University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Visualization of Classical Graph Theory Problems

Fatma Al-Sayegh
20 November 2022

Abstract

Every abstract follows a similar pattern. Motivate; set aims; describe work; explain results.

"XYZ is bad. This project investigated ABC to determine if it was better. ABC used XXX and YYY to implement ZZZ. This is particularly interesting as XXX and YYY have never been used together. It was found that ABC was 20% better than XYZ, though it caused rabies in half of subjects."

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Fatma Al-Sayegh Date: 20 November 2022

Contents

1 Introduction

1.1 Scope of The Project

The project's objective is to visualize graph problems and not to solve them. Solution of such problems could range from moderately difficult to very difficult to solve programatically. Therefore in this project, although the abstraction of Graphs (Sets of Vertices and Sets of Edges), is quite general, care is taken that we provide the solution to the visualization program before hand. Therefore for most problems here, the solution is known before hand and is used for various animations.

1.2 Terms and Definitions

The following are the basic terms and definitions which are used in Graph theory.

1.2.1 Some basic definitions:

- Vertices: A graph has vertices. Which can be understood as points.
- Edge: An edge is an unordered pair of vertices.
- V(G) is the vetex set of graph G.
- E(G) is the edge set of graph G.
- Degree of a vertex: deg(v). Number of edges connected to a vertex.
- isolated vertex v: deg(v) = 0
- End vertex w: deg(w) = 1

1.2.2 Some basic facts

Handshaking Lemma In any graph the sum of all the the vertex-degree is an even number - in fact, twice the number of edges, since each edge contributes exactly 2 to the sum. Total summation of is edes*2

Adjacency Two vertices are adjacent if they have an edge between them. Two edges are adjacent if they have a vertex in common.

What graphs are not about metrical properties: length and shape of the edges. (Irrelevant in graph theory). (so if an edge is smaller , larger etc doesnt matter what matters if the edges between two vertices are connected or not)

Some more definitions

Simple graphs: graphs with no loops and multiple edges.

General graph: loops and multiple edges are allowed.

Digraphs: Directed graphs. When edges have arrows.

Walk: A way of getting from one vertex to another, and consists of a sequence of edges. P -> Q -> R is a walk of length 2

Path: A walk in which no vertex appears more than once.

Cycle A path like this: Q -> S -> T -> Q is called a cycle.

Subgraph G - e is the graph obtained from G by deleting the edge e.

Adjecency Matrix If G is a graph with vertices labelled {1,2,..n}, its adjacency matrix A is n * n matrix whose ijth entry is the number of edges joining vertex i and vertex j.

Null Graph Edge set is empty. All vertices are isolated.

Complete Graph A simple graph in which each pair of distinct vertices are adjacent is a complete. Complete Graph of n vertices are denoted as Kn. They have n(n-1)/2 edges.

Cycle Graphs A connected graph which is regular of degree 2. A cycle graph of n vertices is denoted by Cn.

Path Graph A graph obtained from Cn by removing an edge. Denoted by Pn.

Wheel The graph obtained from Cn-1 by joining each vertex to a new vertex v is wheel on n vertices, denoted by Wn.

Regular Graph Each vertex has the same degree. With that degree r, the graph is a regular of degree r of r-regular.

Bipartite Graph If the vertex set of a graph G can be split into two disjoint sets A and B such that each edge of G joins a vertex of A and a vertex of B, then G is a bipartite graph. (like when we divide it into two sets anything in a will connect with set b not another set of a)

Hamiltonian graphs Graphs containing walks that include every vertex exactly once, ending at initial vertex. (so we should start and end in same point without repeating vertices not all vertices have to be used)

1.3 Topic Explanation

1.3.1 Some Classical Graph Problems:

- Graph Isomorphism
- Hamiltonian Cycle
- Graph Colouring
- Indentification of Independent Set
- Identification of Cliques in a Graph
- Finding Minimum spanning tree in Graph
- Finding of Vertex cover
- Max k-Cut
- Tree Width

1.3.2 Graph Isomorphism:

Two graphs G1 and G2 are isomorphic if there is a one-one correspondence between the vertices of G1 and G2 such that the number of edges between any two vertices in G1 is equal to the number of edges joining the corresponding vertices of G2. Here the graphs may appear to be different in appearance and the labeling of the nodes and edges. But the way one vertex is connected to another in one graph is same to another. Therefore given two graphs, detecting if the graphs are Isomorphic is a problem to solve. One way to explain this would be to manipulate the position of vertices and edges to be appear same as it's isomorphic counterpart. We want to show what isomorphism is.

Visualization: - Display two graphs which are visibly (topologically) different but isomorphic.

User Interaction: – Given a initial graph which the user can manipulate by draging its vertices to make it look equivalent to it's isomorphic equivalent. – Position of vertices and edges in one of the graphs can be moved in an animation to make it look visually the same as its isomorphic counterpart. Graph drawing algorithms should be employed here.

1.3.3 Hamiltonian graphs:

Graphs containing walks (moving from one edge to another) that include every vertex exactly once, ending at initial vertex. (so we should start and end in same point without repeating vertices and cover all the vertices).

Visualization: – Explaination can be given by giving an example graph with a hamiltonian cyle highlighted on it. – We can also give negative examples of paths which are not hamiltonian. – There can also be an Exercise for the user in which she is given a graph and and asked to mark the edges which would make a hamiltonian cycle. – The program should check if that is a hamiltonian cylce by checking if all the related constraints defined in the definition of hamiltonian cycle are satisfied which are if all the vertices are visited and also that no vertex has been visited twice.

1.3.4 Graph Clouring

It is an optimization problem, where the objective is to assign to vertices of a graph a colour such that no two adjacent vertices have the same color, while keeping the number of colours employed to a minimum.

Visualization: A properly coloured graph, which satisfies all the constraints can be displayed. – Integer Linear Programing can be explained here as well.

User Interaction: User is allowed to color vertices. The program will warn the user if adjacent nodes are colored the same. Will be like solving sudoku.

1.3.5 Identification of Maximum Independent Set

An Independent Set of a graph is a set of vertices such that no two vertices in that set are adjacent to each other. A Maximum Independent set is an Inpendent set with the largest possible number of vertices.

Visualization: – Show a graph with members of an Independent Set coloured differently from the rest of the vertices. – Show the same graph with maximum independent set.

1.3.6 Identification of Cliques in a graph

A clique is a set of vertices of a graph such that all the vertices are connected to each other. This set is defined in such a way that there is no other vertex in the graph which can be added to the set, while preserving the property that all the vertices are connected to every other. Visualization: – In a graph, a clique can be highlighted by colouring the vertices and the edges involed in the clique.

1.3.7 Finding Minimum spanning tree in Graph

In a connected graph, minimum spanning tree is a subset of edges such that, all the vertices are connected. This should be a tree without any cycles and the summation of weights should be minimum if there are more than one spanning trees present in the graph.

Visualization: – In a succession the following can be done – Show what a connected graph is. (Define what a connected graph is.) – Draw spanning tree on the graph. (Define what a spanning tree is.) – Draw a minimum spannig tree. (Show that the constraints of the definition are satisfied.)

1.3.8 Minimum Vertex Cover

Minimum Vertex cover of a graph is the minimum amount of vertices such that, all the edges in the graph must have one of such vertices as at least one of their endpoints.

Visualization: Animation: - Show how only a few vertices can cover all the edges, and such vertices are the vertex cover of the graph. - Show all the constraints of the definition are satisfied.

Interactive: - The user chooses an vertex and all the edges incident on it light up. - Keeps choosing a vertex until all the edges light up.

1.3.9 Max k-Cut

A maximum cut, is partioning the vertices of a graph in two groups such that the number of edges between these two groups is maximum. In a weighted graph, where the edges are weighted, the weights of the edges are also taken into consideration. A maximum k-cut, is generalised version of maximum cut, where the graph is partitioned into k subsets, such that the number of edges between these groups is maximised.

Visualization:

1. 2-cut Animation:
    - Making a 2-cut by a curved line on a simple graph to show how maximum number of edges have been cut between the two groups of vertices. or classifying the vertices of the two sets by depicting them in two different graph.
    - Then clustering them and seperating the two groups by a distance (of course without breaking any edges) to show the flow of edges between the two groups.
2. Bipartite Graph (as a trivial example):
    - Showing that a bipartite graph when it is recognised as a bipartite graph is already a trivial 2-cut graph.
3. User Interaction:
    - Let the user choose the vertices he wants to be in set A. The rest of vertices will auto-matically will be assigned to set B. The program will show the number of edges which flow between the two sets.
4. Animation for k-cut:
    - Making a 3-cut by two curved lines on a simple graph to show how maximum number of edges have been cut by the two lines.
    - Making a 2-cut by a line on a weighted graph, where the cut is made keeping the weights of the edges into consideration.
5.
    - Ideas on constructing a graph such that max cut (2-cut) is known before hand.
    - Take two sets of vertices, A and B, initialy with zero edges in the graph. Whenever an edge is made between the vertices of A, then two edges is made between vertices of set A and B.

1.3.10 Tree Width

We will explain in two parts. First we will define what a tree decomposition of a graph is. Then we will define tree width of the graph.

Formally, a tree decomposition of G = (V,E) consists of a tree T and a subset Vt <= V associated with each node t <- T. We will call the subsets Vt pieces of tree decomposition. T and Vt must satisfy: - (Node coverage) Every node of G belongs to at least one piece Vt. - (Edge coverage) For every edge e of G, there is some piece Vt containing both ends of e. - (Coherence) Let t1, t2 and t3 be three nodes of T such that t2 lies on the path from t1 to t3.

All graphs have tree decompositions. A trivial tree decomposition of any graph has just one node with all the vertices of the graph residing in it. It will satisfy all the conditions mentioned in the definition of tree decomposition. But this trivial decomposition will not be useful as it will not have seperable properties of a tree.

Therefore we must look for tree decompositions which have small pieces. Tree width is defined as the size of the biggest Vt - 1.

Therefore tree width of G is the minimum width of any tree decomposition of G.

2 Background

3 Analysis/Requirements

4 Design

5 Implementation

6 | Evaluation

7 Conclusion

A Appendices

7 Bibliography