Decision Making Under Uncertainty

Project 1: Bayesian Network Learning

Wei-Lin Pai, wpai@stanford.edu

1. Algorithm Description

In my implementation of the Bayesian network learning, I choose the K2 Search algorithm to find the possible network structure. The reason I choose K2 Search is based on the following reasons:

1. It begins with a graph with no directed edges, which is straightforward to implement
2. It runs in polynomial time
3. Greedily adding parents

However, it cannot guarantee a globally optimal solution, so I iterate K2 Search through different initial ordering. This part is implemented in iterative_K2_fit(), which essentially run K2 Search function K2_fit() multiple times with random initial ordering. The iteration terminates when the iteration times exceed the maximum iteration (max_iter) time or the improvement of the Bayesian score is lower than the improvement threshold (improvement_threshold).

In the K2 search algorithm, the most essential function is calculating the Bayesian score, and here, I use equation 5.5 from the textbook to calculate the Bayesian Score in function Bayesian_score(). Assuming prior equal to 1, the alpha is calculated in compute_a(). Another unknown in the log Bayesian Score function is m, and it is calculated in compute_m(). One thing to notice is that the DataFrame datatype needs to be converted into an Integer Matrix before doing a row broadcast in Julia; the df_to_matrix() achieves this goal.

Pseudo Code for Iterative K2 Search:

```
Iterative_K2_Search(Data, max_iter, improve_threshold):
        s =  shuffle_order()
        Graph = K2_fit(s,Data)
        if(iter > max_iter || improve< improve_threshold):
        break
        return G_best, Score_best
```

Below is the Bayesian score and their run time I test for 1, 10, 100 iteration:

B_Score/runtime	1 iteration	10 iterations	100 iterations
small.csv	-3839.05 (0.062s)	-3818.33 (0.152s)	-3801.09 (2.984s)
medium.csv	-96945.66 (1.479s)	-97333.55 (18.31s)	-96652.33 (155.592s)
large.csv	-426687.06 (158.532 s)		

I didn't run an iterative K2 Search for large.csv; this is because it will take too much time; the reason should be that I do not set the upper bound on the number of parents, so the structure becomes complex. However, K2 Search is a greedy algorithm, so a one-time search will have guaranteed output.

Data preprocess include calculating the number of instantiations for each node, and I use max() in each row of DataFrame to find out in construct_vars(). Also, I customize a struct Node with names{String} and r{Int} attributes.

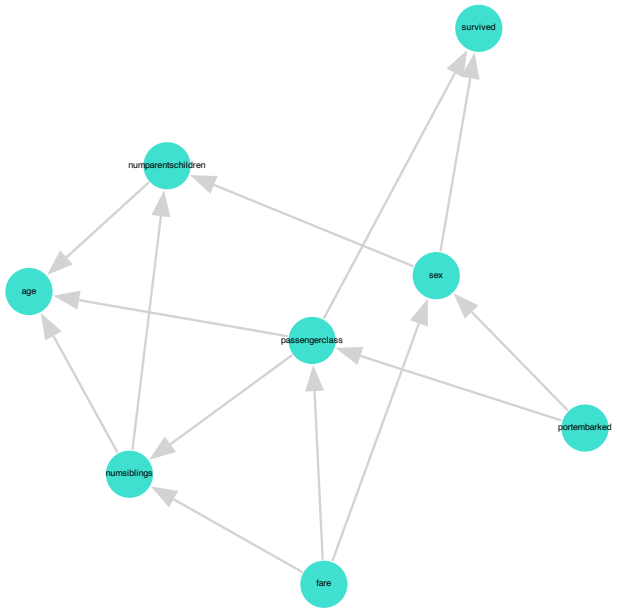

Figure 1. small.gph (1 Iteration)

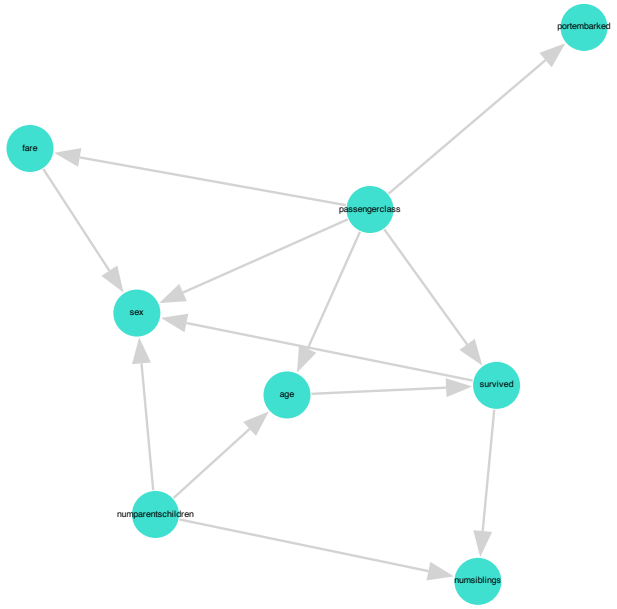

Figure 2. small.gph (10 Iterations)

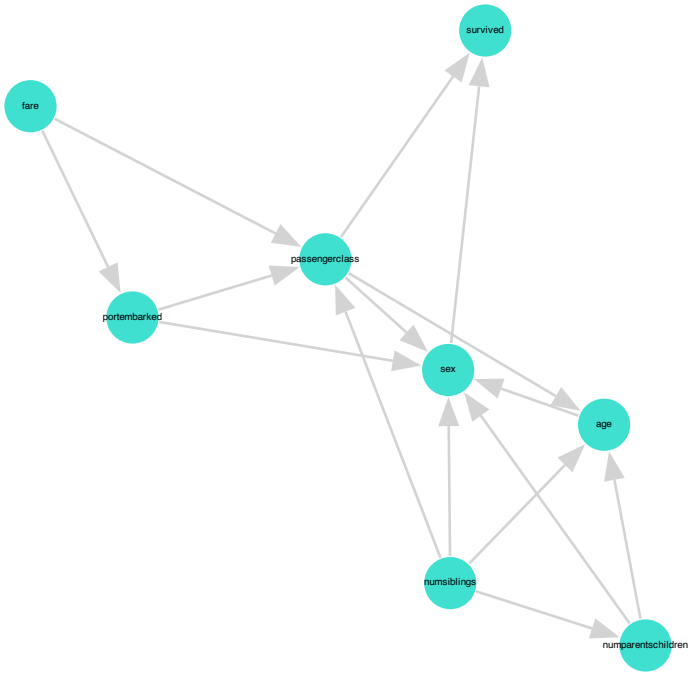

Figure 3. small.gph (100 Iterations)

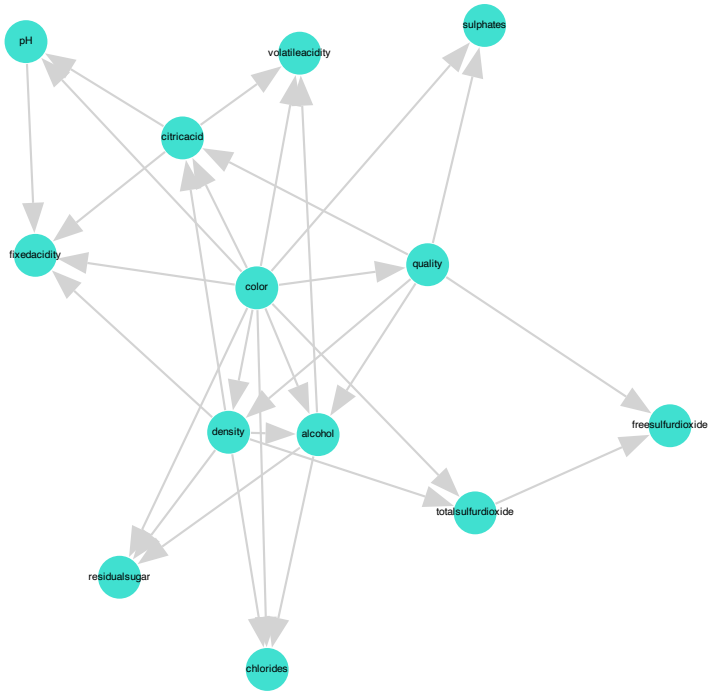

Figure 4. medium.gph (1 Iteration)

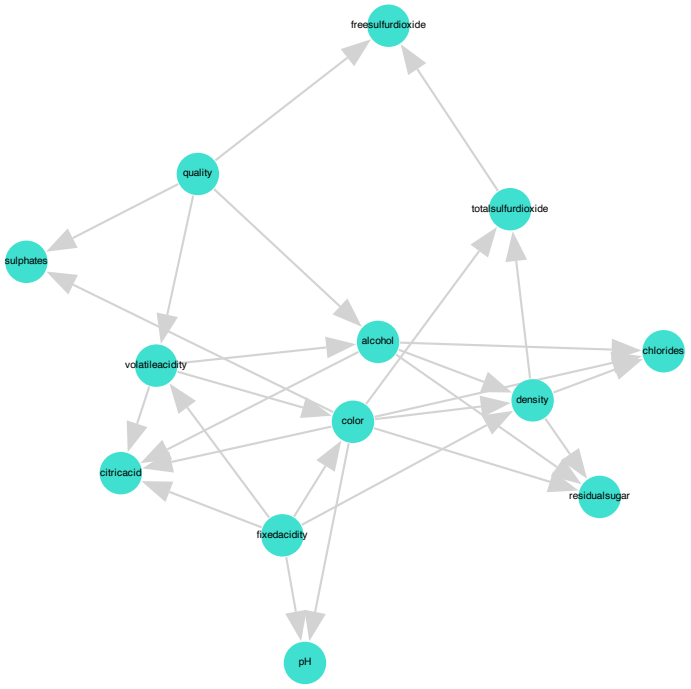

Figure 5. medium.gph (10 Iterations)

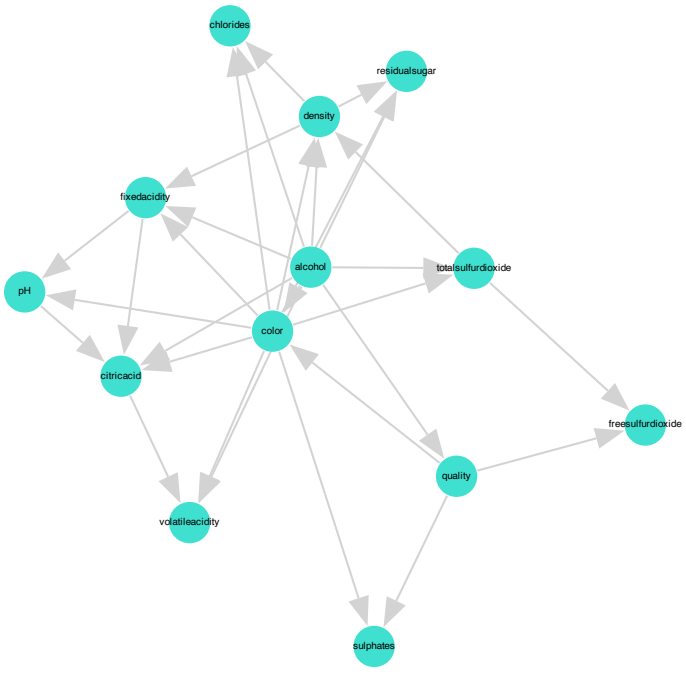

Figure 6. medium.gph (100 Iterations)

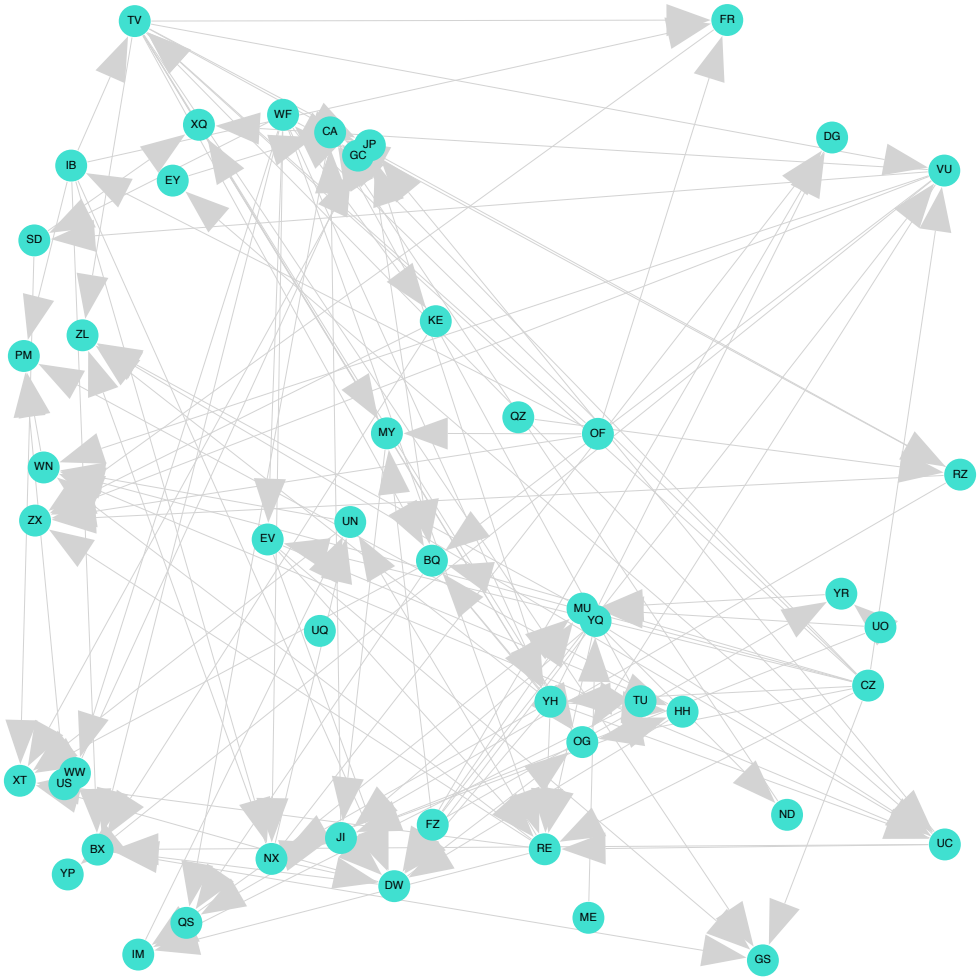

Figure 7. large.gph (1 Iteration)

Project.jl

```
using Printf
using Pkg
Pkg.add("DataFrames")
Pkg.add("CSV")
Pkg.add("Graphs")
Pkg.add("GraphPlot")
Pkg.add("Compose")
Pkg.add("Cairo")
Pkg.add("Fontconfig")
Pkg.add("SpecialFunctions")
using DataFrames
using CSV
```

```
using Graphs
using LinearAlgebra
using GraphPlot
using Compose, Cairo, Fontconfig
using Random
using SpecialFunctions
using Profile

mutable struct Node
    names::String
    r::Int
end

# Establish r for every node
function construct_vars(df)
    node_names = names(df)
    n = length(node_names)
    vars = Node[]
    for i in 1:n
        r = maximum(df[:,i])
        names = node_names[i]
        push!(vars,Node(names,r))
    end
    return vars
end

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

function compute_a(vars,G)
    n = length(vars)
    #   r = # instantiation of X_i
    r = [vars[i].r for i in 1:n]
    # q = # instantiation of Parent Nodes
    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

# Convert DataFrame to Integer Matrix
function df_to_matrix(df::DataFrame)
    return Matrix{Int}(df)
end

function compute_m(vars, G, D::DataFrame)
```

```
    D_matrix = df_to_matrix(D)
    return compute_m(vars, G, D_matrix)
end

function compute_m(vars, G, D::Matrix{Int})
    n = size(D,2)
    r = [vars[i].r for i in 1:n]  #   r = # instantiation of X_i
    q = [prod([r[j] for j in inneighbors(G,i)]) for i in 1:n] # q = # instantiation of
Parent Nodes
    m = [zeros(q[i],r[i]) for i in 1:n]
    for o in eachrow(D)
        for i in 1:n
            k = o[i]
            parents = inneighbors(G,i)
            j = 1 # if no parents, assume 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,a)
    p = sum(loggamma.(m+a))
    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, D::DataFrame)
    D_matrix = df_to_matrix(D)
    return bayesian_score(vars, G, D_matrix)
end

function bayesian_score(vars,G,D)
    n = length(vars)
    # Compute statistic
    m = compute_m(vars,G,D)
    # Compute prior
    a = compute_a(vars,G)
    return sum(bayesian_score_component(m[i],a[i]) for i in 1:n)
end

struct K2Search
```

```
    ordering::Vector{Int}
end

function K2_fit(method::K2Search,vars,D)
    G = SimpleDiGraph(length(vars))
    for (k,i) in enumerate(method.ordering[2:end])
        # Calculate bayesian score
        y = bayesian_score(vars,G,D)
        while true
            y_best, j_best = -Inf, 0
            for j in method.ordering[1:k]
                # Try add adge
                if !has_edge(G,j,i)
                    add_edge!(G,j,i)
                    y_h = bayesian_score(vars,G,D)
                    if y_h > y_best
                        y_best,j_best = y_h,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 iterative_K2_fit(vars,D,max_iter,improvement_threshold)
    # Set up initial G, score
    best_G = nothing
    best_score = -Inf

    for iter in 1: max_iter
        # Shuffle random order for search
        order = shuffle(1:length(vars))
        method = K2Search(order)
        G = K2_fit(method,vars,D)
        b_score = bayesian_score(vars,G,D)

        if b_score > best_score
            # Break if improve is slow
            if (iter > 1 && (b_score-best_score)/abs(best_score) < improvement_threshold)
```

```
                best_score = b_score
                best_G = G
                println("Iter"*string(iter)*": "*string(best_score))
                break
            end
            best_score = b_score
            best_G = G
            println("Iter"*string(iter)*": "*string(best_score))
        end
    end
    return best_G, best_score
end

"""
    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 compute(infile, outfile)

    # Set timer
    total_time = @elapsed begin
        # Read CSV
        df = CSV.read(infile,DataFrame)
        # Setup vars
        vars = construct_vars(df)
        # set up idx to name
        idx2names = Dict(i => vars[i].names for i in 1:length(vars))
        # Set up timer for iterative K2 Search
        learning_time = @elapsed begin
            # iterative K2 Search
            G,score = iterative_K2_fit(vars,df,1,0.0001)
        end
        # Plot graph
        node_labels = [idx2names[i] for i in 1:nv(G)]
        p = gplot(G; nodelabel=node_labels, NODELABELSIZE=10.0,layout=spring_layout)
```

```
        draw(PDF(string(outfile)*".pdf", 100cm, 100cm), p)
        # Write gph
        write_gph(G,idx2names,outfile)
    end
    println("Final Score: "*string(score))
    println("Structure learning time: $(round(learning_time, digits=3)) seconds")
    println("Total runtime: $(round(total_time, digits=3)) seconds")

end

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

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

    compute(inputfilename, outputfilename)
end

main()
# @profile main()
# Profile.print()
```