Project 1: Bayesian Structure Learning

Purushotham Mani purush@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

Local Search with randomized reset upon encountering local minima.

I wasn't aware of NetworkX library when I started the project therefore, I built a graph object. And to ensure acyclic nature of the graph I used Kahn's algorithm.

In the Graph object, I maintain both an adjacency list and a parent list. The adjacency list stores the edges between each node and its children, while the parent list functions similarly but stores the edges between each node and its parents. The Graph object also has methods to add, remove, or change the direction of an edge.

The program initializes a count matrix M under the assumption that the graph contains no edges, updating the matrix values as it traverses neighboring graphs. It updates only the value corresponding to the j^{th} variable when an edge (i, j) is added, removed, or its direction is changed. This update occurs whenever an edge is added, removed or direction changed. For the update M[j], the parents of the j^{th} variable are extracted from the parent list and values are updated according to the samples provided in the csv.

During each iteration, the program evaluates every neighbor of the current graph and selects the one with the minimum Bayesian score. If the program encounters a local optimum, it resets to a random graph and performs a local search from there.

The final output is the graph with the lowest Bayesian score across all random resets and local searches.

1.1 Running time

- Small - 2 min
- Medium - 30 min
- Large - 8 hrs

2. Graphs

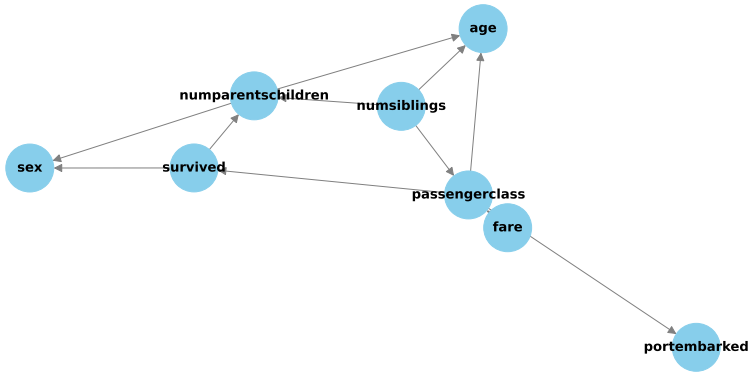

Figure 1: Small Graph

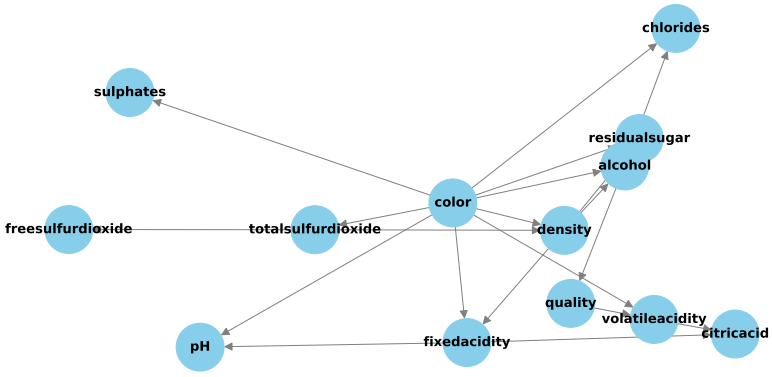

Figure 2: Medium Graph

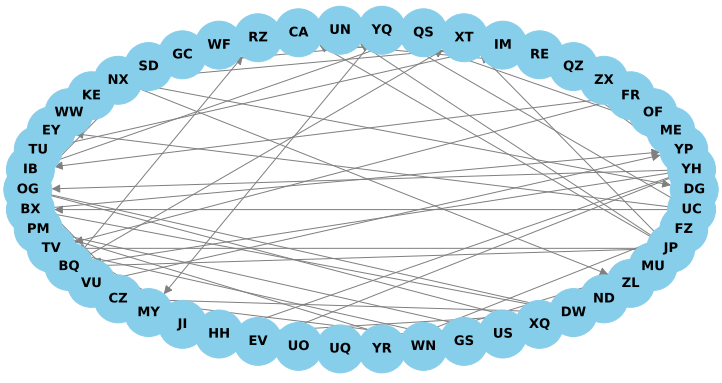

Figure 3: Large Graph

3. Code

```
import sys, copy
import pandas as pd
import numpy as np
from scipy.special import loggamma
from collections import defaultdict, deque
import Visualizer
Parentsize = 3
class Graph:
    def __init__(self, M):
        self.adj_list = defaultdict(set)
        self.parent_list = defaultdict(list)
        self.in_degree = defaultdict(int)
        # self.M = M.copy()

    def add_vertex(self, u):
        self.adj_list[u] = set()
        self.in_degree[u] = 0

    def add_edge(self, u, v, sample):
        if self.creates_cycle(u, v) or len(self.parent_list[v]) == Parentsize
    :
            # print(f"Edge {u} -> {v} creates cycle or exceeds parent limit.")
            return False

        if v not in self.adj_list[u]:
            self.adj_list[u].add(v)
            self.in_degree[v] += 1
            self.parent_list[v].append(u)
            self.M[v] = np.zeros((self.M.shape[1],self.M.shape[2]))
            for i in range(sample.shape[0]):
                if len(self.parent_list[v])==1:
                    self.M[v][sample[i][self.parent_list[v][0]]-1][sample[i][
    v]-1]+=1
                elif len(self.parent_list[v])==2:
                    parents = sample[i][self.parent_list[v]]
                    j = (parents[0]-1)*self.M.shape[2] + (parents[1]-1)
                    self.M[v][j][sample[i][v]-1]+=1
                else:
                    parents = sample[i][self.parent_list[v]]
                    j = (parents[0]-1)*(self.M.shape[2]**2) + (parents[1]-1)*
    self.M.shape[2] + (parents[2]-1)
                    self.M[v][j][sample[i][v]-1]+=1
            return True
        return False

    def remove_edge(self, u, v, sample):
        if v in self.adj_list[u]:
            self.adj_list[u].remove(v)
```

```
            self.in_degree[v] -= 1
            self.parent_list[v].remove(u)
            self.M[v] = np.zeros((self.M.shape[1],self.M.shape[2]))
            for i in range(sample.shape[0]):
                if len(self.parent_list[v])==1:
                    self.M[v][sample[i][self.parent_list[v][0]]-1][sample[i][
    v]-1]+=1
                elif len(self.parent_list[v])==2:
                    parents = sample[i][self.parent_list[v]]
                    j = (parents[0]-1)*self.M.shape[2] + (parents[1]-1)
                    self.M[v][j][sample[i][v]-1]+=1
                else:
                    self.M[v][0][sample[i][v]-1]+=1
            return True
        return False

    def change_direction(self, u, v, sample):
        if self.remove_edge(u, v, sample):
            if not self.add_edge(v, u, sample):
                # Revert if adding v -> u causes a cycle
                self.add_edge(u, v, sample)
                return False
            # print("direction changes", v, u)
            return True
        return False

    def creates_cycle(self, u, v):
        """Check if adding edge u -> v creates a cycle."""
        temp_adj_list =
        defaultdict(set, {k: set(v) for k, v in self.adj_list.items()})
        temp_adj_list[u].add(v)
        return not self.topological_sort_possible(temp_adj_list)

    def topological_sort_possible(self, adj_list):
        """Check if a topological ordering is possible (i.e., no cycles)."""
        in_degree = {node: 0 for node in adj_list}
        for neighbors in adj_list.values():
            for neighbor in neighbors:
                in_degree[neighbor] += 1

        queue = deque([node for node in in_degree if in_degree[node] == 0])
        visited_count = 0

        while queue:
            node = queue.popleft()
            visited_count += 1
            for neighbor in adj_list[node]:
                in_degree[neighbor] -= 1
                if in_degree[neighbor] == 0:
                    queue.append(neighbor)
```

```
        return visited_count == len(in_degree)  # True if no cycle exists

def Bayesian_Score(M):
    alpha = np.ones((M.shape[0], M.shape[1], M.shape[2]))
    alpha0 = np.sum(alpha, axis=2)
    M0 = np.sum(M, axis=2)
    term1 = np.sum(loggamma(alpha0) - loggamma(alpha0 + M0))
    term2 = np.sum(loggamma(alpha + M) - loggamma(alpha))
    return term1 + term2

def random_graph(M, sample):
    # Initialize graph with zero edges
    G = Graph(M)
    N = M.shape[0]
    max_edges = np.random.randint(N)
    for j in range(N):
        G.add_vertex(j)
    edge_count = 0
    while edge_count < max_edges:
        u, v = np.random.choice(N, size=2, replace=False)
        if G.add_edge(u, v, sample):
            edge_count += 1
    return G

def write_graph(graph, var_names,outfile):
    """Write the learned graph to a file."""
    with open(outfile, 'w') as f:
        for u in graph.adj_list:
            for v in graph.adj_list[u]:
                f.write(f"{var_names[u]},{var_names[v]}\n")

def compute(infile, outfile):
    samples = pd.read_csv(infile)
    var_size = len(samples.columns)
    max_value = samples.max().max()
    samples_array = samples.to_numpy()

    # Initialize count matrix M
    M = np.zeros((var_size, max_value ** Parentsize, max_value))
    for i in range(samples.shape[0]):
        for j in range(var_size):
            M[j, 0, samples.iloc[i, j] - 1] += 1
    # Initialize graph with zero edges
    G = Graph(M)
    for j in range(var_size):
        G.add_vertex(j)

    prev_score = Bayesian_Score(G.M)
    Max_G = copy.deepcopy(G)
```

```
max_score = prev_score
failure_count=0
print ("Initial score", prev_score)
for n_iteration in range(500):
    current_score = Bayesian_Score(G.M)
    if (abs(prev_score-current_score)<=1e-10) and n_iteration!=0:
        if (current_score > max_score):
            Max_G = G
            max_score = current_score
        else:
            failure_count+=1
            if (failure_count>=5):
                break
        # print("changing to a random graph")
        G = random_graph(M,samples_array)
        current_score = Bayesian_Score(G.M)
    prev_score = current_score
    G_max = G
    print("iteration", n_iteration, failure_count)
    # print(Bayesian_Score(G.M))
    for v1 in range(var_size):
        for v2 in range(var_size):
            if (v1!=v2):
                for act in range(1,4):
                    if act==1:
                        if G.add_edge(v1,v2, samples_array):
                            new_score = Bayesian_Score(G.M)
                            if (new_score>current_score):
                                G_max = copy.deepcopy(G)
                                current_score = new_score
                            G.remove_edge(v1,v2, samples_array)
                    elif act==2:
                        if G.remove_edge(v1,v2,samples_array):
                            new_score = Bayesian_Score(G.M)
                            if (new_score>current_score):
                                G_max = copy.deepcopy(G)
                                current_score = new_score
                            G.add_edge(v1,v2,samples_array)
                    else:
                        if G.change_direction(v1,v2,samples_array):
                            new_score = Bayesian_Score(G.M)
                            if (new_score>current_score):
                                G_max = copy.deepcopy(G)
                                current_score = new_score
                            G.change_direction(v2,v1,samples_array)
    G = G_max
print("end", Max_G.adj_list)
print(max_score)
write_graph(Max_G, list(samples),outfile)
Visualizer.visualize_graph(Max_G,list(samples))
```

```
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()
```