattern"?

Given that we will be looking at "paper i cites paper j" relationships, it makes sense to represent the citation data as a directed graph (a citation graph) in which the nodes correspond to papers, and there is an edge from node i to node j if the paper corresponding to node i cites the paper corresponding to node j. Since we're interested in understanding how papers get cited, we will analyze the in-degree distribution of a specific graph, and contrast it to those of graphs generated by two different random processes

Answer to Question 1

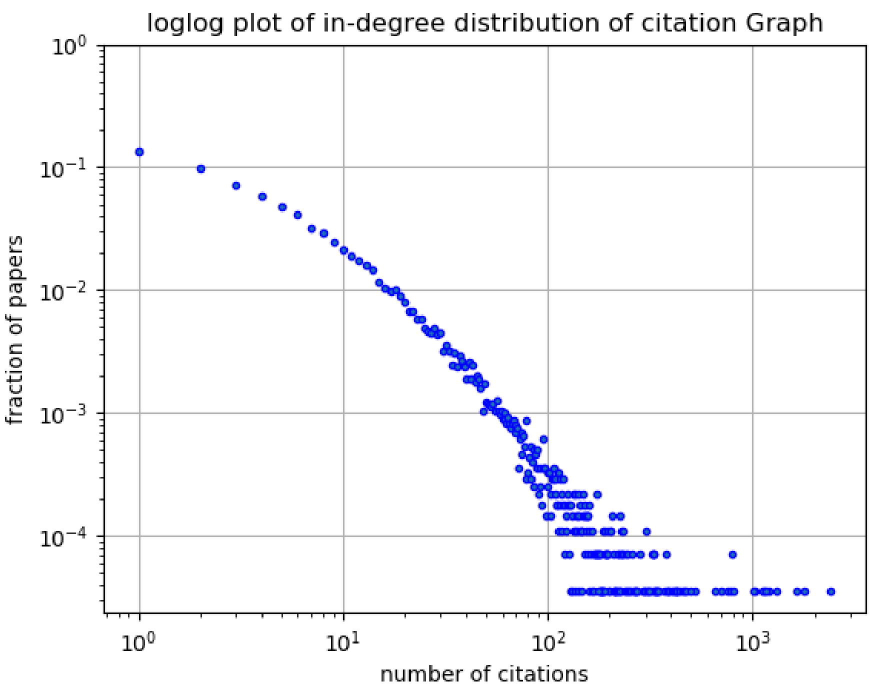

Figure 1: in-dgree distribution of citation graph

Answer to Question 2

Q2.1: Is the expected value of the in-degree the same for every node in an ER graph? Please answer yes or no and include a short explanation for your answer.

Ans: yes it is the same for all nodes since the presence of an edge is independent of all other edges i.e it is independent of the current structure of the graph The expected value of in-degree is given by $p(n-1)$

Q2.2: What does the in-degree distribution for an ER graph look like? Provide a short written description of the shape of the distribution.

Ans: we know that the probability that a given node has degree k is given by a binomial distribution as seen in the homework. Thus as $p \to 0$ (probability p becomes smaller), we see more nodes with smaller in-degree and thus the in-degree distribution shape looks like a bell curve skewed towards the left i.e near in-degree 0. As $p \to 1$, we get more nodes with higher in-degree and the shape is increasing curve with most points near the higher in-degree region. For large number of nodes, and small p, this becomes a symmetric bell shaped curve and approaches a normal distribution

Q2.3: Does the shape of the in-degree distribution plot for ER look similar to the shape of the in-degree distribution for the citation graph? Provide a short explanation of the similarities or differences. Focus on comparing the shape of the two plots as discussed in the class page on "Creating, formatting, and comparing plots".

Ans: As mentioned in answer of Q2.2, the shape for the ER in-degree approaches a bell-shaped curve for large N values and small p. However, for the citation graph it is a decreasing curve with majority of point located near in-degree of zero.

Answer to Question 3

Value of n and m that yield a DPA graph whose number of nodes and edges is roughly the same to those of the citation graph are as follows:

- $m = 27770$
- $m = 13$

Answer to Question 4

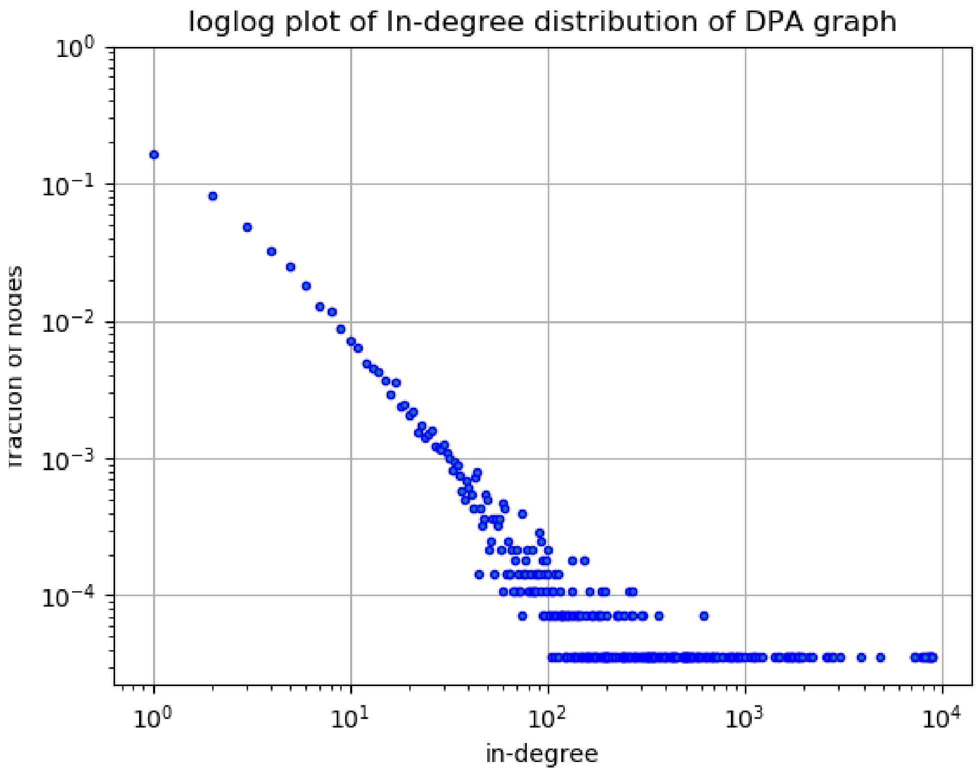

Figure 2: in-degree distribution for DPA graphs

Answer to Question 5

Q5.1: Is the plot of the in-degree distribution for the DPA graph similar to that of the citation graph? Provide a short explanation of the similarities or differences. Focus on the various properties of the two plots as discussed in the class page on "Creating, formatting, and comparing plots".

Ans: Yes they are similar since both follow a linear log-log decreasing trend i.e they both follow the power law distribution and the point are spread out more as the in-degree increases

Q5.2: Which one of the three social phenomena listed above mimics the behavior of the DPA process? Provide a short explanation for your answer.

Ans: DPA process mimics the "rich get richer" or the "preferential attachment" phenomena since every new node that is added to the graph is most likely to be connected to the neighbor with highest in-degree.

Q5.3: Could one of these phenomena explain the structure of the physics citation graph? Provide a short explanation for your answer.

Ans: The citation graph also mimics the "rich get richer" phenomena as the paper with higher citations i.e higher degree tend to be more likely used in other papers as well due to being more visible

A Python code used to answer the Application Questions

```python
"""
Analyze the structure of graphs generated by citation patterns from scientific papers
"""

import matplotlib.pyplot as plt
import numpy as np
import parse_graph
import alg_project1_solution as alg_proj1_sol
import alg_dpa_trial as dpa

##### Q1 Solution #####
# For this question, your task is to load a provided citation graph for 27,770
# high energy physics theory papers. This graph has 352,768 edges. You should
# use the following code to load the citation graph as a dictionary. In
# CodeSkulptor, loading the graph should take 5-10 seconds. (For an extra
# challenge, you are welcome to write your own function to create the citation
# graph by parsing this text representation of the citation graph.)
#
# Your task for this question is to compute the in-degree distribution for this
# citation graph. Once you have computed this distribution, you should normalize
# the distribution (make the values in the dictionary sum to one) and then
# compute a log/log plot of the points in this normalized distribution. How you
# create this point plot is up to you. You are welcome to use a package such as
# matplotlib for desktop Python, use the simpleplot module in CodeSkulptor, or
# use any other method that you wish

# load the graph from the text file
cit_graph = parse_graph.load_graph("citation_graph.txt")

# get the unnormalized in degree distribution
in_deg_dist = alg_proj1_sol.in_degree_distribution(cit_graph)

# normalize the in degree distribution
sum_val = sum(in_deg_dist.values())
in_deg_dist.update((degree, freq / float(sum_val)) for degree, freq in in_deg_dist.items())

# draw the loglog plot of the normalized in degree distribution of the citation graphe
plt.figure(0)
plt.loglog(in_deg_dist.keys(), in_deg_dist.values(), basex=10, basey=10, linestyle='None',
           marker='.', markeredgecolor='blue')
plt.title('loglog plot of in-degree distribution of citation Graph')
plt.xlabel('number of citations')
plt.ylabel('fraction of papers')
plt.grid()
plt.ylim(None, 1)
plt.show()
#plt.savefig("Q1_loglog_degree_dist_citgraph.png")

##### Q2 Solution #####
# In Homework 1, you saw Algorithm ER for generating random graphs and reasoned
# analytically about the properties of the ER graphs it generates. Consider the
# simple modification of the algorithm to generate random directed graphs: For
# every ordered pair of distinct nodes (i, j), the modified algorithm adds the
# directed edge from i to j with probability p.
#
# For this question, your task is to consider the shape of the in-degree
# distribution for an ER graph and compare its shape to that of the physics
# citation graph. In the homework, we considered the probability of a specific
# in-degree, k, for a single node. Now, we are interested in the in-degree
# distribution for the entire ER graph. To determine the shape of this
# distribution, you are welcome to compute several examples of in-degree
# distributions or determine the shape mathematically.
#
# Once you have determined the shape of the in-degree distributions for ER graphs,
# compare the shape of this distribution to the shape of the in-degree distribution
# for the citation graph. When answering this question, make sure to address the
# following points:
#
# Q2.1: Is the expected value of the in-degree the same for every node in an ER graph?
# Please answer yes or no and include a short explanation for your answer.

# Ans: yes it same for all nodes since the presence of an edge is independent of
# all other edges i.e it is independent of the current structure of the graph
# The expected value of in-degree is given by p * (n-1)

# Q2.2: What does the in-degree distribution for an ER graph look like?
# Provide a short written description of the shape of the distribution.

# Ans: we know that the probability that a given node has degree k is given by
# a binomial distribution as seen in the homework. Thus as p -> 0 (probability
# p becomes smaller), we see more nodes with smaller in-degree and thus
# the in-degree distribution shape looks like a bell curve skewed towards the
# left i.e near in-degree 0. As p -> 1, we get more nodes with higher in-degree
# and the shape is increasing curve with most points near the higher
# in-degree region. For large number of nodes, and small p, this becomes
# a symmetric bell shaped curve and approaches a normal distribution

# Q2.3: Does the shape of the in-degree distribution plot for ER look similar
# to the shape of the in-degree distribution for the citation graph?
# Provide a short explanation of the similarities or differences.
# Focus on comparing the shape of the two plots as discussed in the class page on
# "Creating, formatting, and comparing plots".

# Ans: As mentioned in answer of Q2.2, the shape for the ER in-degree approaches
```

```python
# a bell-shaped curve for large N values and small p. However, for the citation
# graph it is a decreasing curve with majority of point located near in-degree
# of zero.

##### Q3 Solution #####
# We next consider a different process for generating synthetic directed graphs.
# In this process, a random directed graph is generated iteratively, where in
# each iteration a new node is created, added to the graph, and connected to a
# subset of the existing nodes. This subset is chosen based on the in-degrees
# of the existing nodes. More formally, to generate a random directed graph in
# this process, the user must specify two parameters: nn, which is the final
# number of nodes, and m (where m <= n), which is the number of existing
# nodes to which a new node is connected during each iteration. Notice that m
# is fixed throughout the procedure.
#
# The algorithm starts by creating a complete directed graph on mm nodes.
# (Note, you've already written the code for this part in the Project.) Then,
# the algorithm grows the graph by adding n-m nodes, where each new node is
# connected to m nodes randomly chosen from the set of existing nodes. As an
# existing node may be chosen more than once in an iteration, we eliminate
# duplicates (to avoid parallel edges); hence, the new node may be connected
# to fewer than m existing nodes upon its addition.

# The algorithm is called Algorithm DPA (note that the m in the input is a
# parameter that is specified to this algorithm, and it does not denote the
# total number of edges in the resulting graph).

# For this question, we will choose values for n and m that yield a DPA
# graph whose number of nodes and edges is roughly the same to those of the
# citation graph. For the nodes, choosing n to be the number of nodes as
# the citation graph is easy. Since each step in the DPA algorithm adds m
# edges to the graph, a good choice for m is an integer that is close to
# the average out-degree of the physics citation graph.

# For this question, provide numerical values for n and m that you will
# use in your construction of the DPA graph.

# calculate n, i.e the number of nodes in citation graph for DPA algorithm
n_nodes = len(cit_graph.keys())
# calculate m, i.e the average out-degree in citation graph for DPA algorithm
m_nodes = int(round(np.mean([len(neighbors) for  neighbors in cit_graph.values()])))

print "Q3 Solution:"
print "n = ", n_nodes
print "m = ", m_nodes

##### Q4 Solution #####
# Your task for this question is to implement the DPA algorithm, compute a DPA
# graph using the values from Question 3, and then plot the in-degree distribution
# for this DPA graph. Creating an efficient implementation of the DPA algorithm
# from scratch is surprisingly tricky. The key issue in implementing the algorithm
# is to avoid iterating through every node in the graph when executing Line 6
# of the provided pseudocode. Using a loop to implement Line 6 leads to implementations
# that require on the order of 30 minutes in desktop Python to create a DPA graph with
# 28000 nodes.
#
# To avoid this bottleneck, you are welcome to use this provided code that implements
# a DPATrial class.
#
# Once you have created a DPA graph of the appropriate size, compute a (normalized)
# log/log plot of the points in the graph's in-degree distribution

# write the function for generating DPA graphs
def alg_dpa(n_num_nodes, m_num_nodes):
    """
    Uses the DPA algorithm provided in Q3 of the Application
    to generates a random directed graph iteratively, where
    each iteration a new node is created, added to the graph,
    and connected to the subset of the existing node

    Arguments:
        n_nodes {integer} -- final number of nodes in the generated graph
        m_nodes {integer} -- number of existing nodes to which a new node is connected
                             during each iteration

    Returns:
        dictionary -- the generated graph based on DPA algorithm
    """

    # create a complete graph of m_nodes noes
    graph = alg_proj1_sol.make_complete_graph(m_num_nodes)

    # create the DPA trial object corresponding to complete graph
    dpa_trial = dpa.DPATrial(m_num_nodes)

    # add each new ode to m_nodes from the existing graph randomly
    # chosen with probability:
    # (in-degree of new_node + 1) / (in-degree of all nodes +
    # total number of existing nodes)
    # simulated by the run_trial of the DPATrial class
    for new_node in range(m_num_nodes, n_num_nodes):
        # randomly select m_nodes from the existing graph that
        # the new_node will be connected to. Remove if any
        # duplicate nodes in the m_nodes selected
        new_node_neighbors = dpa_trial.run_trial(m_num_nodes)

```

```
        # update the existing graph to add this new node and its
        # neighbors
        graph[new_node] = new_node_neighbors


    return graph

# create the graph using the DPA algorithm
dpa_graph = alg_dpa(n_nodes, m_nodes)

# get the in-degree distribution for the DPA graph
in_deg_dist_dpa = alg_proj1_sol.in_degree_distribution(dpa_graph)

# normalize the in degree distribution for the DPA graph
sum_val = sum(in_deg_dist_dpa.values())
in_deg_dist_dpa.update((degree, freq / float(sum_val)) for degree, freq in in_deg_dist_dpa.items())

# draw the loglog plot of the normalized in-degree distribution of the DPA graph
plt.figure(1)
plt.loglog(in_deg_dist_dpa.keys(), in_deg_dist_dpa.values(), basex=10, basey=10,
           linestyle='None', marker='.', markeredgecolor='blue')
plt.title('loglog plot of In-degree distribution of DPA graph')
plt.xlabel('in-degree')
plt.ylabel('fraction of nodes')
plt.ylim(None, 1)
plt.grid()
plt.show()
#plt.savefig("Q4_loglog_indegree_dist_dpa.png")

##### Q5 Solution #####
# In this last problem, we will compare the in-degree distribution for the citation graph
# to the in-degree distribution for the DPA graph as constructed in Question 4. In
# particular, we will consider whether the shape of these two distributions are similar
# and, if they are similar, what might be the cause of the similarity.
#
# To help you in your analysis, you should consider the following three phenomena:
# - The "six degrees of separation" phenomenon,
# - The "rich gets richer" phenomenon, and
# - The "Hierarchical structure of networks" phenomenon.
#
# Your task for this problem is to consider how one of these phenomena might explain
# the structure of the citation graph or, alternatively, how the citations patterns
# follow one of these phenomena.
#
# When answering this question, please include answers to the following:
#
# Q5.1: Is the plot of the in-degree distribution for the DPA graph similar to that of the
# citation graph? Provide a short explanation of the similarities or differences.
# Focus on the various properties of the two plots as discussed in the class page on
# "Creating, formatting, and comparing plots".

# Ans: Yes they are similar since both follow a linear log-log decreasing trend i.e
# they both follow the power law distribution and the point are spread out more
# as the in-degree increases

# Q5.2: Which one of the three social phenomena listed above mimics the behavior of the DPA
# process? Provide a short explanation for your answer.

# Ans: DPA process mimics the "rich get richer" or the "preferential attachment" phenomena
# since every new node that is added to the graph is most likely to be connected to
# the neighbor with highest in-degree.

# Q5.3: Could one of these phenomena explain the structure of the physics citation graph?
# Provide a short explanation for your answer.

# Ans: The citation graph also mimics the "rich get richer" phenomena as the paper with
# higher citations i.e higher degree tend to be more likely used in other papers
# as well due to being more visible
```

B All functions for project 4 used in the application

```
"""
Functions for Prject #1: "Degree Distribution for Graphs". These functions will be
used in the Application #1: "Analysis of Citation Graphs"
"""

# define directed graph constants for testing
EX_GRAPH0 = {
    0 : set([1, 2]),
    1 : set(),
    2 : set(),
    }

EX_GRAPH1 = {
    0 : set([1, 4, 5]),
    1 : set([2, 6]),
    2 : set([3]),
    3 : set([0]),
    4 : set([1]),
    5 : set([2]),
    6 : set([]),
    }

EX_GRAPH2 = {
```

```
    0 : set([1, 4, 5]),
    1 : set([2, 6]),
    2 : set([3, 7]),
    3 : set([7]),
    4 : set([1]),
    5 : set([2]),
    6 : set(),
    7 : set([3]),
    8 : set([1, 2]),
    9 : set([0, 3, 4, 5, 6, 7]),
    }

def make_complete_graph(num_nodes):
    """
    create and return a complete graph with nodes from
    0 to num_nodes - 1 for num_nodes > 0. Otherwise
    the function returns a dictionary corresponding to
    the empty graph

    Arguments:
        num_nodes {integer} -- number of nodes for the graph

    Returns:
        dictionary -- returns a dictionary corresponding to a complete directed
        graph with the specified number of nodes.
    """
    # local variable for the complete graph
    graph = {}

    # return an empty graph if num_nodes is not positive
    if num_nodes <= 0:
        return graph

    for node in range(num_nodes):
        # create an adjacency list for a directed complete graph with no
        # self loops or parallel edges
        graph[node] = set([val for val in range(num_nodes) if val != node])

    return graph

def compute_in_degrees(digraph):
    """
    computes the in-degree of the nodes in a graph

    Arguments:
        digraph {dictionary} -- a directed graph with no self loop or parallel edges

    Returns:
        dictionary -- returns a dictionary with same set of keys(nodes) as digraph
                      whose corresponding values are the number of edges whose
                      head matches a particular node

    """

    # initialize the in degree for the nodes of digraph to 0
    in_degree = dict(zip(digraph.keys(), len(digraph) * [0]))

    for tail_node in digraph:
        for head_node in digraph[tail_node]:
            in_degree[head_node] += 1

    return in_degree

def in_degree_distribution(digraph):
    """
    computes the unnormalized distribution of the in-degrees of the graph

    Arguments:
        digraph {dictionary} -- a directed graph with no self loops or parallel
                                edges

    Returns:
        dictionary -- unnormalized distribution of the in-degrees of the graph
                      with key being the in-degree of nodes in the graph and
                      the value associated with each particular in-degree is
                      the number of nodes with that in-degree. In-degrees with
                      no corresponding nodes in the graph are not included in
                      the dictionary.
    """
    # initialize the dictionary to store
    in_degree_dist = {}
    # get the in-degree for each node
    in_degrees = compute_in_degrees(digraph)

    for degree_vals in in_degrees.values():
        if in_degree_dist.has_key(degree_vals):
            in_degree_dist[degree_vals] += 1
        else:
            in_degree_dist[degree_vals] = 1

    return in_degree_dist
```

C Code to load the graphs from text file

```python
"""
Common functions used for the both Application #1 and #2
"""

def load_graph(graph_file):
    """
    Helper function to solve Q1 Application #1: Analysis of
    Citation Graphs
    converts the text representation of a graph from
    a text file to dictionary representation.

    Arguments:
        graph_file {string} -- a file name of the file with text representation of a graph

    Returns:
        dictionary -- returns a dictionary representation of the graph
    """
    # will store the dictionary representation of the graph
    graph = {}

    with open(graph_file) as grh_file:
        for line in grh_file:
            # get the tail node and corresponding head nodes in the current
            # line of the adjacency list text representation of graph
            nodes = line.split()
            # convert the head node string to integer
            head_nodes = map(int, nodes[1:])
            # add the key, the tail node, and value, the head nodes to the
            # dictionary representation of the graph
            graph[int(nodes[0])] = set(head_nodes)

    return graph
```