Project 1: Bayesian Structure Learning

Hayato Nakamura hayaton@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

I used the simulated annealing algorithm to search the Bayesian graph structure. The main difficulty I encountered using this approach is the tuning of start and end temperatures, which control the amount of injected randomness. The amount of time my algorithm took to find the graph structure submitted to the Gradescope is
- Small Graph: 21.52741503715515 sec
- Medium Graph: 74.88871836662292 sec
- Large Graph: 643.7997753620148 sec

Note: because my Bayesian score computation utilizes the intermediate values (results of np.unique function) to speed up, the exact runtimes are highly dependent on the temperatures and number of steps given to the simulated annealing function.

2. Graphs

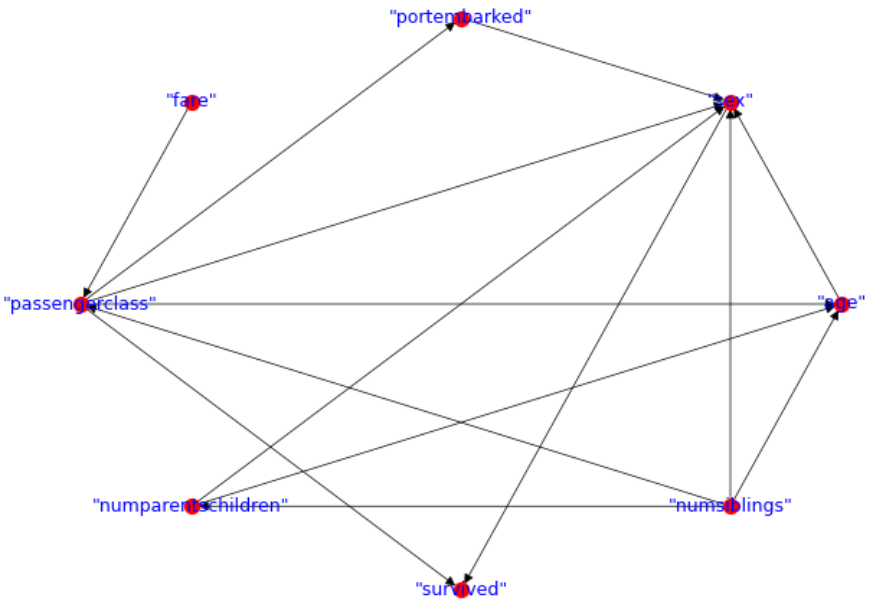

Figure 1: Small Graph

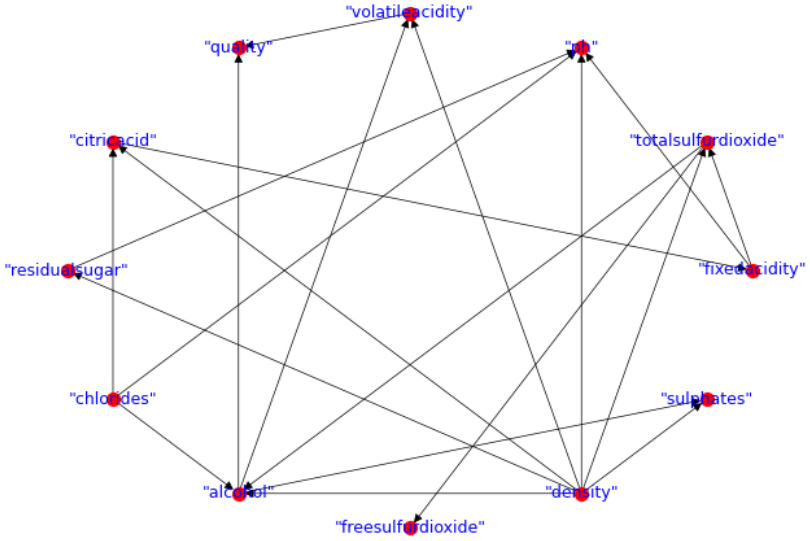

Figure 2: Medium Graph

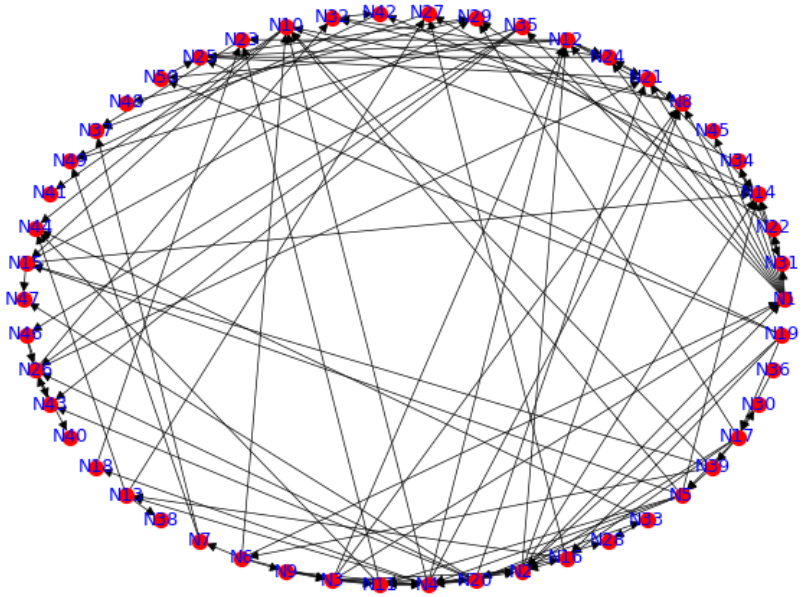

Figure 3: Large Graph

3. Code

```
import sys
import time
import networkx as nx
import numpy as np
from scipy.special import loggamma
import random
import math
from scipy.special import gammaln

unique_dict = {}
def fast_unique(node_set, data):
    '''
    Compute the unique rows and counts of the outcomes of the given node set
    and store the result in a dictionary for future use for efficiency
    Args:
        node_set is a list of integers
        data is a numpy array of shape (n, m)
        - n is the number of data points
        - m is the number of variables
    Returns:
        unique_rows is a numpy array of distinct outcomes of the node set
        counts is a the number of times each outcome appears
    '''
    str_node_set = str(node_set)
    if str_node_set in unique_dict:
        return unique_dict[str_node_set]
    else:
        outcomes = data[:, node_set]
        unique_rows, counts = np.unique(outcomes, axis=0, return_counts=True)
        unique_dict[str_node_set] = [unique_rows, counts]
        return [unique_rows, counts]

def write_gph(dag, idx2names, filename):
    with open(filename, 'w') as f:
        for edge in dag.edges():
            f.write("{}, {}\n".format(idx2names[edge[0]], idx2names[edge[1]]))

def load_gph(names, filename):
    '''
    Load the graph from a gph file

    Args:
        names is a numpy array of shape (m,)
        names[i] is the name of the i-th variable
        filename is the name of the gph file
    Returns:
        dag is a networkx DiGraph
    '''
```

```
    dag = nx.DiGraph()
    with open(filename, 'r') as f:
        lines = f.readlines()
    for line in lines:
        line = line.strip()
        if line == "":
            continue
        nodes = line.split(",")
        # remove the leading and trailing spaces
        nodes = [node.strip() for node in nodes]
        dag.add_edge(np.where(names == nodes[0].strip())[0][0], np.where(
    names == nodes[1].strip())[0][0])
    return dag

def load_csv(infile):
    '''
    Load the data from a csv file

    Args:
        infile is the name of the input file
    Returns:
        names is a numpy array of shape (m,)
        names[i] is the name of the i-th variable
        data is a numpy array of shape (n, m)
        data[i, j] is the value of the j-th variable for the i-th data point
    '''
    with open(infile, 'r') as f:
        lines = f.readlines()

    lines = [line.strip() for line in lines]
    names = lines[0].split(",")
    data = [line.split(",") for line in lines[1:]]

    return np.array(names), np.array(data, dtype=int)

def compute_data_range(data):
    '''
    Compute the range of each variable in the data set

    Args:
        data is a numpy array of shape (n, m)
        - n is the number of data points
        - m is the number of variables
    Returns:
        data_range is a numpy array of shape (m, 3)
        data_range[i, 0] is the minimum value of the i-th variable
        data_range[i, 1] is the maximum value of the i-th variable
        data_range[i, 2] is the number of possible values of the i-th
    variable
    '''
```

```
    data_range = np.zeros((data.shape[1], 3), dtype=int)
    for i in range(data.shape[1]):
        data_range[i, 0] = np.min(data[:, i])
        data_range[i, 1] = np.max(data[:, i])
        data_range[i, 2] = data_range[i, 1] - data_range[i, 0] + 1
    return data_range

def compute(infile, outfile):
    # load the data from csv
    names, data = load_csv(infile)

    start_time = time.time()
    # find the best dag
    # initial_dag = load_gph(names, "./output/large_ckpt.gph")
    # dag = simulated_annealing(data, initial_dag=initial_dag)
    dag = simulated_annealing(data)
    print("Time: ", time.time() - start_time)

    # Write the output file
    write_gph(dag, names, outfile)

def fast_bayesian_score(data, dag):
    score = 0
    data_range = compute_data_range(data)
    # ith node loop
    for node in dag.nodes:
        parent_set = [p for p in dag.predecessors(node)]
        parent_set = sorted(parent_set)
        a_ij0 = data_range[node, -1]
        if len(parent_set) == 0:
            # node has no parent
            unique_rows, counts = fast_unique([node], data)

            m_ij = counts
            m_ij0 = np.sum(m_ij)
            score += loggamma(a_ij0) - loggamma(a_ij0+m_ij0)

            # k-th node value loop
            for m_ijk in m_ij:
                score += loggamma(1+m_ijk)
        else:
            # node has parent
            p_unique_rows, p_counts = fast_unique(parent_set, data)

            parent_set.append(node)
            unique_rows, counts = fast_unique(parent_set, data)

            # j-th parent loop
            for idx, p_count in enumerate(p_counts):
                m_ij0 = p_count
```

```
                score += loggamma(a_ij0) - loggamma(a_ij0+m_ij0)

                # k-th node value loop
                m_ij = counts[np.all(unique_rows[:, :-1] == p_unique_rows[idx
    ], axis=1)]
                for m_ijk in m_ij:
                    score += loggamma(1+m_ijk)
    return score

def simulated_annealing(
        data,
        initial_dag=None,
        num_restarts=10,
        num_steps=1000,
        start_temp=100,
        end_temp=0.1):
    '''Simulated annealing algorithm to find the best DAG
    Args:
        data: numpy array of shape (n, m)
        initial_dag: initial DAG
        num_restarts: number of restarts
        num_steps: number of steps in each restart
        start_temp: starting temperature
        end_temp: ending temperature
    Returns:
        best_dag: best DAG
    '''
    n, m = data.shape
    best_score = -np.inf
    best_dag = None

    for i in range(num_restarts):
        print("restart", i)
        # dag = nx.gnm_random_graph(m, m * (m - 1) // 2, directed=True)
        if initial_dag is None:
            while True:
                # dag = nx.erdos_renyi_graph(m, 0.05, directed=True)
                dag = nx.fast_gnp_random_graph(m, 0.03, directed=True)
                # dag = nx.DiGraph()
                # dag.add_nodes_from(range(m))
                # dag = nx.gnm_random_graph(m, m * (m - 1) // 2, directed=
    True)
                if nx.is_directed_acyclic_graph(dag):
                    break
        else:
            dag = initial_dag.copy()
            print("dag loaded")

        assert nx.is_directed_acyclic_graph(dag) == True
        temp = start_temp
```

```
            score = fast_bayesian_score(data, dag)
            for j in range(num_steps):
                print(i, j, score, temp)
                new_dag = modify_dag(dag)
                if not nx.is_directed_acyclic_graph(new_dag):
                    continue
                new_score = fast_bayesian_score(data, new_dag)
                delta = new_score - score
                if delta > 0:
                    dag = new_dag
                    score = new_score
                    if score > best_score:
                        best_score = score
                        best_dag = dag.copy()
                else:
                    prob = math.exp(delta / temp)
                    if random.uniform(0, 1) < prob:
                        dag = new_dag
                        score = new_score
                temp = start_temp * (end_temp / start_temp) ** (j / num_steps)
            print("current best:",best_score)
        print("overall best:",best_score)
        assert nx.is_directed_acyclic_graph(best_dag) == True
        return best_dag

    def modify_dag(dag):
        '''
        Modify the DAG by adding or removing an edge
        '''
        new_dag = dag.copy()
        nodes = list(new_dag.nodes)
        node1 = random.choice(nodes)
        node2 = random.choice(nodes)
        if node1 == node2:
            return new_dag
        if new_dag.has_edge(node1, node2):
            new_dag.remove_edge(node1, node2)
        else:
            new_dag.add_edge(node1, node2)
        return new_dag

    def main():
        if len(sys.argv) != 3:
            raise Exception("usage: python project1.py <infile>.csv <outfile>.gph
        ")

        inputfilename = sys.argv[1]
        outputfilename = sys.argv[2]
        compute(inputfilename, outputfilename)
```

```
if __name__ == '__main__':
    main()
```