Project 1: Bayesian Structure Learning

Thomas Sounack tsounack@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

The algorithm is implemented using Object Oriented Programming. The **project1.py** file is used to interpret the terminal input, and instantiates a graph object to call its underlying functions.

The **graph.py** file defines the graph class. When an object is instantiated with an input file, the dataset and networkx graph containing its nodes are set as attributes.

1.1 Computing the bayesian score

The bayesian score is computed using two functions: on that computes the bayesian score for any node, and one that sums this score over all nodes. This approach enables significant computational gains when only a small part of the graph is modified, as the bayesian score does not have to be completely recomputed.

To compute the bayesian score for a node, the algorithm finds the number of possible instances for the node and the list of instances combination for the node's parents. For each combination, the algorithm isolates the data corresponding to the combination and computes the associated bayesian score. Summing these results yields the node's bayesian score.

1.2 K2 algorithm

The K2 algorithm first computes the local score for each node. It then iterates over the nodes of the graph.

For each node (potential child), the algorithm keeps iterating over all other nodes (potential parents) to add edges until no improvement is seen: for each potential parent K2 first verifies that the node is valid, ie not equal to the child node, not its parent or descendant, and not the last node. Indeed, from the observed datasets the last node always corresponds to the expected final state (survived or not, quality of the wine), and it would make little sense to have these parameters influence other parameters. If the parent is valid, an edge is drawn and the child's bayesian score recomputed. If this edge does not lead to an acyclic graph and yields improved results, it is saved as the current best option and the edge is removed. After every other potential parent has been tested, the edge that yields the best results is added to the graph. If there has been no improvement with any potential parents, the algorithm breaks out of the while loop and considers another potential child.

After every node has undergone this process, the score is computed as the sum of each node's bayesian score.

1.3 Exporting the results

Three functions are used to export the results. **export_bayesian_score** exports the network's bayesian score as a .score file, **write_gph** writes the graph as a .gph file containing a list of (parent, child) combinations, and **draw_gph** exports the graph as a pdf using networkx's drawing function and graphviz.

2. Runtime for each problem

Runtime for each dataset		
Small	Medium	Large
4 seconds	17 seconds	28 minutes

3. Graphs

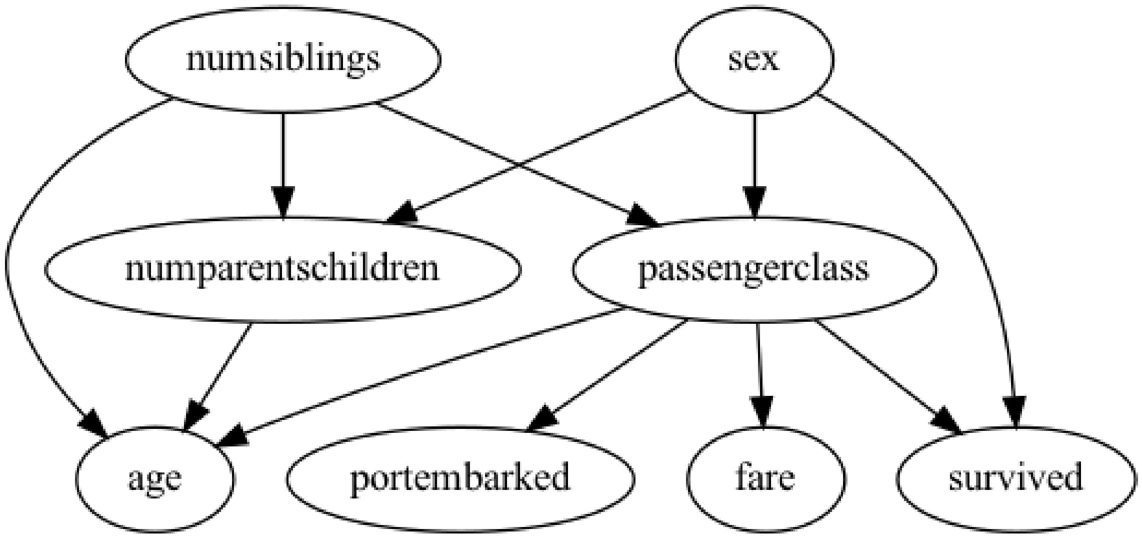

Figure 1: Graph produced with the small dataset

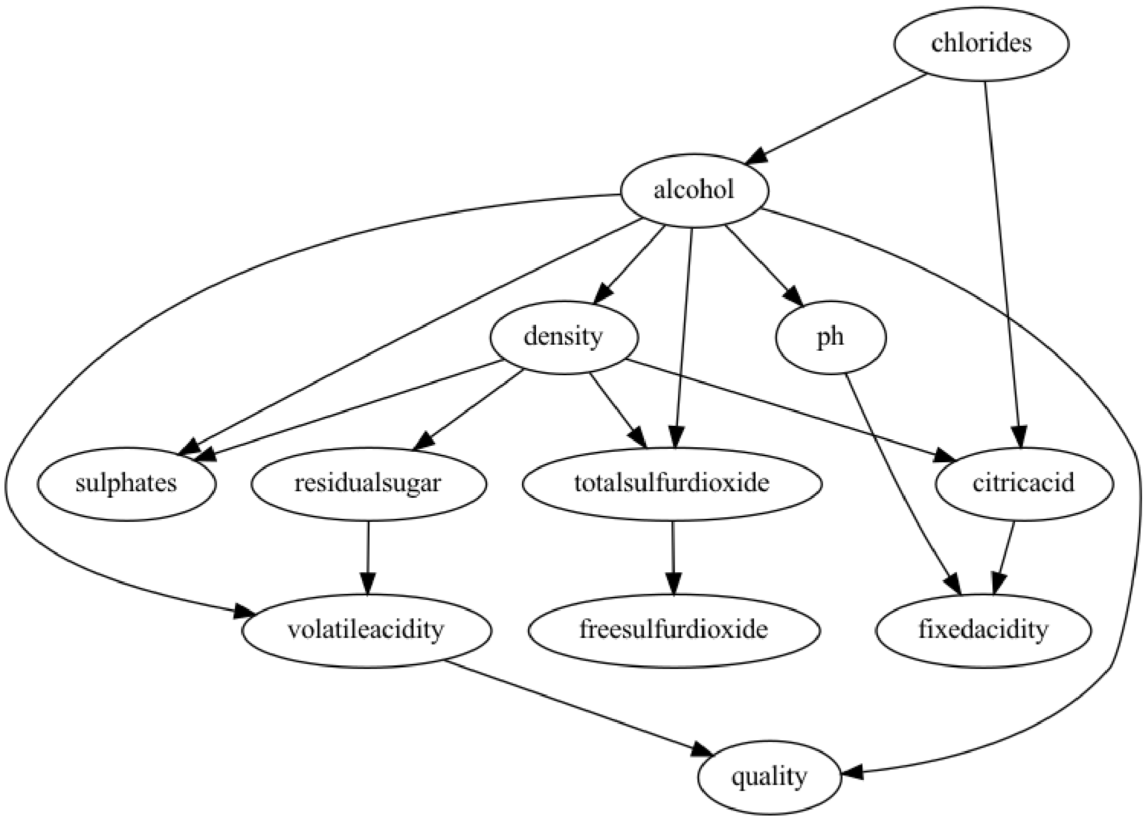

Figure 2: Graph produced with the medium dataset

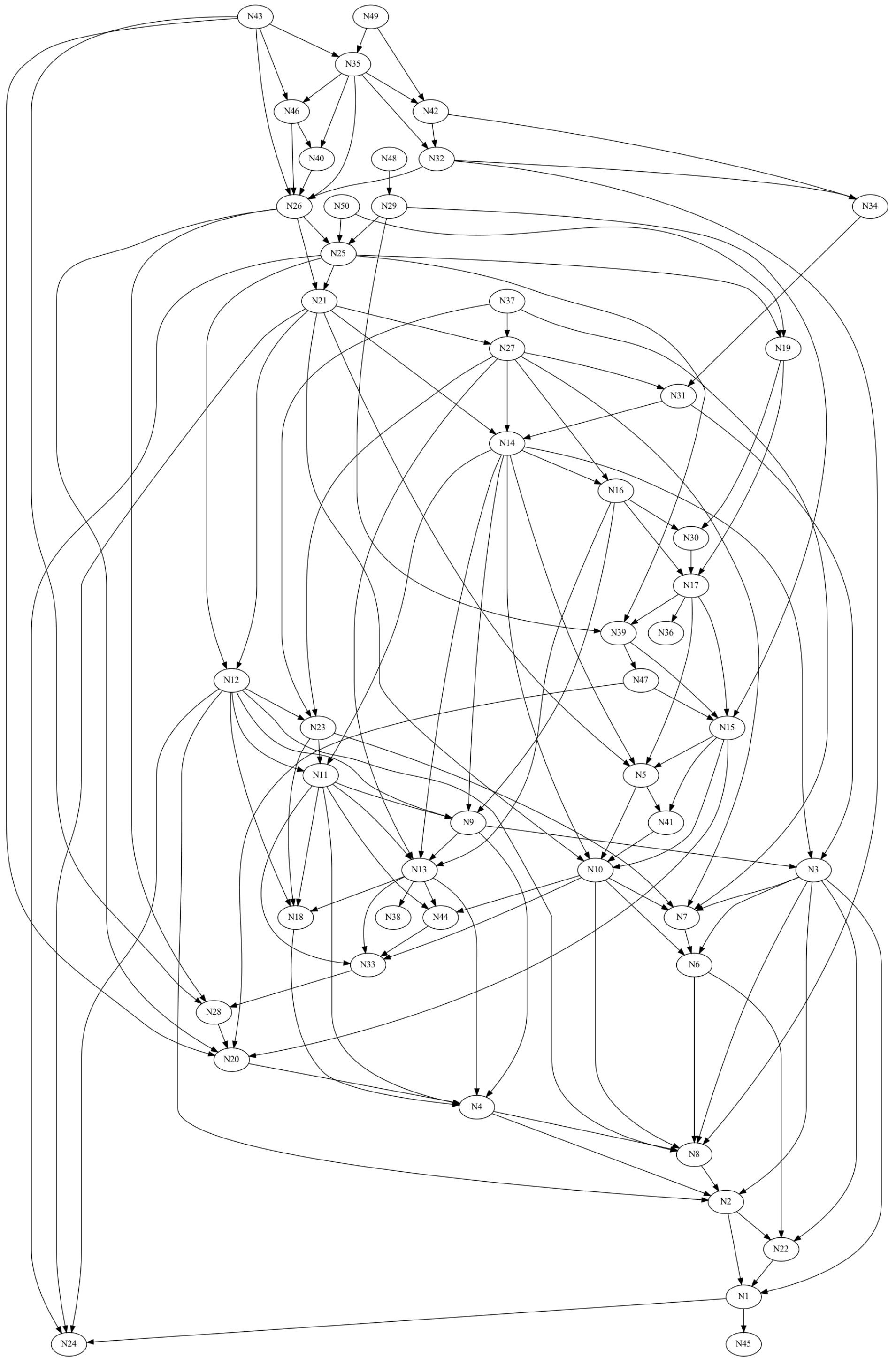

Figure 3: Graph produced with the large dataset

4. Code

```
import sys

import graph


def compute(infile, outfile):
    G = graph.Graph(infile)
    G.K2_search()
    G.export_bayesian_score(outfile)
    G.draw_gph(outfile)
    G.write_gph(outfile)


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()
```

Listing 1: project1.py

```
from scipy.special import gammaln

import graphviz
import itertools
import matplotlib.pyplot as plt
import networkx as nx
import os
import pandas as pd
import pydot
import tempfile

class Graph:
    """
    This class is used to create, modify and perform operations on graphs.
    """

    def __init__(self, infile):
        """
        This function initiates the graph object using a pandas dataframe. It
        sets the networkx graph and dataframe as attributes, and adds nodes
        corresponding to the dataframe's column names.
        """
        self.graph = nx.DiGraph()
        self.data  = pd.read_csv(infile)
        self.graph.add_nodes_from(list(self.data.columns))


    def compute_bayesian_score_node(self, node):
        """
        This function takes a node as argument to compute its bayesian score.
        """
        r = max(self.data[node])
        q = itertools.product(*[self.data[parent].unique() for parent in list
    (self.graph.pred[node])])

        score = 0

        for instance in q:
            query = ""
            for i, parent in enumerate(self.graph.pred[node]):
                query = f"{parent} == {instance[i]}" if i == 0 else f"{query}
    & {parent} == {instance[i]}"

            temp_data = self.data.query(query) if query != "" else self.data
            score += gammaln(r) - gammaln(r + len(temp_data))

            for node_instance in range(r):
                m = len(temp_data[temp_data[node] == node_instance + 1])
                score += gammaln(1 + m)

        return score


    def compute_bayesian_score(self):
        """
        This function computes the bayesian score of the whole graph using
    the
        helper function compute_bayesian_score_node.
        """
        return sum(self.compute_bayesian_score_node(node) for node in self.
    graph)


    def K2_search(self):
        """
        This function implements the K2 algorithm to modify the graph's edges
        until it is optimized.
        """
        counter = 1
        while True:
            initial_score = self.compute_bayesian_score()
            print("K2 Trial: ", counter, "\n")

            local_scores = [self.compute_bayesian_score_node(node) for node
    in self.graph]

            for i, node in enumerate(self.graph):
                print("K2 Search: node {} / {}".format(i+1, len(self.graph)))

                while True:
                    best_local_score = local_scores[i]
                    best_parent      = None

                    for parent in self.graph:

                        if parent == node                  or \
                           parent in self.graph.pred[node] or \
                           node in self.graph.pred[parent] or \
                           parent == self.data.columns[-1]:
                            continue

                        self.graph.add_edge(parent, node)
                        local_score = self.compute_bayesian_score_node(node)

                        if nx.is_directed_acyclic_graph(self.graph) and \
                           local_score > best_local_score:
                            best_local_score, best_parent = local_score,
    parent

                        self.graph.remove_edge(parent, node)

                    if best_parent:
                        local_scores[i] = best_local_score
                        self.graph.add_edge(best_parent, node)

                    else:
                        break

            counter += 1
            final_score = sum(local_scores)
            print("\n" + "Score: ", final_score)
            print("----------------------")
            if final_score == initial_score:
                break


    def export_bayesian_score(self, outfile):
        """
        This function is used to output the bayesian score of the graph to a
        .score file using the outfile string as output path.
        """
        with open(outfile + ".score", 'w') as f:
            f.write(str(self.compute_bayesian_score()))


    def write_gph(self, filename):
        """
        This function writes the graph to a .gph file using:
        - filename: file name for the output file
        """
        with open(filename + ".gph", 'w') as f:
            for edge in self.graph.edges():
                f.write("{}, {}\n".format(edge[0], edge[1]))


    def draw_gph(self, filename):
        """
        This function takes the current graph and exports it as a .png file
    using
        the filename variable as the destination path.
        """
        with tempfile.NamedTemporaryFile() as f:
            nx.drawing.nx_pydot.write_dot(self.graph, f.name)
            img_file = graphviz.render('dot', 'png', f.name)
            os.rename(img_file, filename + ".png")
```

Listing 2: graph.py