CliqueSolve

Sommario

Introduction

CliqueSolve is a web application developed by Guelfo Watterson, student at KU Leuven in Gent for the Academic Year 2023/24, under the supervision of Professor Tony Wauters. The application aims to read and parse undirected graph files in different formats in order to display the main properties of the graphs. Furthermore, the central feature of the application is to find the maximum clique of the uploaded graphs.

Graph theory and Max-Clique Problem

In this file some specific terms referring to graphs are used.

Undirected Graph: A graph in which edges have no direction. Each edge is an unordered pair of vertices, meaning the edge (u, v) is identical to the edge (v, u).

Vertex (Plural: Vertices): A fundamental unit of which graphs are formed. A vertex represents an endpoint of an edge in a graph.

Edge: A connection between two vertices in a graph. In an undirected graph, an edge is an unordered pair of vertices.

Min Degree: The smallest number of edges connected to any vertex in the graph.

Max Degree: The largest number of edges connected to any vertex in the graph.

Density: A measure of how many edges are in the graph compared to the maximum possible number of edges. It is calculated as:

$$\frac{2 \cdot number\ of\ edges}{number\ of\ vertices \cdot (number\ of\ vertices - 1)}$$

Clique: a subset of vertices such that every two distinct vertices in the clique are adjacent (i.e., there is an edge between every pair of vertices in the clique).

Max Clique (Maximum Clique): the largest possible clique in a graph, containing the maximum number of vertices all connected to each other.

The problem of finding a maximum clique is a strongly hard NP-hard problem. This is one of the reasons why this problem is relevant for programmers and mathematicians.

Developing fast and efficient algorithms to solve the max clique problem can be fundamental to improve the performance of all the applications which have to handle large graphs. In detail, maps applications, navigator and social networks frequently use graphs as a data structure to represent connections between places and people.

Application overview

CliqueSolve aims to create a free online tool to find the main properties of a graph, including the maximum clique, in a fast and easy-to-use way.

The application is designed to run on all the main browsers and to be easily expandable for developers. Detailed information about it can be found in this document under the section “Extending the application”.

The application is developed using the Vue.js 3 framework, making it easier to maintain, expand and deploy. The choice of the framework was due to the extreme flexibility of Vue, which was the best option to develop an application with basic graphics but complex algorithms.

All the code that runs on the front-end of the application is written in TypeScript, since it guarantees a better compatibility with Vue.js and it makes it easier to understand and maintain the code.

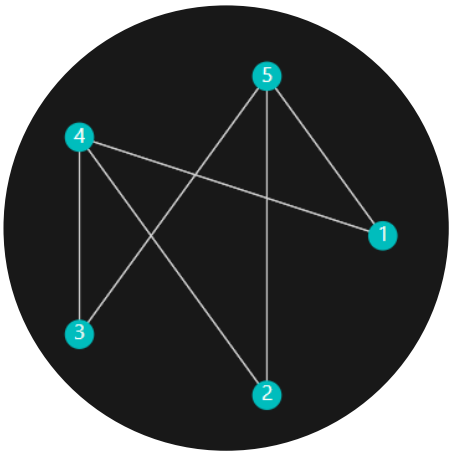
The application also uses a Node.js server, in order to run external max clique algorithms on the back-end. This allows developers to extend the application by adding external algorithms written in Python and C++.

Features

Firstly, the user can upload a graph file with .txt extension, in one of the supported formats:

- Adjacency list
- Adjacency matrix
- DIMACS format
- Graph6 format

Here a brief explanation of these formats with an example referring to the following graph:



Adjacency List

An adjacency list is a data structure used to represent a graph where each vertex is associated with a list of adjacent vertices. In this representation, for each vertex, a list is maintained that contains all the vertices directly connected to it by an edge. This format is memory-efficient for sparse graphs, where the number of edges is much smaller than the

number of vertices, as it only stores information about existing edges. All the front-end functions of CliqueSolve receive the adjacency list of a graph as an input, since this format is easy to read and to convert into all the other formats.

Ex.

```
1: 4 5
2: 4 5
3: 4 5
4: 1 2 3
5: 1 2 3
```

Adjacency Matrix

An adjacency matrix is a square matrix used to represent a graph. In this matrix, rows and columns correspond to vertices, and the presence of an edge between two vertices is indicated by a non-zero value in the corresponding cell. Typically, a value of 1 or a weight value represents the presence of an edge, while a value of 0 represents no edge. This format is memory-intensive for large graphs, but it allows for efficient lookup of edge presence between any pair of vertices.

Ex.

```
0 0 0 1 1
0 0 0 1 1
0 0 0 1 1
1 1 1 0 0
1 1 1 0 0
```

DIMACS Format

DIMACS (DIMensional Array Collection System) format is a text-based file format commonly used to represent graphs in computer science. It includes a header line (starting with the character ‘p’) that contains the word ‘edge’ followed by the number of vertices and edges, and a body section with graph data. The body section contains lines representing edges, where each line starts with the character ‘e’ and specifies the vertices connected by an edge. DIMACS format is often used for input and output in graph algorithms and graph-related research. Comment lines start with ‘c’.

Ex.

```
p edge 5 6
e 1 4
e 1 5
e 2 4
e 2 5
e 3 4
e 3 5
```

Graph6 Format

Graph6 format is a compact text-based format used to represent undirected graphs. It provides a concise encoding of graph structures using a combination of printable ASCII characters. The format consists of a single line of text representing the entire graph. The first char gives information about the number of vertices, the following ones about the edges. Graph6 format is useful for storing and sharing small to medium-sized graphs efficiently, especially in situations where space is limited or when transmitting graph data over networks.

Ex.

```
DFw
```

Then, the application will display a graphical visualization of the graph (if the number of vertices is less than 50) and the main properties of the graph:

- Number of vertices
- Number of edges
- Minimum degree
- Maximum degree
- Density

At this point, the user can choose one of the available algorithms and run it, in order to obtain the visualization of the maximum clique of the graph. There are currently five algorithms available:

Greedy algorithm

The greedy algorithm aims to find a maximum clique in an undirected graph represented by an adjacency list. It first filters out vertices with no connections. For each remaining vertex, the algorithm tries to build a clique by adding connected vertices greedily. It uses a helper function to ensure each added vertex is connected to all current clique members. The largest clique found during this process is returned as the result. This greedy approach provides a good approximation of the maximum clique and guarantees a low execution time even for large graphs, though it may not always find the absolute largest one. The algorithm is written in TypeScript and runs in the front-end of the application.

Backtracking algorithm

The backtracking algorithm finds the maximum clique in an undirected graph represented by an adjacency list using a backtracking approach. It starts by checking if the graph is complete; if so, it returns all vertices. Otherwise, it calculates vertex degrees, sorts vertices by degree, and uses a recursive backtracking function to explore possible cliques. During this process, it ensures each candidate set forms a valid clique and keeps track of the largest clique found. This guarantees to find the effective maximum clique if the algorithm is run until the end. Anyway, the algorithm includes the possibility to set a time limit to prevent long execution times, since it could run for a very long time when receiving a large dense

graph as input. When the time limit is reached, the algorithm returns the partial result found at that moment. The algorithm is written in TypeScript and runs in the front-end of the application.

C++ algorithm by Shalin Shah

The C++ algorithm by Shalin Shah finds the maximum clique in an undirected graph in DIMACS format. The algorithm is written in C++ and runs on the back-end of the application thanks to the implementation of a Node.js server, that executes the pre-complied .exe file. The user can select the number of iterations before running the algorithm. Further information about the algorithm can be found on the GitHub page of the project (see References).

Branch & Bound Python algorithm by Grigory Zhigalov

The Branch & Bound algorithm by Grigory Zhigalov finds the maximum clique in an undirected graph in DIMACS format. The algorithm is written in Python and runs on the back-end of the application thanks to the implementation of a Node.js server. The user can set a time limit before running the algorithm. Note that, differently from the front-end Backtracking algorithm, this one does not return a partial result when the time limit is reached. Further information about the algorithm can be found on the GitHub page of the project (see References).

NetworkX Python algorithm

The NetworkX Python algorithm finds the maximum clique in an undirected graph represented by an adjacency list. The algorithm was written by Guelfo Watterson using the max_clique function from NetworkX's Python library to find the maximum clique. Also, this algorithm offers the possibility to set a time limit. Further information about the max_clique function can be found on NetworkX website (see References).

Application architecture

Front-end

The front-end application is realized with the framework Vue.js 3, using HTML, CSS and TypeScript. All the graphical elements are included in a single component called NewGraph.vue.

All the TypeScript functions needed to run the application are divided into four class files, that can be found in the folder '…/MaxClique_project/src/classes':

- GraphParse.vue: functions to parse graph files, convert between the different formats and extract information such as number of vertices and number of edges.

- GraphProperties.vue: functions to calculate the basic properties of the graphs.

- GraphVisualization.vue: functions to display the graph drawing vertices and edges (for graphs with less than 51 vertices).

- MaxClique.vue: functions that implement the Greedy algorithm and the Backtracking algorithms to find the maximum clique.

The folder '…/MaxClique_project/src/external_algorithms' contains the external maximum clique algorithms that are sent to the server to be run in the back-end.

Back-end

The folder '…/MaxClique_project/server' contains all the necessary files to run external algorithms in the back-end. The file algoConfig.json is the one that has to be edited when new algorithms are added (see Extending the application).

The file server.js is a Node.js server using Express and Cors that contains three posts to handle the requests coming from the front-end:

- Load JSON: when one of the external algorithms is selected to find maximum clique, the client sends the algorithm name to the server, that sends back all the configuration data of the selected algorithm.
  Input: algorithm name.
  Output: path, extension, input graph format, parameters, call format.

- Execute .exe: when one of the external algorithms is run and the variable 'extension' contains '.exe', the client sends all the necessary files and information to the server to run the algorithm.
  Input: .exe file, graph file, parameters (if required), call format.
  Output: .exe file's output.

- Execute .py: when one of the external algorithms is run and the variable 'extension' contains '.py, the client sends all the necessary files and information to the server to run the algorithm.
  Input: .py code, graph file, parameters (if required), call format.
  Output: .py file's output.

Environment setup and application deployment

Prepare your environment and test the application

1. Ensure Node.js and npm are installed on your local machine. Download and install them from the https://nodejs.org/.

2. Install Vue CLI and Vite (if not already installed)

```
npm install -g @vue/cli vite
```

3. Test the project locally

```
cd .../MaxClique_project
npm run build
```

4. Open another Powershell tab and run the server

```
cd .../MaxClique_project/server
node server.js
```

Build the Vue.js Front-end

1. Navigate to the project folder

```
Cd .../MaxClique_project
```

2. Install dependencies

```
npm install
```

3. Create a `.env.production` file in the root of the Vue.js project to set production environment variables if needed.

4. Build the project

```
npm run build
```

This will generate a `dist` directory with your production-ready static files.

Deploying the Node.js Backend

1. If you don't have an AWS account, sign up at https://aws.amazon.com/.

2. Set up an EC2 instance
   a. Log in to the AWS Management Console.
   b. Navigate to EC2 and launch a new instance (ensure it is running Linux).
   c. Note down the public IP address of your EC2 instance.

3. Use SSH to connect to your EC2 instance

```
ssh -i your-key.pem ec2-user@your-ec2-public-ip
```

4. Update the package index and install Node.js

```
sudo yum update -y
curl -sL https://rpm.nodesource.com/setup_14.x | sudo bash -
sudo yum install -y nodejs
```

5. Clone your Node.js server repository onto your EC2 instance.

6. Install dependencies and start the server

```
cd …\Max-Clique_project\server\server.js
npm install
npm start
```

7. Ensure your server has the necessary permissions to execute .exe and Python files. You may need to grant appropriate execution permissions using the `chmod` command.

8. Configure Python environment

Since out Node.js server needs to execute Python scripts, make sure Python is installed on the EC2 instance:

```
sudo yum install python3
```

Serve Vue.js Front-end with Nginx on AWS

1. Install Nginx

```
sudo yum install nginx -y
```

2. Configure Nginx

a. Edit the default Nginx configuration file:

```
sudo nano /etc/nginx/nginx.conf
```

b. Update the `server` block to serve your Vue.js app:

```
server {
    listen 80;
    server_name your_domain_or_EC2_IP;

    location / {
        root .../MaxClique_project/dist;
        try_files $uri $uri/ /index.html;
    }
}
```

3. Restart Nginx

```
sudo systemctl restart nginx
```

Deploy Your Application Online

1. Select a reliable web hosting provider. Popular options include AWS, Google Cloud Platform, DigitalOcean, Heroku, and Netlify.

2. Sign up for an account with your chosen hosting provider and follow their instructions to set up your account.

3. Upload the built Vue.js project:

a. Build the Vue.js project as explained in the previous steps.

b. Use the hosting provider's dashboard or command-line tools to upload your built Vue.js project files.

4. If you have a custom domain, configure the DNS settings to point to your hosting provider's servers. This typically involves setting up an A record or a CNAME record.

5. Test Your Application on Multiple Browsers:

a. Ensure the application works as expected on popular browsers such as Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, and Opera.

b. Use browser developer tools to identify and fix any compatibility issues.

6. Monitor and Maintain Your Application:

a. Regularly test the application on different browsers and devices to ensure it remains compatible.

b. Monitor for any compatibility issues and address them promptly.

c. Keep up-to-date with changes in browser specifications and standards.

d. Update the code and dependencies accordingly to maintain compatibility with the latest browser versions.

Extending the application

Adding new formats

1. Open the component "GraphNew.vue", look for the form with id="graphUploadForm" and add a new option:

```
<option value="newFormat">New Format</option>
```

2. In the function “handleExternalMaxClique”, add a new case to handle your format:

```
case 'new_format':
          convertedGraphFile =
parseFunctions.adjacencyListToGraph6(adjacencyList,
numberOfVertices.value);
          break;
```

3. Open the class file "GraphParse.vue", go to the function "parseUploadedGraph" and add a new case for your format:

```
case 'new_format':
          result = this.yourFormatToAdjacencyList(graphData);
          break;
```

4. Make sure to implement the functions "yourFormatToAdjacencyList" and “adjacencyListToYourFormat” inside the class "GraphParse" to convert your file format both ways.

Adding new max clique algorithms

To add additional external algorithms to the application, copy the file of your algorithm in the folder /src/external_algorithms.

You can add a python algorithm (extension .py), or a pre-compiled code (format .exe). No other formats are supported for now.

Make sure to edit the file algoConfig.json in the server folder, adding all the required information for your algorithm:

```
{
    "name": "",
    "path": "",
    "extension": "",
    "format": "",
    "parameters": [
      {
        "name": "graphFile.txt",
        "type": "file"
      },
      {
        "name": "",
        "type": ""
      }
    ],
    "call_format": ""
  }
```

The path should be `"src/external_algorithms/yourFile.extension"`.

The extension can be ".py" or ".exe".

The format is referred to the graph representation format that your algorithm receives as input. The application currently supports adjacency list, adjacency matrix, DIMACS format and Graph6 format.

The first parameter should be the graph file. You should nod edit this part since it is automatically renamed by the application as soon as you upload it on the page.

Then, make sure to add all the required input parameters for your algorithm. For now, the app supports the input of two possible parameters: timeLimit and numOfIterations.

The call format should specify the command line that launches your algorithm. Write it as a string, with variable names written in the following format:

```
${varName}
```

Always use "graphFilePath" to refer to the relative path of the graph file, "executablePath" or "pythonFilePath" to refer to the file path of your algorithm and "numIterations" and "timeLimit" to refer to the parameters. Different ways of calling the variables are not recognized by the app:

Ex:

```
"call_format": "${executablePath} ${graphFilePath} ${numIterations}"
```

Furthermore, make sure that the output of your algorithm contains the Max Clique in the following way:

```
...
... Clique:
1 2 5 6 7
...
```

It is not relevant if the output contains other type of information or text, the app will only read the line that follows the word "Clique".

Before adding a new algorithm to this application, always check its license and copyright.

References and Additional Resources

The C++ algorithm implemented in the application is an open-source algorithm developed by Shalin Shah. All the information about it can be found at:

https://github.com/shah314/clique/blob/master/README.md

The Branch and Bound algorithm implemented in the application is an open-source Python algorithm developed by Grigory Zhigalov. All the information about it can be found at:

https://github.com/donfaq/max_clique/blob/master/README.md

The one descripted as “NetworkX” algorithm is a Python script that uses the max_clique() function of NetworkX.