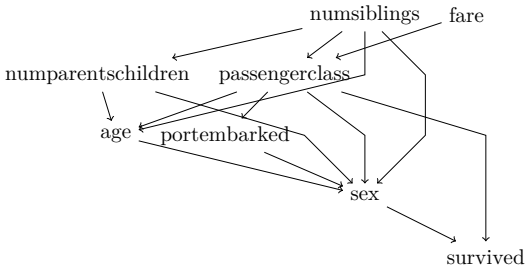
Caleb Logemann
AERE 504 Intelligent Air Systems
Project 1

1 Description of Methods

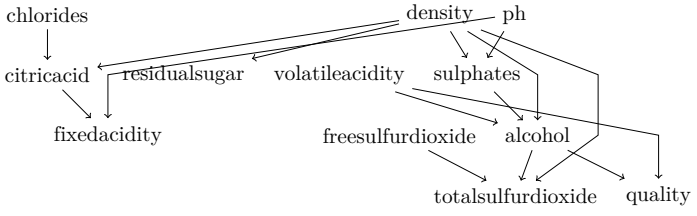
For this project I implemented both the k2 search algorithm and the local search algorithm. The k2 algorithm starts with a ordering of the variables and then generates a Bayesian Network structure that maximizes the Bayesian score and such that the ordering is a topological sort of the nodes. The local search algorithm starts from a random structure and then takes local moves to maximize the Bayesian score.

For my full search algorithm I used a combination of both of these algorithms. I first generated a random ordering of variables and then applied the k2 algorithm. I then used the local search on the result of the k2 search. This guarantees that I am at a local maximum and can only improve upon the restult of the k2 algorithm. I also used a randomized restarting process to more fully search the space and to make sure the algorithm didn't get stuck at a poor local maximum.

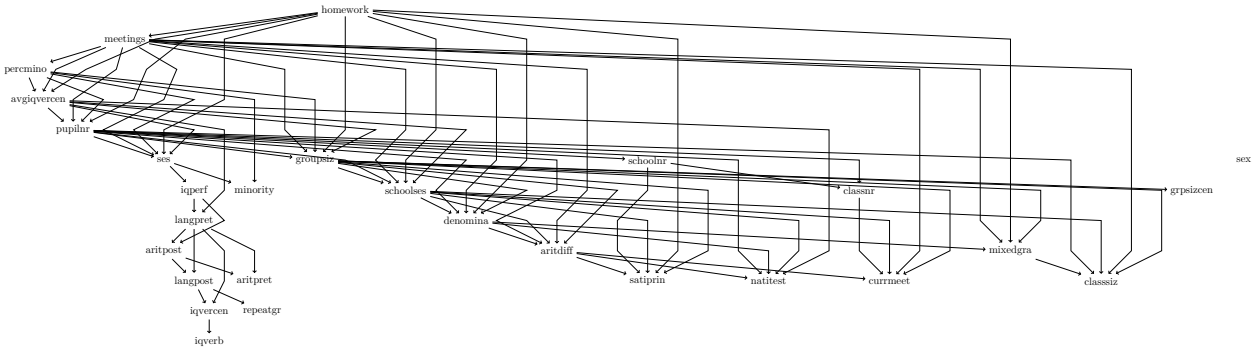
2 Graphs and Scores



The Bayesian Score for this structure is . It took XXXXX seconds to compute a 1000 random restarts.



The Bayesian Score for this structure is . It took XXXXX seconds to compute a 500 random restarts.



The Bayesian Score for this structure is . It took XXXXX seconds to compute a 100 random restarts. The Bayesian Score for this structure is . It took XXXXX seconds to compute a 1000 random restarts.

```
using CSV, DataFrames, Distributions, BayesNets, SpecialFunctions, LightGraphs
using TikzGraphs, TikzPictures, Printf

include("BayesianNetworks.jl")

function compute_r(dt, nVariables)
    r = zeros(Int64, nVariables);
    for i in 1:nVariables
        r[i] = maximum(dt[:,i]) - minimum(dt[:,i]) + 1;
    end
    return r;
end

function save_graph(g, filename, variables)
    filename = filename*".pdf"
    t = plot(g, map(string, variables));
    save(PDF(filename), t);
end

function save_graph_file(g, filename, variables)
    filename = filename*".gph"
    file = open(filename, "w")

    for edge in LightGraphs.edges(g)
        @printf(file,"%s,%s\n",variables[LightGraphs.src(edge)], variables[
            LightGraphs.dst(edge)])
    end

    close(file)
end
```

```
titanicData = CSV.read("titanic.csv");
whitewineData = CSV.read("whitewine.csv");
schoolgradesData = CSV.read("schoolgrades.csv");
structuredlearningData = CSV.read("structuredlearning_test.csv");

dfArray = [titanicData, whitewineData, schoolgradesData, structuredlearningData]
    ;
nRestartsArray = [1000, 500, 100, 1000];
filenames = ["titanic", "whitewine", "schoolgrades", "structuredlearning_test"]

for i = 1:4
    df = dfArray[i];
    nRestarts = nRestartsArray[i];
    dt = convert(Array, df);
    nVariables = size(dt, 2);
    r = compute_r(dt, nVariables);
    @time (g, max_score) = full_search(g->BayesianScore(g, df), nRestarts,
        nVariables);
    println(filenames[i]);
    println(max_score);
    save_graph(g, filenames[i], names(df));
    save_graph_file(g, filenames[i], names(df));
end
```

```
using CSV, DataFrames, Distributions, BayesNets, SpecialFunctions, LightGraphs
using TikzGraphs, TikzPictures, Random

function BayesianScore(graph, df)
    BayesNets.bayesian_score(graph, names(df), df)
end

function local_search(scoring_function, graph_0)
    g = graph_0;
    max_score = scoring_function(g);
    nVariables = nv(g);
    g_max_score = g;
    has_updated = true;
    while (has_updated)
        g = g_max_score;
        has_updated = false;
        for i = 1:nVariables
            for j = i:nVariables
                if (has_edge(g, i, j))
                    # remove edge
                    rem_edge!(g, Edge(i, j));
                    fg = scoring_function(g);
                    if(fg > max_score)
                        has_updated = true;
                        max_score = fg;
                        g_max_score = g;
                    end

                    # try switching edge direction
                    add_edge!(g, Edge(j, i))
```

```
                if(!is_cyclic(g))
                    fg = scoring_function(g);
                    if(fg > max_score)
                        has_updated = true;
                        max_score = fg;
                        g_max_score = g;
                    end
                end
                rem_edge!(g, Edge(j, i));
                add_edge!(g, Edge(i, j));
            elseif (has_edge(g, j, i))
                # remove edge
                rem_edge!(g, Edge(j, i))
                fg = scoring_function(g);
                if(fg > max_score)
                    has_updated = true;
                    max_score = fg;
                    g_max_score = g;
                end

                # try add edge other direction
                add_edge!(g, Edge(i, j))
                if(!is_cyclic(g))
                    fg = scoring_function(g);
                    if(fg > max_score)
                        has_updated = true;
                        max_score = fg;
                        g_max_score = g;
                    end
                end
                rem_edge!(g, Edge(i, j));
                add_edge!(g, Edge(j, i));
            else
                # try adding edge i \to j
                add_edge!(g, Edge(i, j))
                if(!is_cyclic(g))
                    fg = scoring_function(g);
                    if(fg > max_score)
                        has_updated = true;
                        max_score = fg;
                        g_max_score = g;
                    end
                end
                rem_edge!(g, Edge(i, j));

                # try adding edge j \to i
                add_edge!(g, Edge(j, i))
                if(!is_cyclic(g))
                    fg = scoring_function(g);
                    if(fg > max_score)
                        has_updated = true;
                        max_score = fg;
                        g_max_score = g;
                    end
                end
                rem_edge!(g, Edge(j, i));
            end
        end
```

```
        end
    end
    return (g_max_score, max_score);
end

function k2search(scoring_function, variables, max_parents)
    nVariables = size(variables, 1);
    g = SimpleDiGraph(nVariables);
    max_score = scoring_function(g);
    for i in 1:nVariables
        child = variables[i];
        has_updated = true;
        nParentsAdded = 0;
        while (has_updated && nParentsAdded < max_parents)
            has_updated = false;
            max_new_parent = 0;
            for j in 1:i-1
                parent = variables[j]
                if (!has_edge(g, parent, child))
                    add_edge!(g, parent, child);
                    fg = scoring_function(g);
                    if(fg > max_score)
                        has_updated = true;
                        max_new_parent = parent;
                        max_score = fg;
                    end
                    rem_edge!(g, parent, child);
                end
            end
            if (has_updated)
                # update g
                add_edge!(g, max_new_parent, child)
                nParentsAdded = nParentsAdded+1;
            end
        end
    end
    return (g, max_score);
end

# search function using k2 search and local search with some
# random restarting
function full_search(scoring_function, nRestarts, nVariables)
    g = SimpleDiGraph(nVariables);
    max_score = scoring_function(g);
    n_updates = 0;
    for n = 1:nRestarts
        # randomly shuffle variable order, give different results in k2 search
        variables = shuffle(1:nVariables)
        #println(variables);
        max_parents = convert(Int64, floor(n/(nRestarts/7) + 1))
        #max_parents = min(10, nVariables);
        #println(max_parents)
        #max_parents = 5;
        (temp_g, temp_max_score) = k2search(scoring_function, variables,
             max_parents);
        (temp2_g, temp2_max_score) = local_search(scoring_function, temp_g);
        if(temp2_max_score > max_score)
            n_updates = n_updates+1;
```

```
            println(n)
            max_score = temp2_max_score;
            g = temp2_g;
        end
    end
    #println(n_updates);
    return (g, max_score);
end
```