```
function d = graphDissimilarity(mat1,mat2,NameValueArgs)
% This function computes the graph-based dissimilarity between two
% hypergraphs.
%
% Inputs
% mat1 and mat2: matrices representing the two hypergraphs to compare.
%   Depending on the type input, these can either be the graph adjacency
%   matrices or the graph laplacian matrices.
%
% type: {'Hamming', 'Jaccard', 'deltaCon', 'centrality', 'heatKer', ...
%   'spanTree', 'Spectral'}. Specifies the dissimilarity measure to be
%   used. 'Hamming', 'Jaccard', 'deltaCon', and 'centrality' take adjacency
%   matrices for mat1 and mat2 while the last three types take Laplacian
%   matrices for mat1 and mat2.
%
% varargin: arguments for 'deltaCon', 'heatKer', and 'centrality'.
%   'deltaCon': a single float to represent epsilon in the deltaCon
%               algorithm
%   'heatKer': a single float ot represent tau.
%   'centrality': single string or character vector representing the
%               centrality type. See MATLAB's centrality function for
%               details.
%
% Output
% d: double representing computed dissimilarity between the hypergraphs
%   represented by mat1 and mat2.
%

arguments
    mat1
    mat2
    NameValueArgs.type = "Hamming"
    NameValueArgs.eps = 10e-3
    NameValueArgs.tau = 10e-3
    NameValueArgs.centrality = 'eigenvector'
end
d=NaN;

type = NameValueArgs.type;
eps = NameValueArgs.eps; % parameter for deltaCon
tau = NameValueArgs.tau; % parameter for heatKer
cenType = NameValueArgs.centrality;


types = {'Hamming', 'Jaccard', 'deltaCon', 'centrality', 'heatKer', ...
    'spanTree', 'Spectral'};
type = validatestring(type, types);


switch type

    case 'Hamming'
```

```
        % mat1 and mat2 are adjacency matrices
        if ~isempty(mat1) && ~isempty(mat2)
            n=size(mat1,1);
            d=norm(mat1-mat2,1)/(n*(n-1));
        end

    case 'Jaccard'
        % mat1 and mat2 are adjacency matrices
        if ~isempty(mat1) && ~isempty(mat2)
            mat1=mat1(:); mat2=mat2(:);
            minA=min([mat1 mat2],[],2);
            maxA=max([mat1 mat2],[],2);
            d=1-sum(minA)/sum(maxA);
        end


    case 'deltaCon'
        % mat1 and mat2 are adjacency matrices
        if ~isempty(mat1) && ~isempty(mat2)
           D1=diag(sum(mat1,1));
           D2=diag(sum(mat2,1));
           S1=inv(eye(size(mat1))+eps^2*D1-eps*mat1);
           S2=inv(eye(size(mat2))+eps*D2-eps*mat2);
           n1=size(mat1,1);
           n2=size(mat2,1);
           d1=(sqrt(S1)-sqrt(S2)).^2;
           d=(1/(n1*n2))*sqrt(sum(d1(:)));
        end

    case 'centrality'
        % mat1 and mat2 are adjacency matrices
        if ~isempty(mat1) && ~isempty(mat2)
            G1 = graph(mat1,'omitselfloops','upper');
            G2 = graph(mat2,'omitselfloops','upper');
            switch cenType
                case 'eigenvector'
                    c1=centrality(G1,cenType,'Importance',G1.Edges.Weight);
                    c2=centrality(G2,cenType,'Importance',G2.Edges.Weight);
                otherwise
                    c1=centrality(G1,cenType,'Cost',G1.Edges.Weight);
                    c1=c1/norm(c1);
                    c2=centrality(G2,cenType,'Cost',G2.Edges.Weight);
                    c2=c2/norm(c2);
            end
            n1=size(mat1,1);
            n2=size(mat2,1);
            d=1/(n1)*norm(c1-c2);
        end

    case 'heatKer'
        % mat1 and mat2 are Laplacian matrices
        [U1, lam1] = eig(full(mat1));
        [U2, lam2] = eig(full(mat2));
        lam1 = diag(lam1);
```

```
        lam2 = diag(lam2);
        if ~isempty(mat1) && ~isempty(mat2)
            n=size(mat1,1);
            [lam1,ind1]=sort(lam1);
            [lam2,ind2]=sort(lam2);
            U1=U1(:,ind1); U2=U2(:,ind2);
            eLam1=diag(exp(-lam1*tau));
            eLam2=diag(exp(-lam2*tau));
            hW1=U1*eLam1*U1';
            hW2=U2*eLam2*U2';
            delta=hW1-hW2;
            d=1/n*trace(delta'*delta);
        end

     case 'spanTree'
        % mat1 and mat2 are Laplacian matrices
        if ~isempty(mat1) && ~isempty(mat2)
            [~, lam1] = eig(full(mat1));
            [~, lam2] = eig(full(mat2));
            lam1 = diag(lam1);
            lam2 = diag(lam2);
            lam1=sort(lam1);
            lam2=sort(lam2);
            log1=sum(log(abs(lam1(2:end))));
            log2=sum(log(abs(lam2(2:end))));
            d=abs(log1-log2);
        end

    case 'Spectral'
        % mat1 and mat2 are Laplacian matrices
        if ~isempty(mat1) && ~isempty(mat2)
            n=size(mat1,1);
            lam1=eig(mat1);
            lam2=eig(mat2);
            lam1=sort(lam1);
            lam2=sort(lam2);
            d=1/n*norm(lam1-lam2);
        end

end
```

Published with MATLAB® R2021b