Project 1: Bayesian Structure Learning

Talia Blum taliab@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

For the algorithm, I used a combination of two algorithms, k2 search and a modified local directed graph search.

First, I used K2 search to find a graph structure. I used Bayesian score as the scoring function, and gave the search a random ordering of variables. The algorithm creates a graph by iterating over the variables according to the random ordering and greedily adds parents to the nodes in a way that maximally increases the Bayesian score.

Then, the graph outputted by the k2 search is used as the starting point for a modified local directed graph search. The idea for this graph search was inspired by simulated annealing. I generated a random neighbor of the current graph, and move to it as the next graph if its Bayesian score is an improvement over the current graph or with probability

$$\frac{1}{5}e^{-k},$$

where k is the current step of the algorithm. This was run for 10000 steps. This ran in polynomial time. Runtimes for each dataset are listed in graph captions.

2. Graphs

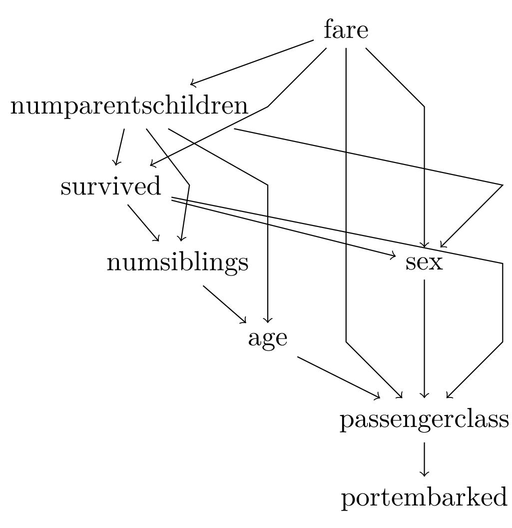

Figure 1: Graph structure for small dataset. This graph structure took 20 seconds to compute.

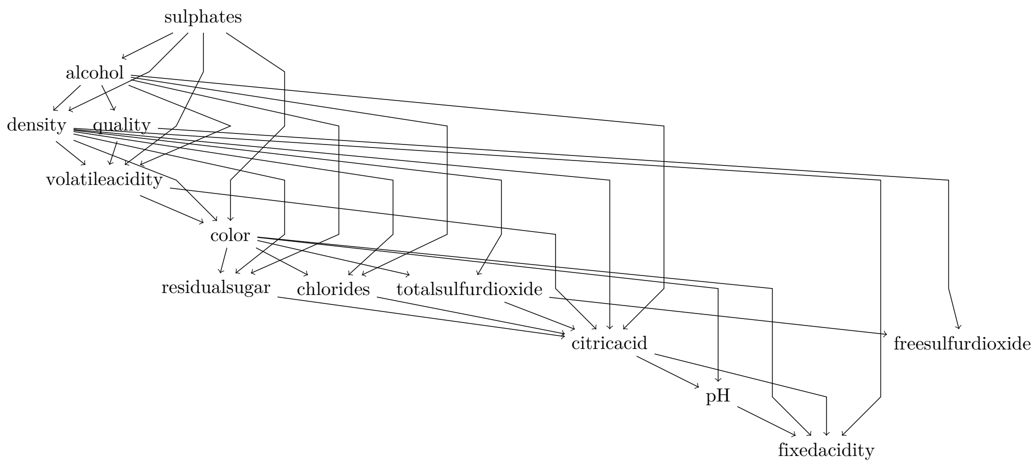

Figure 2: Graph structure for medium dataset. This graph structure took 2 minutes and 22 seconds to compute.

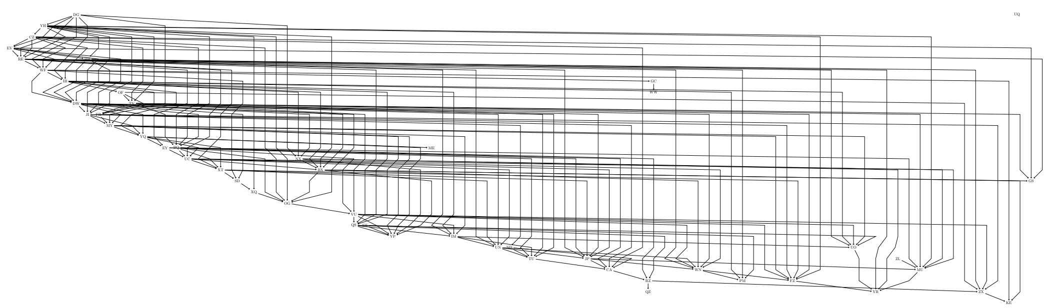

Figure 3: Graph structure for large dataset. This graph structure took about 17 minutes to compute.

3. Code

```
\begin{algorithm}
\begin{lstlisting}[language=Python]
using Graphs
using Printf
using CSV
using DataFrames
using Random
using SpecialFunctions
using LinearAlgebra
using GraphPlot
using Compose
using Cairo, Fontconfig
```

```
using TikzGraphs, TikzPictures

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

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

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

function bayesian_score(vars, G, D)
    n = length(vars)
    M = statistics(vars, G, D)
    alpha = prior(vars, G)
    return sum(bayesian_score_component(M[i], alpha[i]) for i in 1:n)
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_prime = bayesian_score(vars, G, D)
                    if y_prime > y_best
                        y_best, j_best = y_prime, 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

struct LocalDirectedGraphSearch
    G # initial graph
```

```
    k_max # number of iterations
end

function rand_graph_neighbor(G)
    n = nv(G)
    i = rand(1:n)
    j = mod1(i + rand(2:n)-1, n)
    G_prime = copy(G)
    has_edge(G, i, j) ? rem_edge!(G_prime, i, j) : add_edge!(G_prime, i, j)
    return G_prime
end

function fit(method::LocalDirectedGraphSearch, vars, D)
    G = method.G
    y = bayesian_score(vars, G, D)
    half_k = method.k_max / 2
    for k in 1:method.k_max
        G_prime = rand_graph_neighbor(G)
        y_prime = is_cyclic(G_prime) ? -Inf : bayesian_score(vars, G_prime, D
    )
        if y_prime > y || rand(Float64) < .02*exp(-k)
            y, G = y_prime, G_prime
        end
    end
    return G
end

function drawgraph(G, idx2names, outfile)
    outputfile_pdf = string(outfile[1:end-4])
    # draw(PDF(outputfile_pdf, 16cm, 16cm), gplot(G, nodelabel=idx2names))
    t = TikzGraphs.plot(G, idx2names)
    TikzPictures.save(TikzPictures.PDF(outputfile_pdf), t)
end

function compute(infile, outfile)
    Data = CSV.read(infile, DataFrame)
    varinfo = describe(Data,:max)
    vars = [Variable(varinfo.variable[i], varinfo.max[i]) for i in 1:length(
    varinfo.variable)]

    D = Matrix(transpose(Matrix(Data)))

    nvars = length(vars)
    ordering = shuffle(collect(1:nvars))
    varnames = names(Data)

    k2_init = K2Search(ordering)
    G = fit(k2_init, vars, D)

    score = bayesian_score(vars, G, D)
```

```
    println("Score after k2 search:", score)

    ldgs_init = LocalDirectedGraphSearch(G, 10000)
    G = fit(ldgs_init, vars, D)

    idx2names = varnames

    score = bayesian_score(vars, G, D)
    println("Score after graph search:", score)

    write_gph(G, idx2names, outfile)
    drawgraph(G, idx2names, outfile)
end

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

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

compute(inputfilename, outputfilename)
```