Project 1: Bayesian Structure Learning

Etienne Jacquot ejacquot@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

In this implementation, the simple hill-climbing algorithm was used. This local search algorithm starts with an initial empty graph and then moves to the highest-scoring neighbor. At each step, the algorithm iteratively explores different local modifications, such as adding an edge provided it doesn't introduce a cycle. The improvement between graphs is calculated by comparing the current score with the current best score, and it stops iterating when no improvements are found. Each graph is therefore scored using the Bayesian score formula. If the new score is found to be better, the new edge is kept and if not, it is removed. The process repeats for a maximum number of iterations to avoid running for too long. This iterative yet computationally efficient approach was found to produce better-scoring structures than the K2 algorithm with the provided data while maintaining reasonable computation time. Alternative search strategies (including genetic algorithm) were considered and tested out but found to be prohibitively expensive.

In the figures below, the plotted `.gph` files are shown together with their total runtime that includes the plotting time of approximately 11 seconds on average. The final code used is also attached at the end of this document. All experiments were run on macOS 15.0 with Julia 1.10, runtimes are wall-clock measurements.

2. Graphs

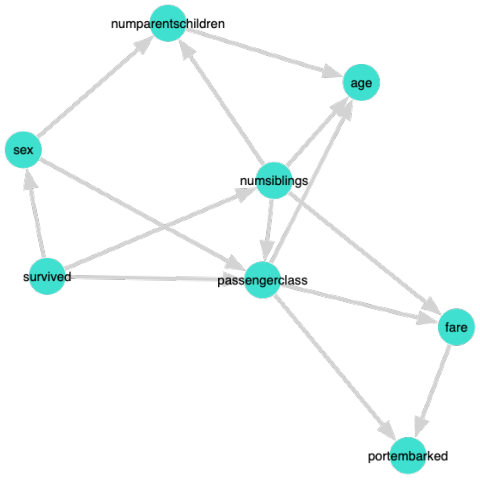

Figure 1: `small.csv` structure, total runtime: 15s

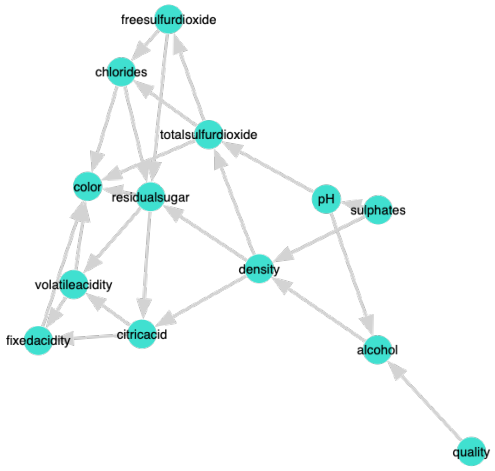

Figure 2: `medium.csv` structure, total runtime: 18s

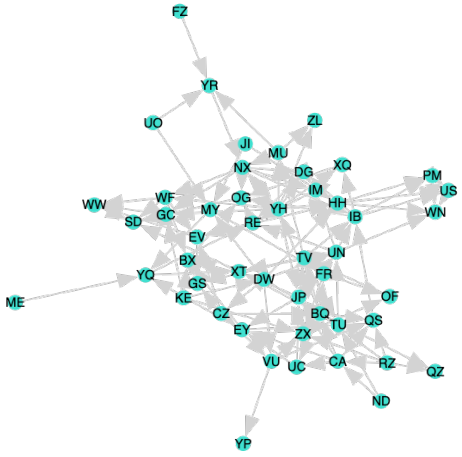

Figure 3: `large.csv` structure, total runtime: 249s

3. Code

```
using CSV, DataFrames, Graphs, StatsBase, SpecialFunctions, Random, Printf,
    Dates
using GraphPlot, Colors, Cairo, Fontconfig, Compose

function write_gph(dag::DiGraph, idx2names, filename)
    open(filename, "w") do io
        for e in edges(dag)
            println(io, "$(idx2names[src(e)]),$(idx2names[dst(e)])")
```

```
        end
    end
    println("Wrote graph to $filename")
end

# Bayesian Scoring

function bayesian_score_component(M::Matrix{Float64}, a::Matrix{Float64})
    p  = sum(loggamma.(a .+ M))
    p -= sum(loggamma.(a))
    p += sum(loggamma.(sum(a, dims=2)))
    p -= sum(loggamma.(sum(a, dims=2) .+ sum(M, dims=2)))
    return p
end

function bayesian_score(vars, G::DiGraph, D::DataFrame; a_total=1.0)
    levels = Dict(v => unique(D[:, v]) for v in vars)
    score = 0.0
    for (i, vname) in enumerate(vars)
        parents = [vars[p] for p in inneighbors(G, i)]
        vals = levels[vname]
        r_i = length(vals)
        groups = isempty(parents) ? [D] : groupby(D, parents)
        q_i = length(groups)
        a = fill(a_total / (q_i * r_i), q_i, r_i)
        M = zeros(Float64, q_i, r_i)
        for (j, g) in enumerate(groups)
            counts = countmap(g[:, vname])
            for (k, v) in enumerate(vals)
                M[j, k] = get(counts, v, 0)
            end
        end
        score += bayesian_score_component(M, a)
    end
    return score
end

# Simple Hill Climbing

function hillclimb(vars, D; a_total=1.0, max_iter=10)
    n = length(vars)
    G = SimpleDiGraph(n)
    best = bayesian_score(vars, G, D; a_total)
    println("Initial score = $(round(best, digits=2))")

    for iter in 1:max_iter
        improved = false
        for (i, j) in Iterators.product(1:n, 1:n)
            i == j && continue
            if !has_edge(G, i, j) && !has_path(G, j, i)
```

```
                add_edge!(G, i, j)
                s = bayesian_score(vars, G, D; a_total)
                if s > best
                    best = s
                    improved = true
                else
                    rem_edge!(G, i, j)
                end
            end
        end
        println("Iter $iter: score=$(round(best, digits=2)), edges=$(ne(G))")
        if !improved
            println("No improvement: stopping early.")
            break
        end
    end
    return G, best
end

# Graph Plotting

function read_gph(filename)
    edges = []
    open(filename, "r") do f
        for line in eachline(f)
            line = strip(line)
            isempty(line) && continue
            src, dst = split(line, ",")
            push!(edges, (strip(src), strip(dst)))
        end
    end
    return edges
end

function build_graph(edges)
    nodes = unique(vcat([e[1] for e in edges], [e[2] for e in edges]))
    name_to_id = Dict(n => i for (i, n) in enumerate(nodes))
    g = DiGraph(length(nodes))
    for (src, dst) in edges
        add_edge!(g, name_to_id[src], name_to_id[dst])
    end
    return g, nodes
end

function plot_gph(infile::String; outfile::String)
    edges = read_gph(infile)
    g, nodes = build_graph(edges)
    p = gplot(g; layout=spring_layout, nodelabel=nodes)
    draw(SVG(outfile * ".svg", 16cm, 16cm), p)
    println("Saved DAG plot to $(outfile).svg")
```

```
end

# Main

function compute(infile::String, outfile::String)
    total_start = time()
    println("Loading dataset: $infile")
    df = CSV.read(infile, DataFrame)
    vars = names(df)
    Random.seed!(1234)

    a = 1.0

    println("\nStarting hill-climb search...")
    t_hc_start = time()
    G, s = hillclimb(vars, df; a_total=a, max_iter=5)
    t_hc = time() - t_hc_start
    println("Hill-climb completed in $(round(t_hc, digits=3)) s")

    println("\nFinal score = $(round(s, digits=2)), edges = $(ne(G))")

    idx2names = Dict(i => vars[i] for i in 1:length(vars))
    write_gph(G, idx2names, outfile)

    t_plot_start = time()
    plot_gph(outfile; outfile=outfile)
    t_plot = time() - t_plot_start

    total_time = time() - total_start
    println("\nRuntime summary:")
    println("  Hill-climb : $(round(t_hc, digits=3)) s")
    println("  Plotting   : $(round(t_plot, digits=3)) s")
    println("  Total time : $(round(total_time, digits=3)) s")
    println("$(outfile).gph and $(outfile).svg")
end

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

compute(ARGS[1], ARGS[2])
```