Dynamic Programming for Min Max Search
in Complete Multipartite Graph

Complete multipartite graph

A graph is a set of vertex and edge, i.e. $G = (V, E)$. A bipartite graph is a graph having vertices divided into two disjoint sets $V = (V_1, V_2)$, such that for all edges in the graph, it must have a node from V_1 and a node from V_2, i.e. for all edge $e \in E$, such that $e = (\upsilon_1, \upsilon_2)$, where $\upsilon_1 \in V_1$ and $\upsilon_2 \in V_2$. A complete bipartite graph is a bipartite graph such that for a node from V_1 and a node from V_2, then there exists an edge joining them, while there is no edge between nodes from the same vertex subset, i.e. given vertex $\upsilon_1 \in V_1$ and $\upsilon_2 \in V_2$, $\exists e \in E$, such that $e = (\upsilon_1, \upsilon_2)$. We can then generalize this concept to complete multipartite graph. Suppose all vectices can be divided into N subsets, i.e. $V = (V_1, V_2, V_3, \ldots V_N)$, then for any two vertices, each from a neighbouring subsets, from an edge, while there is no edge between nodes from the same vertex subset.

Min Max problem

In an application, a path P_i is defined as $(\upsilon_{1,i1}\ \upsilon_{2,i2},\ \upsilon_{3,i3},\ \ldots\ \upsilon_{N,iN})$, where vertice $\upsilon_{n,in} \in V_n\ \forall n \in [1,N]$, then is there any efficient algorithm (other than exhaustive search) for solving the following :

$$\min_{P_i} \max_{n\in[1,N-1]_i} f(v_{n,i_n}, v_{n+1,i_{n+1}})$$

where f is a function of 2 variables.

$$= \min_{P_i} g(P_i)$$

For simplicity, we denote the maximum value of function f along path P_i as $g(P_i)$. Lets consider the special case when there is only one vertex in the last partition, i.e. $V_N = (\upsilon_{N,1})$, while there are M vertices in the N-1 partition, i.e. $V_{N-1} = (\upsilon_{N-1,1}\ \upsilon_{N-1,2}\ \upsilon_{N-1,3},\ \ldots\ \upsilon_{N-1,M})$. Suppose all possible paths starting from any vertex in partition 1 and reaching vertex $\upsilon_{N-1,m}$ in partition N-1 form the set $(Q_{m,1},\ Q_{m,2},\ \ldots,\ Q_{m,km})$. We can break down this graph optimization problem into subproblems.

$$\min_{P_i} \max_{n\in[1,N-1]_i} f(v_{n,i_n}, v_{n+1,i_{n+1}})$$

$$= \min_{m\in[1,M],k\in[1,K_m]} \max(g(Q_{m,k}), f(v_{N-1,m}, v_{N,1}))$$

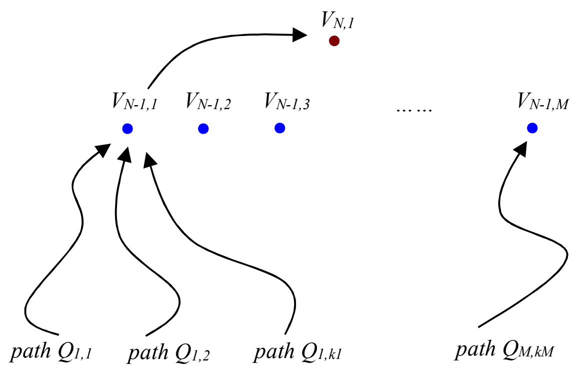
$$= \min_{m\in[1,M]} \left[\min_{k\in[1,K_m]} \max(g(Q_{m,k}), f(v_{N-1,m}, v_{N,1})) \right]$$

$$= \min_{m\in[1,M]} \left[\min \begin{bmatrix} \max(g(Q_{m,1}), f(v_{N-1,m}, v_{N,1})) \\ \max(g(Q_{m,2}), f(v_{N-1,m}, v_{N,1})) \\ \ldots \\ \max(g(Q_{m,k_m}), f(v_{N-1,m}, v_{N,1})) \end{bmatrix} \right]$$

$$= \min_{m\in[1,M]} \max \left[f(v_{N-1,m}, v_{N,1}), \min \begin{bmatrix} g(Q_{m,1}), \\ g(Q_{m,2}) \\ \ldots \\ g(Q_{m,k_m}) \end{bmatrix} \right]$$

by making use of the min-max property in equation 1

$$= \min_{m\in[1,M]} \max(f(v_{N-1,m}, v_{N,1}), f_{optimum}(v_{N-1,m}))$$



where $f_{optimum}$ is defined as the optimum of subgraph with ending vertex $V_{N-1,m}$.

$$f_{optimum}(v_{N-1,m}) = \min \begin{bmatrix} g(Q_{m,1}), \\ g(Q_{m,2}) \\ \ldots \\ g(Q_{m,k_m}) \end{bmatrix}$$

Thus this problem can be solved by dynamic programming.

Suppose the number of vertices of a quad-partite graph are N, M, K, L respecitively, then the computational load for exhaustive search and dynamic programming respectively are :

computation load of exhaustive search = $N \times M \times K \times L$

computation load of dynamic programming = $N \times M + M \times K + K \times L$

Min Max property

Lets derive the following property :

$$\min(\max(x, y_1), \max(x, y_2), \ldots, \max(x, y_N)) \quad = \quad \max(x, \min(y_1, y_2, \ldots, y_N)) \qquad \text{(equation 1)}$$

Consider case N = 2

	$\max(x,y)$	$\max(x,z)$	$\min(\max(x,y),\max(x,z))$	$\min(y,z)$	$\max(x,\min(y,z))$
when $x<y<z$	y	z	y	y	y
when $x<z<y$	y	z	z	z	z
when $y<x<z$	x	z	x	y	x
when $y<z<x$	x	x	x	y	x
when $z<x<y$	y	x	x	z	x
when $z<y<x$	x	x	x	z	x

hence we have : $\min(\max(x,y),\max(x,z)) \quad = \quad \max(x,\min(y,z))$

Now suppose it is true for case N-1, now consider :

$$\begin{aligned}
& \min\big(\max(x,y_1),\max(x,y_2),\ldots,\max(x,y_N)\big) & \\
= \; & \min\big(\min(\max(x,y_1),\max(x,y_2),\ldots,\max(x,y_{N-1})),\max(x,y_N)\big) & \text{apply case N-1} \\
= \; & \min\big(\max(x,\min(y_1,y_2,\ldots,y_{N-1})),\max(x,y_N)\big) & \\
= \; & \max\big(x,\min(\min(y_1,y_2,\ldots,y_{N-1}),y_N)\big) & \text{apply case 2} \\
= \; & \max\big(x,\min(y_1,y_2,\ldots,y_{N-1},y_N)\big) & \text{Q.E.D.}
\end{aligned}$$

Implementation

Here is the implementation using C++ template.

```
template <typename T, typename F>
T min_max(const complete_multipartite_graph<T>& graph, const F& functor)
{
    typedef typename complete_multipartite_graph<T>::const_y_iterator const_y_iterator;
    typedef typename complete_multipartite_graph<T>::const_x_iterator const_x_iterator;

    const_y_iterator y_prev_layer = graph.begin();
    const_y_iterator y_this_layer = graph.begin();
    std::vector<T> subprob_opt_prev_layer;
    std::vector<T> subprob_opt_this_layer;

    ++y_this_layer;
    for(; y_this_layer!=graph.end(); ++y_prev_layer, ++y_this_layer)
    {
        for(const_x_iterator x_this_layer  = y_this_layer->begin();
                             x_this_layer != y_this_layer->end(); ++ x_this_layer)
        {
            T subprob_min = std::numeric_limits<T>::max();

            size_t index = 0;
            for(const_x_iterator x_prev_layer  = y_prev_layer->begin();
                                 x_prev_layer != y_prev_layer->end(); ++ x_prev_layer, ++index)
            {
                T temp0 = functor(*x_this_layer, *x_prev_layer);
                T temp1 = std::max(subprob_opt_prev_layer[index], temp0);

                if (subprob_min > temp1)
                    subprob_min = temp1;
            }
            subprob_opt_this_layer.push_back(subprob_min);
        }

        subprob_opt_prev_layer = subprob_opt_this_layer;
        subprob_opt_this_layer.clear();
    }
    return *std::min_element(subprob_opt_prev_layer.begin(), subprob_opt_prev_layer.end());
}
```