Midterm Evaluation

The research paper discusses how graph theory can be applied to study plasma chemical reaction engineering, particularly focusing on atmospheric pressure plasmas. Here's a simplified summary:

Plasma, which is often referred to as the fourth state of matter, has various applications in chemical engineering due to its ability to rapidly alter energy conditions without requiring high pressure or temperature equipment. However, one major challenge is its lack of selectivity at atmospheric pressure, meaning it's hard to control which reactions occur. This research aims to address this challenge using graph theory.

Graph theory deals with mathematical structures consisting of nodes (points) connected by edges (lines). By representing chemical reactions and species as nodes and edges, respectively, a "reaction network" can be created. This network helps visualize the relationships between different reactions and species, allowing researchers to identify important pathways and potential problems.

The paper explores the application of graph theory to understand and optimize plasma chemical reaction engineering, particularly focusing on atmospheric pressure plasmas. These plasmas have diverse applications in chemical engineering due to their ability to rapidly change energy conditions without requiring high pressure or temperature equipment. However, a significant challenge in utilizing atmospheric pressure plasmas is their lack of selectivity, meaning it's challenging to control which reactions occur.

To address this challenge, the researchers propose using graph theory, dealing with structures composed of nodes (points) connected by edges (lines), to model the complex network of chemical reactions involved in plasma chemistry. By representing individual chemical reactions as nodes and the species involved as edges, a comprehensive "reaction network" can be constructed. This network allows researchers to visualize the intricate relationships between different reactions and species, providing insights into potential pathways and interactions.

One of the key methodologies employed in the study is Dijkstra's algorithm, a mathematical approach used to find the shortest pathways between nodes in a graph. Applied to the plasma chemical reaction network, Dijkstra's algorithm helps identify which reactions are most likely to occur and which conditions are favorable for their formation. Additionally, the researchers develop an algorithm called OCARINA (Optimal Condition Approaching via Reaction-In-Network Analysis) to estimate optimal conditions for targeted chemical formation based on the network analysis.

The study utilizes open-source graph visualization software, Gephi, to visualize the reaction network. By representing reactions and species as nodes and edges with various attributes such as size, color, and label, researchers can gain valuable insights into the complex interactions within the plasma chemical system. The visualizations generated by Gephi help researchers identify patterns, potential bottlenecks, and promising pathways for further investigation.

Limitations

However, the research also acknowledges several limitations. For instance, the study assumes linear relationships between rate coefficients and electron energy or gas temperature, which may not always hold true in real-world scenarios. Additionally, the complexity of plasma chemical systems poses challenges in visualizing and interpreting large networks, especially when considering numerous reactions and species. The research focuses on forward reactions without considering equilibrium reactions. While this simplification may be appropriate for certain plasma systems, it overlooks the potential impact of reversible reactions and dynamic equilibria on the overall behavior of the system. Ignoring equilibrium reactions may lead to incomplete or inaccurate insights into reaction pathways and kinetics.

Problems

Maximum Score After Applying Operations

class Solution {

public:

vector<int>v;

long long solve(int s,int pre,vector<vector<int>> &g)

{

long long ans=1e15;

ans=v[s] ;

long long tem=0;

int c=0;

for(auto i: g[s]) {

if(i!=pre) {

c++;

tem+=solve(i,s,g);

}

}

if(c)

ans= min(ans, tem);

return ans;

}

long long maximumScoreAfterOperations(vector<vector<int>>& edges, vector<int>& a) {

int n=a.size();

v=a;

vector<vector<int>> g;

g.resize(n);

for(auto i:edges)

{

g[i[0]].push\_back(i[1]);

g[i[1]].push\_back(i[0]);

}

long long sum=0;

for(auto i: a) sum+=i;

long long t=solve(0,-1,g);

return sum- t;

}

};

Binary Tree Flattened To Linked List

class Solution {

public:

TreeNode\* rightmost(TreeNode\* root){

if (root->right==NULL) return root;

return rightmost(root->right);

}

void flatten(TreeNode\* root) {

if (root==NULL) return;

TreeNode\* nextright;

TreeNode\* rightMOST;

while (root){

if (root->left){

rightMOST = rightmost(root->left);

nextright = root->right;

root->right = root->left;

root->left=NULL;

rightMOST->right=nextright;

}

root=root->right;

}

}

};

Vertical Order Traversal Of A Binary Tree

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

map<int,map<int,vector<int>>> mp;

queue<pair<TreeNode\*,pair<int,int>>> q;

q.push(make\_pair(root,make\_pair(0,0)));

while(!q.empty())

{

pair<TreeNode\*,pair<int,int>> front=q.front();

q.pop();

TreeNode\* frontNode=front.first;

int hd=front.second.first;

int lvl=front.second.second;

mp[hd][lvl].push\_back(frontNode->val);

if(frontNode->left)

{

q.push(make\_pair(frontNode->left,make\_pair(hd-1,lvl+1)));

}

if(frontNode->right)

{

q.push(make\_pair(frontNode->right,make\_pair(hd+1,lvl+1)));

}

}

vector<vector<int>> ans;

for(auto i:mp)

{

vector<int> temp;

for(auto j:i.second)

{

sort(j.second.begin(), j.second.end());

for(auto k:j.second)

{

temp.push\_back(k);

}

}

ans.push\_back(temp);

}

return ans;

}

};