Approach:

I first imported the data from the CSV into a DataFrame. I extracted the values as a Matrix, and transposed it. I then constructed the array of variable structures by setting the name of each variable, as well as the number of possible values, which is simply the maximum value that appears for that variable in the data. Then, I applied the K2 algorithm. I arbitrarily used the ordering of variables 1 to *n*. The K2 algorithm is a hill-climbing algorithm used for learning the structure of a Bayesian network from data. The fit function takes an instance of K2Search, a list of variables vars, and a data matrix D. It initializes an empty directed graph G using the SimpleDiGraph type from the Graphs package. The function then iterates over the variables in the specified order and attempts to add edges to the graph based on a Bayesian scoring criterion.

The K2 algorithm proceeds as follows:

1. Initialize an empty directed graph G with nodes corresponding to the variables in the input data.
2. Iterate over the variables in the specified order (excluding the first variable, as indicated by [2:end]).
3. Inside the loop for each variable i, it initializes a variable y with the Bayesian score of the current graph.
4. Enter a while loop that iteratively considers adding edges to the graph and evaluates the Bayesian score (y') for each modified graph.
5. For each potential parent variable j (from the ordering up to the current variable i), check if adding an edge from j to i improves the Bayesian score.
6. If adding the edge improves the score, update the best score (y_best) and the best parent variable (j_best).
7. After evaluating all potential parent variables, if a better parent is found, add the corresponding edge to the graph. Otherwise, break out of the while loop.
8. The process is repeated for the next variable in the ordering until all variables have been processed.
9. The final directed acyclic graph (DAG) with added edges is returned.

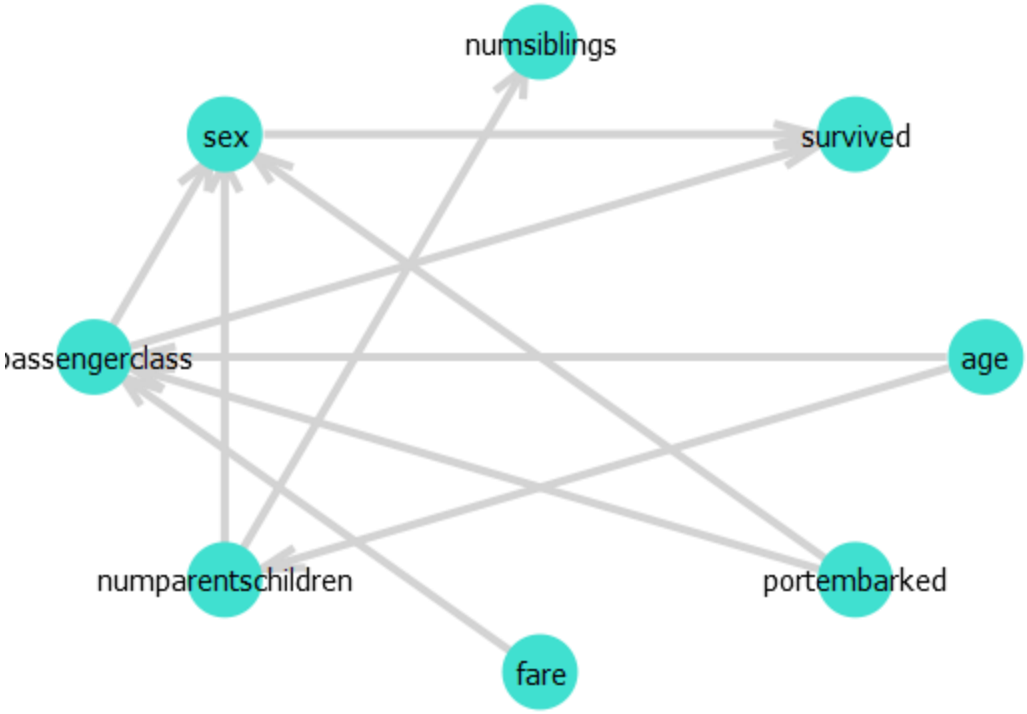
Afterwards, I simply write out the graph and the final Bayesian Score. I used the GraphPlot package in Julia combined with Compose and Cairo to draw the graph and save it as a PNG.

Results:

<u>Small</u>:

Bayesian Score: -3835.67942521279

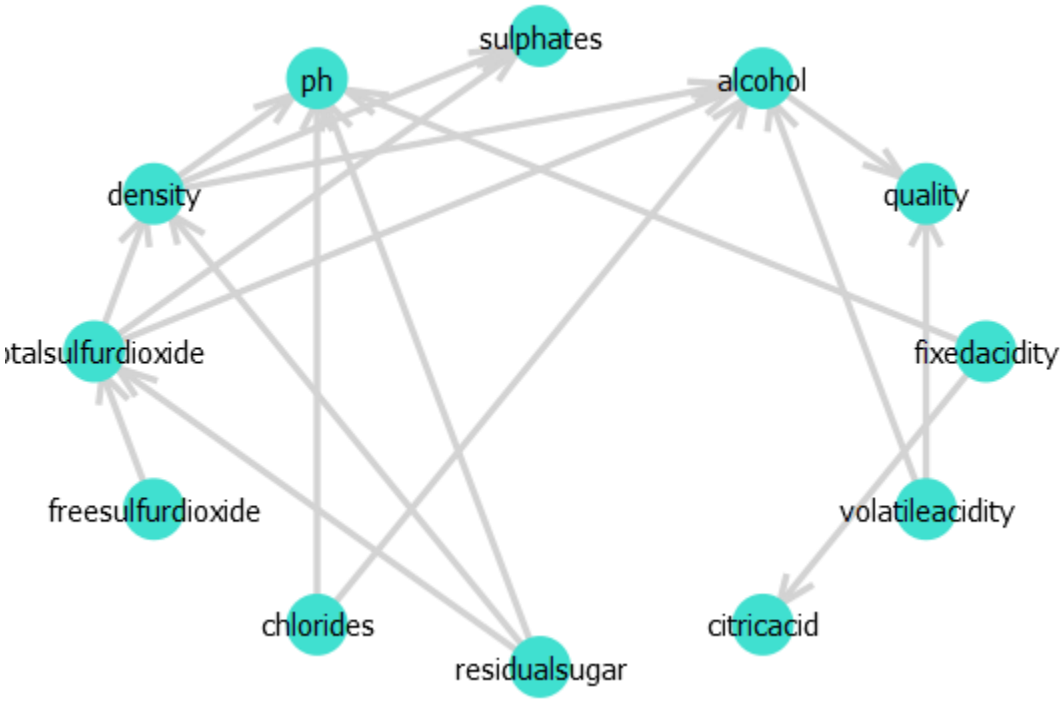
Runtime: 15.093954 seconds



Medium:

Bayesian Score: -42060.47640776388

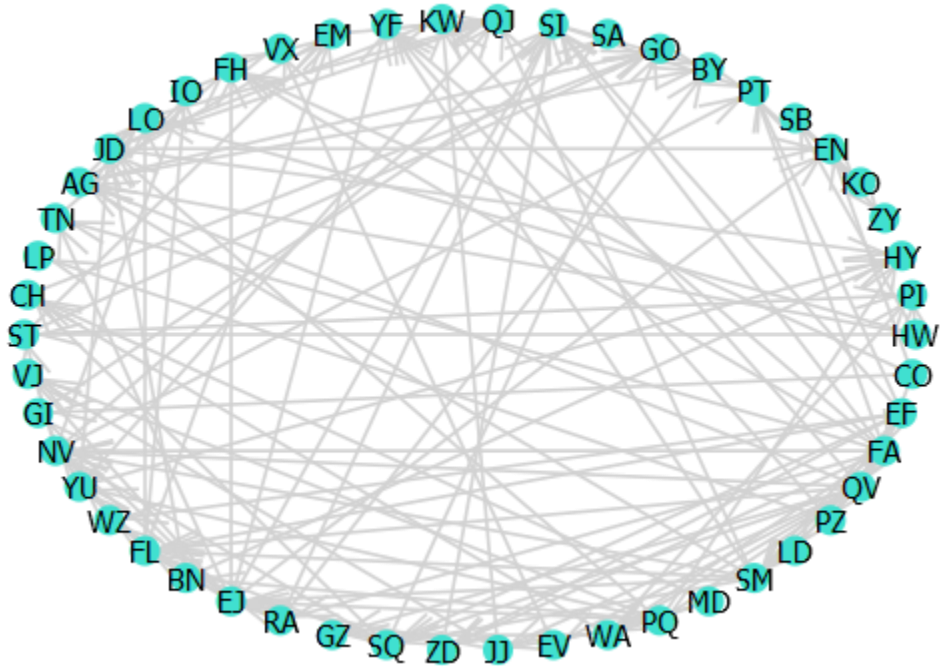
Runtime: 24.539603 seconds



Large:

Bayesian Score: -408087.15267893655

Runtime: 1261.929406 seconds



Code:

```
using Graphs
using Printf
using CSV
using DataFrames
using SpecialFunctions
using LinearAlgebra
using Graphs  # for DiGraph and add_edge!
using TikzGraphs   # for TikZ plot output
using TikzPictures # to save TikZ as PDF
using GraphPlot
using Compose, Cairo

"""
    write_gph(dag::DiGraph, idx2names, filename)
```

```
Takes a DiGraph, a Dict of index to names and a output filename to write the
graph in `gph` format.
"""
function write_gph(dag::DiGraph, idx2names, filename)
    open(filename, "w") do io
        for edge in edges(dag)
            @printf(io, "%s,%s\n", idx2names[src(edge)], idx2names[dst(edge)])
        end
    end
end

function prior(vars, G)
    n = length(vars)
    r = [vars[i].r for i in 1:n]
    q = [prod([r[j] for j in inneighbors(G,i)]) for i in 1:n]
    return [ones(q[i], r[i]) for i in 1:n]
end

function bayesian_score_component(M, α)
    p = sum(loggamma.(α + M))
    p -= sum(loggamma.(α))
    p += sum(loggamma.(sum(α,dims=2)))
    p -= sum(loggamma.(sum(α,dims=2) + sum(M,dims=2)))
    return p
    end


function bayesian_score(vars, G, D)
    n = length(vars)
    M = statistics(vars, G, D)
    α = prior(vars, G)
    return sum(bayesian_score_component(M[i], α[i]) for i in 1:n)
end

struct Variable
    name::Symbol
    r::Int # number of possible values
end


function sub2ind(siz, x)
    k = vcat(1, cumprod(siz[1:end-1]))
    return dot(k, x .- 1) + 1
end
```

```
function statistics(vars, G, D::Matrix{Int})
    n = size(D, 1)
    r = [vars[i].r for i in 1:n]
    q = [prod([r[j] for j in inneighbors(G,i)]) for i in 1:n]
    M = [zeros(q[i], r[i]) for i in 1:n]
    for o in eachcol(D)
        for i in 1:n
            k = o[i]
            parents = inneighbors(G,i)
            j = 1
            if !isempty(parents)
                j = sub2ind(r[parents], o[parents])
            end
            M[i][j,k] += 1.0
        end
    end
    return M
end


struct K2Search
    ordering::Vector{Int} # variable ordering
end

function fit(method::K2Search, vars, D)
    G = SimpleDiGraph(length(vars))
    for (k,i) in enumerate(method.ordering[2:end])
        y = bayesian_score(vars, G, D)
        while true
            y_best, j_best = -Inf, 0
            for j in method.ordering[1:k]
                if !has_edge(G, j, i)
                    add_edge!(G, j, i)
                    y' = bayesian_score(vars, G, D)
                    if y' > y_best
                        y_best, j_best = y', j
                    end
                    rem_edge!(G, j, i)
                end
            end
            if y_best > y
                y = y_best
                add_edge!(G, j_best, i)
            else
                break
```

```
            end
        end
    end
    return G
end



function 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

    # read in data
    data = CSV.File(infile) |> DataFrame
    # show(data)
    D = Matrix(Matrix(values(data))')

    # construct vars
    variable_names = names(data)
    vars = []
    for v in variable_names
        var = Variable(Symbol(v), maximum(data[!, v]))
        push!(vars, var)
    end

    # do k2 search
    ordering=1:length(vars)
    method = K2Search(ordering)
    G = fit(method, vars, D)

    # write out graph
    idx2names = Dict((i, vars[i].name) for i in 1:length(vars))
    write_gph(G, idx2names, outfile)
    # p = plot(G, variable_names) # create TikZ plot with labels
    # save(PDF(outfile*".pdf"), p) # save TikZ as PDF
    p = gplot(G; nodelabel=variable_names, layout=circular_layout)
    draw(PNG(outfile*".png"),p)

    # Print out final Bayesian score
    println("Final Bayesian Score: ", bayesian_score(vars,G,D))
end
```

```
if length(ARGS) != 2
    error("usage: julia project1.jl <infile>.csv <outfile>.gph")
end

inputfilename = ARGS[1]
outputfilename = ARGS[2]

@time begin
    compute(inputfilename, outputfilename)
end
```