Dataset	Bayesian score	Runtime
small	-3840.8151913305037	3.266s
medium	-42736.538224422446	89.056s
large	-461425.2944602857	39m 42s

Table 1: Results

Project 1: Bayesian Structure Learning

Ha Tran hahntrn@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

I used the K2 algorithm with a uniform Dirichlet prior as the search strategy and Bayesian scoring as the scoring function. The graph begins with only nodes, no edges. As we iterate over each node in each observation, we add parents to the node greedily and keep the edges that yield the highest increasing Bayesian score.

2. Graphs

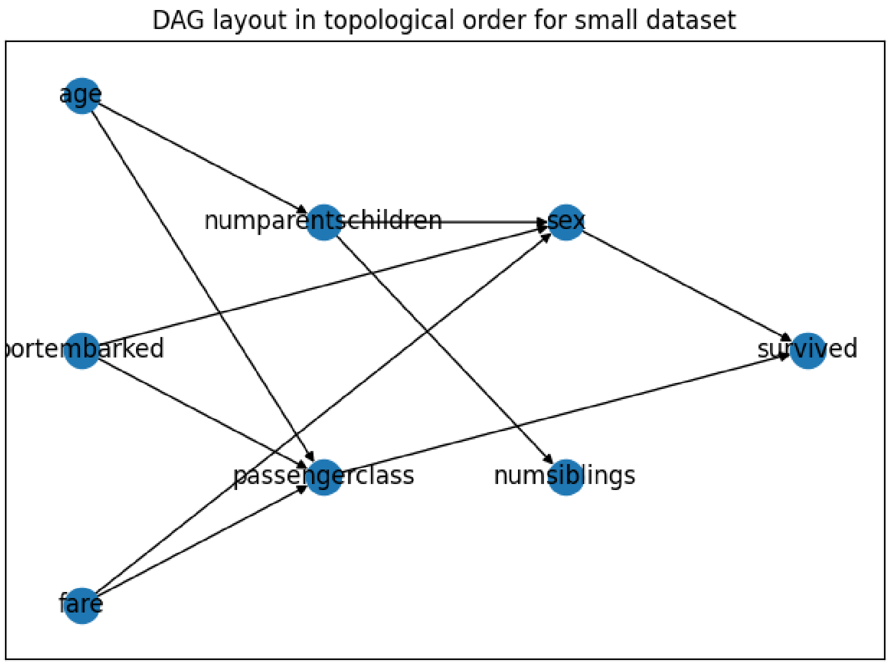

Figure 1: Small graph

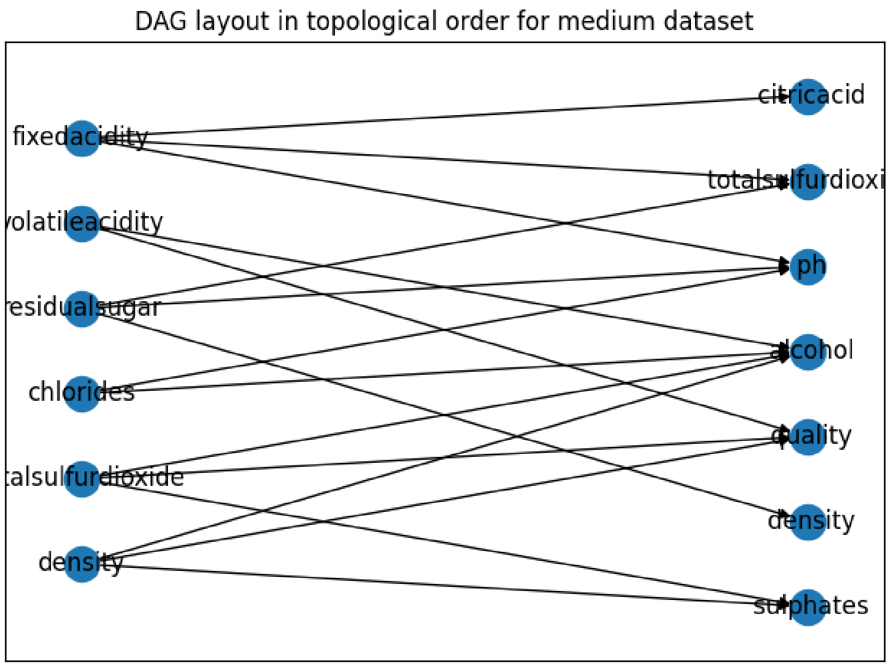

Figure 2: Medium graph

3. Code

```python
import networkx as nx
import sys
import time
import pandas as pd
import numpy as np
from scipy.special import loggamma
import matplotlib.pyplot as plt

MAX_PARENTS = 4

def get_parents(G, node):
    return [neighbor[0] for neighbor in G.in_edges(node)]

def get_children(G, node):
    return [neighbor[1] for neighbor in G.out_edges(node)]

def get_q(G, variables):
    return [np.prod([G.nodes[parent]['r']
        for parent in get_parents(G, var)], dtype=int)
        for var in variables]
```

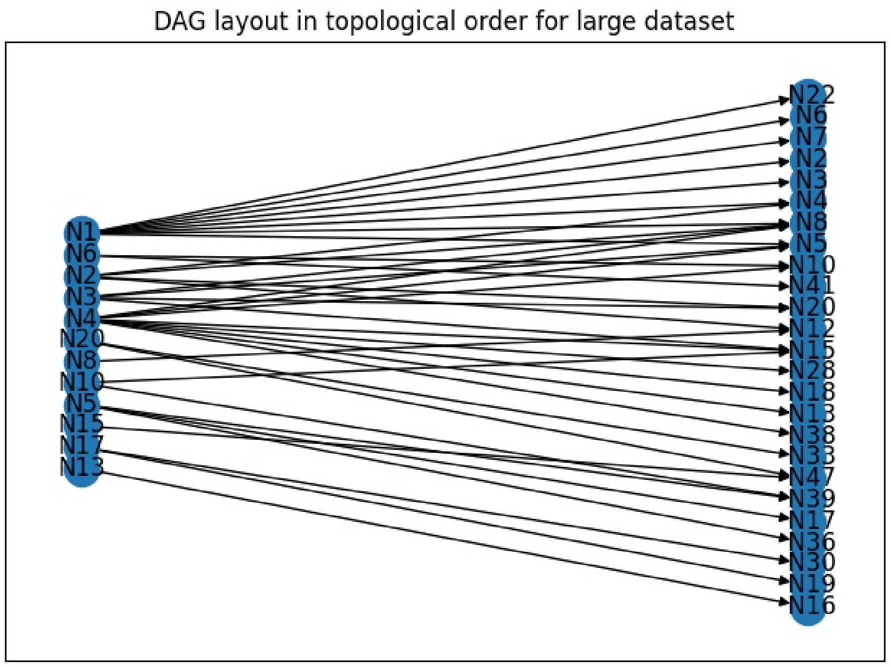

Figure 3: Large graph

```
def get_r(G, variables):
    return [G.nodes[var]['r'] for var in variables]

def get_j_index(qi, parents_r_values, observed_parents_values, parents):
    one = np.array([1], dtype=int)
    cump = np.cumprod(parents_r_values[:-1])
    k = np.concatenate([one, cump])
    observed_parents_values = np.array(observed_parents_values) - 1
    return (np.dot(k, observed_parents_values)).astype(int)

def statistics(variables, G, D):
    n = D.shape[1]
    q = get_q(G, variables)
    M = [np.zeros((q[i], G.nodes[variables[i]]['r']), dtype=int)
        for i in range(n)]
    # print(q, [m.shape for m in M])
    for obs_i, obs in D.iterrows():
        for i,var in enumerate(variables):
            # use the observed value of variable as index into M
            k = obs[var]-1
            parents = get_parents(G, variables[i])
            j = 0
            if len(parents) > 0:
                j = get_j_index(q[i],
                    [G.nodes[parent]['r'] for parent in parents],
                    [obs[parent] for parent in parents], parents)
            M[i][j,k] += 1
    return M

def prior(variables, G):
    n = len(variables)
    r = get_r(G, variables)
    q = get_q(G, variables)
    return [np.ones((q[i], r[i])) for i in range(n)]

def bayesian_score_component(M, alpha):
    p =  np.sum(loggamma(M + alpha))
    p -= np.sum(loggamma(alpha))
    p += np.sum(loggamma(np.sum(alpha, axis=1)))
    p -= np.sum(loggamma(np.sum(alpha, axis=1) + np.sum(M, axis=1)))
    return p

def bayesian_score(variables, G, D):
    n = len(variables)
    M = statistics(variables, G, D)
    alpha = prior(variables, G)
    return sum(bayesian_score_component(M[i], alpha[i]) for i in range(n))

def fit(variables, data, dataset_name, tic):
    G = nx.DiGraph()
    for i, v in enumerate(variables):
        G.add_node(v, r=max(data[v]), index=i)

    sorted_variables = list(nx.topological_sort(G))
    for k,i in enumerate(sorted_variables[1:]):
        y = bayesian_score(variables, G, data)
        while True:
            y_best, j_best = -np.inf, None
            for j in sorted_variables[:k]:
                if G.has_edge(j, i):
                    continue
                G.add_edge(j, i)
                y_new = bayesian_score(variables, G, data)
                if y_new > y_best:
                    y_best, j_best = y_new, j
                G.remove_edge(j, i)
            if y_best > y:
                print(f'best bayesian score: {y_best} (old: {y})')
                toc = time.perf_counter()
                print('Elapsed time:', toc-tic, 'seconds')
                if G.in_degree(i) > MAX_PARENTS:
                    # limit number of parents
                    break
                G.add_edge(j_best, i)
                y = y_best
                if dataset_name == 'large':
                    write_gph(G, {v:v for v in (variables)},
                        f"data/{dataset_name}.gph")
                    plot_graph(dataset_name, G)
            else:
                break
    return G

def plot_graph(dataset_name, G=None):
    if G is None:
        edgelist = np.genfromtxt(f"data/{dataset_name}.gph",
            delimiter=",", dtype=str)
        G = nx.DiGraph()
        G.add_edges_from(edgelist)

    for layer, nodes in enumerate(nx.topological_generations(G)):
        # `multipartite_layout` expects the layer as a node attribute,
        # so add the numeric layer value as a node attribute
        for node in nodes:
            G.nodes[node]["layer"] = layer

    # Compute the multipartite_layout using the "layer" node attribute
    pos = nx.multipartite_layout(G, subset_key="layer")

    fig, ax = plt.subplots()
    nx.draw_networkx(G, pos=pos, ax=ax)
    ax.set_title(f"DAG layout in topological order for {dataset_name} dataset
    ")
    fig.tight_layout()
    plt.savefig(f"data/{dataset_name}.png")

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 compute(dataset_name):
    t0 = time.perf_counter()
    data = pd.read_csv(f"data/{dataset_name}.csv")
    variables = list(data.columns)
    G = fit(variables, data, dataset_name, t0)
    print(f'Done training {dataset_name} dataset! Final bayesian score:',
        bayesian_score(variables, G, data))
    print('Elapsed time:', time.perf_counter() - t0, 'seconds')

def main():
    if len(sys.argv) != 2:
        raise Exception("usage: python project1.py <dataset_name>")
    compute(sys.argv[1])

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