Project 1: Bayesian Structure Learning

Jong Ha, Lee jongha98@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

The algorithm I used primarily was the K2 algorithm. In summary, for each "children" node that we specify, the K2 algorithm attempts to greedily add potential parent nodes to get a better Bayesian score. It requires an initial ordering of nodes (typically one can think of it as left to right) which defines a hierarchy specifying which nodes can be parents for which other nodes. Then, given the ordering, K2 greedily adds one parent at a time to a given child node until the Bayesian score doesn't improve, and moves onto the next children node. While simple to understand and to implement, K2 does not guarantee a global optimal structure, and may be computationally complex if one does not specify the maximum number of parents for any one node. In my case, I specified the maximum number of parents to be 25% of the total nodes in the graph structure.

2. Running Time

Runtimes are shown in Figure 1. Specifically,

- Small: 3.03 seconds
- Medium: 13.82 seconds
- Large: 23 minutes, 53 seconds

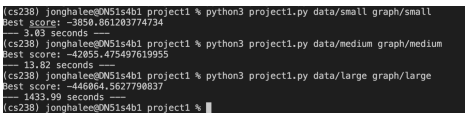

Figure 1: Runtime by datataset size.

3. Graphs

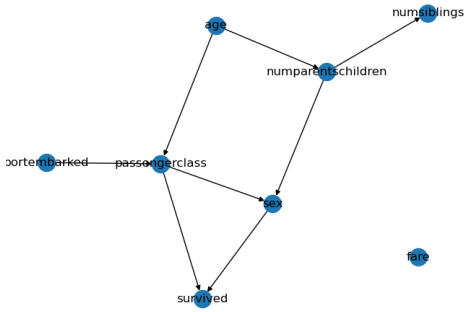

Figure 2: Best proposed graph for small dataset.

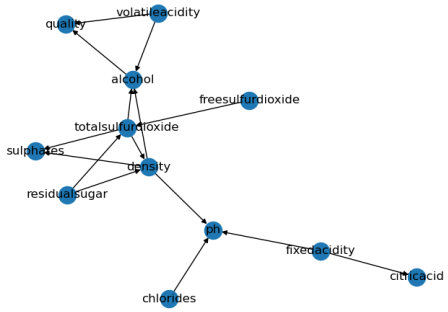

Figure 3: Best proposed graph for medium dataset.

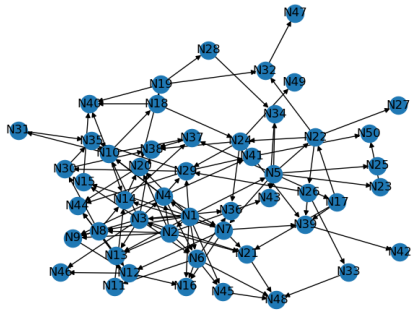

Figure 4: Best proposed graph for large dataset.

4. Code

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

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(infile, outfile):
    # WRITE YOUR CODE HERE
    # FEEL FREE TO CHANGE ANYTHING ANYWHERE IN THE CODE
    # THIS INCLUDES CHANGING THE FUNCTION NAMES, MAKING THE CODE MODULAR,
    BASICALLY ANYTHING
    df = read_data(infile+'.csv')

    # Run K2 algorithm for df
    G = nx.DiGraph()
    ordering = list(df.columns)
    G.add_nodes_from(ordering)
    best_graph, best_score = k2_algo(G, df, ordering, len(ordering) // 4)
    print("Best score:", best_score)

    # Write .gph output
    out_df = pd.DataFrame(columns=['src', 'tgt'])
    for src, tgt in best_graph.edges:
        out_df = out_df.append({'src': src, 'tgt':tgt}, ignore_index=True)
    out_df.to_csv(outfile+'.gph', header=False, index=False)

    # Graph viz write
    pos = nx.nx_agraph.graphviz_layout(best_graph)
    nx.draw(best_graph, with_labels=True, pos=pos)
    plt.savefig(outfile+".png", format="PNG")


def read_graph(gph_fname):
    df = pd.read_csv(gph_fname, header=None, names=['source', 'target'])
    gph = nx.DiGraph()
    G = nx.from_pandas_edgelist(df, create_using=gph)

    return G
```

```
def read_data(data_fname):
    df = pd.read_csv(data_fname)
    return df

def node_bayesian_score(child, nx_graph, orig_data):
    data = orig_data.copy()

    parents = list(nx_graph.predecessors(child))
    # If no parents, then groupby all columns
    if len(parents) == 0:
        data['parent'] = 1
        parents = ['parent']

    # count at ijk level
    ijk_df = (
        data
        .groupby(parents+[child])
        .size()
        .reset_index(name='m_ijk')
    )

    # uniform priors
    ijk_df['alpha_ijk'] = 1

    # counts at ij0 level
    ij0_df = (
        ijk_df
        .groupby(parents)['m_ijk']
        .sum()
        .reset_index()
    )
    ij0_df.columns = parents+['m_ij0']
    ij0_df['alpha_ij0'] = ijk_df.drop_duplicates(subset=[child, 'alpha_ijk'])
    ['alpha_ijk'].sum()

    # Calculating bayesian score
    p = np.sum(loggamma(ijk_df['alpha_ijk'] + ijk_df['m_ijk']))
    p -= np.sum(loggamma(ijk_df['alpha_ijk']))
    p += np.sum(loggamma(ij0_df['alpha_ij0']))
    p -= np.sum(loggamma(ij0_df['alpha_ij0'] + ij0_df['m_ij0']))

    return p

def graph_bayesian_score(nx_graph, orig_data):
    bayesian_score = 0
    for child in nx_graph.nodes():
        curr_p = node_bayesian_score(child, nx_graph, orig_data)
        bayesian_score += curr_p

    return bayesian_score
```

```
def k2_algo(orig_graph, orig_data, ordering, max_parents):
    nx_graph = orig_graph.copy()
    # For each child node:
    for (i, child) in enumerate(ordering[1:]):
        # print("CHILD:", child)
        global_score = graph_bayesian_score(nx_graph, orig_data)

        # keep going with current child node
        # until 1) too many parents added, or 2) adding parent not good
    enough
        while True and len(list(nx_graph.predecessors(child))) < max_parents:
            best_parent_score, best_parent = float('-inf'), 0

            # Get the best parent to add given current graph
            for parent in ordering[:(1+i)]:

                # Only add parent if not already added yet
                if not nx_graph.has_edge(parent, child):
                    nx_graph.add_edge(parent, child)
                    test_score = graph_bayesian_score(nx_graph, orig_data)
                    if test_score > best_parent_score:
                        best_parent_score, best_parent = test_score, parent
                    nx_graph.remove_edge(parent, child)

            # If best parent > global graph, add edge
            # If even after iterating thru all parents not good enough, move
    onto next node
            if best_parent_score > global_score:
                global_score = best_parent_score
                nx_graph.add_edge(best_parent, child)
            else:
                break
    return nx_graph, global_score


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__':
    start_time = time.time()
    main()
    print("--- %s seconds ---" % round(time.time() - start_time, 2))
```