d'CartesiaN

Jurnal Matematika dan Aplikasi

p-ISSN:2302-4224 e-ISSN: 2685-1083

Journal homepage: https://ejournal.unsrat.ac.id/index.php/decartesian

GUI Application to Setup Simple Graph on the Plane using Tkinter of Python

Gery J. Sumual[1], Benny Pinontoan[1*], Luther A. Latumakulita[1]

[1]Department of Mathematics-Faculty of Mathematics and Natural Sciences–Sam Ratulangi University Manado, Indonesia

[*]Corresponding author : bpinonto@gmail.com

A B S T R A C T

In graph theory, drawing a simple graph on the plane might result an intersection of pair of edges that is not intended as a vertex called crossing. The least amount of crossing in any simple drawing of a graph is the crossing number of that graph, $cr(G)$. Given a graph G and an integer K, the general problem to proof $cr(G) \leq K$ is an NP-Complete problem, which means, it is likely intractable. One of the way to proof $cr(G) \leq K$ is by showing the drawing of graph G with K number of crossing; doing it with a computer application can be much of help. Therefore, the purpose of our research is to create one using Tkinter of Python. The development of the application is feature driven. The developed features are used to find any simple drawing of graphs on the plane. As the result, the application can proof $cr(K_{3,3}) \leq 1$, $cr(K_6) \leq 3$, and $cr(KG_{5,2}) \leq 2$.

ARTICLE INFO

Received : 14 January 2021

Received after revision : 10 July 2021

Available *online* : 11 July 2021

Keywords:
Graph Theory
Crossing Number
Tkinter
Python
Feature Driven Development

1. INTRODUCTION

Graph is one of the modelling tools of objects or processes and their relations. In 1735 the mathematician Leonard Euler drawn graph model of the Konigsberg bridgeproblem on a two dimensional plane, solved it, and consequently, instigated a new field in mathematics which is now known as Graph Theory. Many other real-world problems can be modelled using graph provided there are set of objects or processes and pairwise relations between them. The objects or processes can be represented as vertices and their relations as edges.

In a drawing of a graph on the plane, edges can cross. This is, in some cases, to be minimized, if not to be avoided. One of the concepts in graph theory that concern with this is the crossing number, which is the minimum number of crossings among all 'good' drawings of a graph in the plane [15]. In real world application such as the circuit layout of Very Large Scale Integration (VLSI), crossing number is used to obtain the lower bound on the amount of chip area of VLSI devices like microprocessors and memory chips [10]; Additionally, crossings in the circuit layout could cause short circuit and therefore worth minimized independent of the chip area consideration.

Problems studying the minimizing of crossings of a graph G, in graph theory, include: crossing number $cr(G)$, rectilinear crossing number $\overline{cr}(G)$, and pagenumber $\rho(G)$. Crossing number is an NP (nondeterministic polynomial time)-Complete problem, whereas rectilinear crossing number and page number are NP-Hard problems. Hence, these problems are likely to be difficult, and it is justified, in general, to focus on inexact methods that only estimate crossing numbers.

To prove that $cr(G) = k$, usually by proving $cr(G) \leq k$ and $cr(G) \geq k$. One of the ways to prove $cr(G) \leq k$ is by drawing the graph G showing k crossings. This is not easy to do manually. It is needed and would be handy to have an application that can setup graphs which have drawing conform to the one in crossing number definition, and also capable of moving vertices, along with their incident edges, and reroute vertices.

The motivation to use Python programming language [17] is the fact that it has been widely used in recent times and has large community all around the world; it's well maintained and well documented. The language is dynamic and expressive with high readibility, enabling high productivity and faster innovation for its programmers.

The Application is in a graphical user interface (GUI) format for the user to draw vertices and edges in setting up a simple graph on the plane, and altering the layout by moving the vertices along with their incident edges. One of the Python GUI framework that's considered to be a genuine one is Tkinter [19], it is lightweight and easy to use compared to other frameworks. This makes Tkinter a compelling choice for developing the application quickly and functional.

Thus, the aimed result is to create an application using Tkinter of Python that solely used to setup graph layout on the plane, and have the capability to proof $cr(G) \leq k$ of some graph G with a given integer k by showing the drawing of the graph with no more than k crossing(s).

2. LITERATURE REVIEW

2.1. Graph Theory

In 1735 Leonhard Euler was trying to come up with a solution for a problem called "Seven Bridges of Königsberg". The problem was to find a walk through the city that would cross each bridge once and only once. He was looking for some elegant and practical abstraction of this problem. His abstraction laid down the foundation of graph theory. Euler was the first person who used terms like vertex and edge. He formed a brand new mathematical discipline. Graph theory can be used for modeling almost any structure which contains objects and relations between them [20].

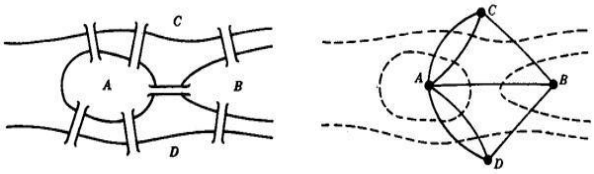


Figure 1. (a) The Königsberg Bridge problem in 1735; (b) Euler's graphical representation

According to [20], nowadays graph theory is one of the most evolving disciplines of mathematics. Graph theory is used for modeling lots of real-world problems like electric circuits, road networks, railway networks, social and information systems. Without graph theory there would be no satellite navigation in our cars, Google, etc.

2.2. Graph and Simple Graph

According to [2], A Graph G is an ordered triple $(V(G), E(G), \psi_G)$ consisting of a nonempty set V(G) of vertices, a set $E(G)$, disjoint form $V(G)$, of edges, and an incidence function ψ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G, if e is an edge and u and v are vertices such that $\psi_G(e) = uv$, then e is said to join u and v; the vertices u and v are called the ends of e. Example from [2]:

$$G = (V(G), E(G)), \psi\_G)$$
$$V(G) = \{v_1, v_2, v_3, v_4, v_5\}$$
$$E(G) = \{e, e_2, e_3, e_4, e_5, e_6, e_7\}$$

And ψ_G is defined by

$$\psi_G(e_1) = v_1v_2, \psi_G(e_2) = v_2v_3, , \psi_G(e_3) = v_3v_3,$$
$$\psi_G(e_4) = v_3v_4, \psi_G(e_5) = v_2v_4, \psi_G(e_6) = v_4v_5,$$
$$\psi_G(e_7) = v_2v_5, \psi_G(e_8) = v_2v_5$$

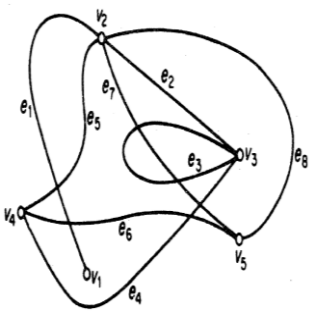


Figure 4. A diagram of G

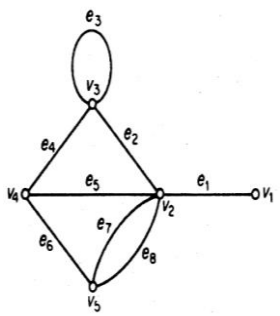


Figure 3. Another diagram of G

Here are some basic terminology and their definition according to [6] :

- If vertex v is an endpoint of edge e, then v is said to be *incident* on e, and e is *incident* on v.
- A vertex u is *adjacent* to vertex v if they are joined by an edge.
- *Adjacent* edges are two edges that have an endpoint in common.
- A *proper edge* is an edge that joins two distinct vertices.
- A *simple adjacency* between vertices occurs when there is exactly one edge between them.
- *Multi-edge* is a collection of two or more edges having identcial endpoints.
- A *self-loop* is an edge that joins a single endpoint to itself.

According to [6], Most of theoretical graph theory is concerned with simple graphs. This is partly because many problems regarding general graphs can be reduced to problems about simple graphs. A simple graph is a graph that has no self-loops or multi-edges. An example of a simple graph:

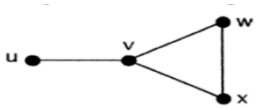


Figure 5. A simple graph

2.3. Drawing of graph on the plane

Drawing of graph on the plane can be done in many different ways. According to [15], The properties of a good drawing are: no edge crosses itself, no pair of adjacent edges cross, two edges cross at most once, and no more than two edges cross at one point. Furthermore, [8] points out that edge may cross only edges, there's no edge crossing a vertex or a vertex cross another vertex (coincide). Lastly, the remark from [21] is that a good graph is "one for which all intersecting graph edges intersect in a single point and arise from four distinct graph vertices".

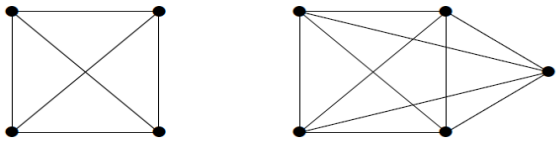


Figure 2. Good drawing of simple graphs on the plane

2.4. Crossing number problem

Given a good graph, The crossing number is the minimum possible number of crossings with which the graph can be drawn, including using curved (non-rectilinear) edges [21]. According to [16], a graph G is said to be k-crossing-critical if $cr(G)$ satisfies: "for every edge e of G, $cr(G - e) < k$, yet $cr(G) \geq k$". [3] has shown that the general crossing number decision problem "Given G and an integer K is $cr(G) \leq K$?" is an NP-complete problem which means likely to be intractable.

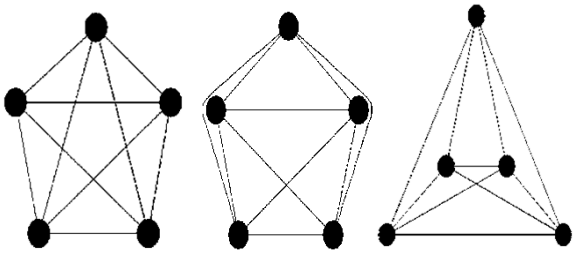


Figure 6. Different drawings of the complete graph K_5, which is a graph with five vertices and simple adjacency among them; $cr(K_5) = 1$ and K_5 is a 1-crossing-critical graph

2.5. Rectilinear crossing number problem

A straight-line drawing of a graph G is a mapping which assigns to each vertex a point in the plane and to each edge a straight-line segment connecting the corresponding two points. The rectilinear crossing number of a graph G, $\overline{cr}(G)$, is the minimum number of pairs of crossing edges in any straight-line (rectilinear) drawing of G. Determining or estimating $\overline{cr}(G)$ appears to be a difficult problem, and deciding if $\overline{cr}(G) \leq k$ is known to be NP-hard [4]. A corresponding example of this problem is in the figure 6 above ignoring the middle graph with curved edges.

2.6. Pagenumber problem

The book with k pages is the topological space B_k that consists of a line called the spine and k halfplanes called the pages, all having the spine as their common boundary [14]. A book-embedding of a graph G is an embedding of vertices of G along the spine of a book, and edges of G on the pages so that no two edges on the same page intersect; the minimum number of pages in which a graph can be embedded is called the *page number* $\rho(G)$ [7]. It has been shown by [11] that optimal book embedding is an NP-hard problem.

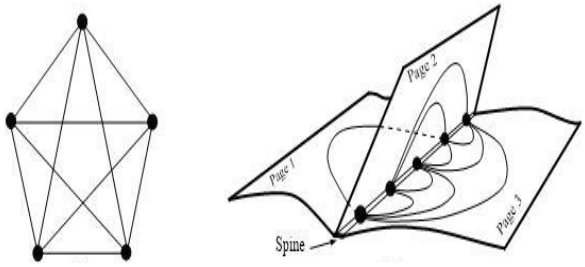


Figure 7. Embedding K_5 in a three-page book

2.7. Python programming language

Python is a general-purpose, high-level programming language which is widely used in the recent times. Its design philosophy emphasizes code readibility, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language constructs enable the user to write clear programs on both a small and large scale. The most important feature in python being it supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles [18].

2.8. Graphical User Interface (GUI)

A Graphical user interface allows a user to interact with a computer program using a pointing device that manipulates small pictures on a computer screen. The small pictures are called icons or widgets. Various types of pointing devices can be used such as a mouse, a stylus pen, or a human finger ona touch screen [12].

[12] outline the structure of a GUI program as the following:

- Create the icons and widgets that are displayed to a user and organize them inside a screen window.
- Define functions that will process user and application events.
- Associate specific user events with specific functions.
- Start an infinite event-loop that processes user events. When a user event happens, the event-loop calls the function associated with that event.

Python does not implement implement GUI, event-driven-programming in its core functionality. GUI programming is implemented using imported modules which are often referred to as "toolkits", a few of them are Tkinter, PyQt, wxPython, and kivy [12].

2.9. Tkinter

Tkinter, or "Tk interface", is a module of python that provides an interface to tk GUI toolkit, developed in TCL (*Tool command Language)* and multiplatform, with support for Linux, MAC OS and MS Windows [2]. Tk is natively present in Linux and MAC OS, and can be easily installed on MS Windows, but it is not part of Python. Tkinter is part of Python, being called "Tkinter" in versions prior to 3, and "tkinter" on subsequent versions

2.10. System/software Development

Systems development is the process of defining, designing, testing, and implementing a new software application or program. According to [9], developing a software typically involves:

- Selecting methodology
- Gathering requirements
- Developing a design
- Constructing code
- Testing
- Managing configuration and defects
- Deploying the software

3. RESEARCH METHODOLOGY

3.1. Time and Research Place

The research had been conducted in Mathematics Departments of Sam Ratulangi University from February until march 2020, and then continued from home until June 2020 due to CoVid-19 pandemic.

3.2. Softwares

1) Integrated Development and Learning Environment (IDLE) of Python

IDLE is an integrated development environment that is bundled with the Python distributions of Windows operating system. IDLE is written in Python using Tk library. The version of Python and IDLE used to develop the application are the same, *3.7.6*.

2) Python Tkinter toolkit

Tkinter is an abbreviation for "Tk interface". "Tk" is a platform independent, customizable, and configurable GUI library. The Python package 'tkinter' allows Python programs to use the TK libraries. The virtues of 'tkinter' are that it is fast, and usually comes bundled with Python. The version of Tk used in this research is *8.6.9*.

3.3. Research Stages

The stages to be used in completing this research are as follows:

1) Study the relevant literature from articles, books, journals, and websites. It was was done in order to understand the background, context, scope, and what justifies the research, as well as refining the problem at hand and knowing the methods to develop the application.
2) Determine the methodology and method for the application development. Based on [1], it was the feature driven development (FDD) method, as part of Agile methodology, that was chosen. Agile methodology is about feedback and change [22], as part of this approach, FDD method consists of five sequential processes [12] where the last two (design and build) are iterative part of this method which supports quick adaptations to late requirements and needs.

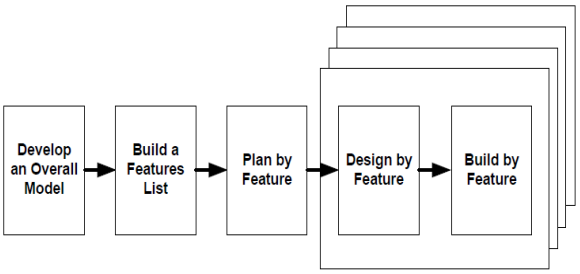


Figure 8. Processes of FDD

4. RESULTS AND DISCUSSION

4.1. Python IDLE

IDLE is a GUI application that is written in Python using Tkinter, so it is a good example of Tkinter in action. Therefore, Ironically enough, the GUI application from this research is developed within another GUI application that is also based on Tkinter framework.

4.2. Canvas Widget of Tkinter

In Tkinter there is a widget called Canvas, a widget inspired from a real world canvas that is versatile and used for drawing simple shapes to complicated ones. The shapes created in tkinter canvas are coincide each other in a sequential manner, the sequence is in the canvas display list and it's mutable. The options to setup the Canvas widget including size and background color. Canvas widget have properties and methods that are indispensable for creating the application:

- Canvas.create_oval($x0, y0, x1, y1$, options*) for creating vertices
- Canvas.create_line $(x0, y0, x1, y1, \dots, xn, yn, options*)$ for creating edges; the edges goes through the series of points $(x0, y0), (x1, y1), \dots, (xn, yn)$
- *Object ids* and *Tags* for identifying every single objects on canvas and associating a tag with any number of them
- Canvas.coords($tag\ or\ id$, $x0, y0, \dots, xn, yn$) for returning the canvas coordinates of a canvas object if *tag or id* is the only argument passed, else, replacing the canvas coordinates of a canvas object with the new canvas coordinates passed in the argument.

4.3. Overall model

- The application is mainly consist of a canvas widget to draw vertices and edges.
- The drawing of the graphs should be simple, that is, any self-loop or multi-edge must be prevented.
- The layout of graphs must be alterable in order to find any simple drawing of graphs
- The drawing the graphs must include using curved(non-rectilinear) edge

4.4. Feature list

Based on the overall model of the application, the features that were created are as follows:

- Creating vertices for drawing vertices flexibly on the canvas.
- Creating edges for making proper edges and ensuring simple adjacency between vertices
- Moving vertex along with its incident edges for finding any simple and rectilinear drawing of the graph on the plane.
- Bendable edges around vertices to include non-rectilinear drawing of the graphs

4.5. Plan by feature

Creating vertices is the most simple and independent of all other features, whereas the creating edges is the second to it. It is obvious that those two features are required for moving vertex along with its incident edges, which can make the graph layout alterable, enabling the ability to prove rectilinear crossing number of some graph.

Bendable edges around vertices should be the last to be designed and built for it is dependent of all other features, and would need some calibration. The edges or lines will bend when they go through points between and in addition to their endpoints; these points will be called control points. Potent control points will be surrounding the vertices so that the edges can bend around them. This feature will fully realized the research aim of proving crossing number of some graph.

4.6. Design and build by feature

1) Creating vertices

Inside the canvas,

i. If the left mouse is clicked, create a new vertex with the click position as its center and radius predetermined from the coding;
ii. Move the mouse while holding the left mouse to drag the newly created vertex according to the movement;
iii. Release the left mouse to put the vertex in place and create set of points surrounding the vertex as the potent control points of edges to bend around with.

The functions built are *createNode(event), dragNode(event)*, *releaseNode(event)*.

2) Creating edges

Inside the canvas,

i. If the cursor position is on a vertex and the right mouse is clicked, create a new edge with the vertex as its starting point, and the cursor position as its end tip,
ii. Move the mouse while holding the right mouse to drag the end tip of the newly created edge; If the end tip is being dragged near any vertex except the starting point vertex, then the end tip will latch and fixed onto that vertex which will become its end point,
iii. If the right mouse button is released when the edge would create a multi-edge or when it's not fixed onto any vertex as its endpoint, then, the edge will be cancelled.

The functions built for this feature are *createEdge(event)*, *dragEdge(event)*, *releaseEdge(event)*.

3) Moving a vertex along with its incident edges

Inside the canvas,

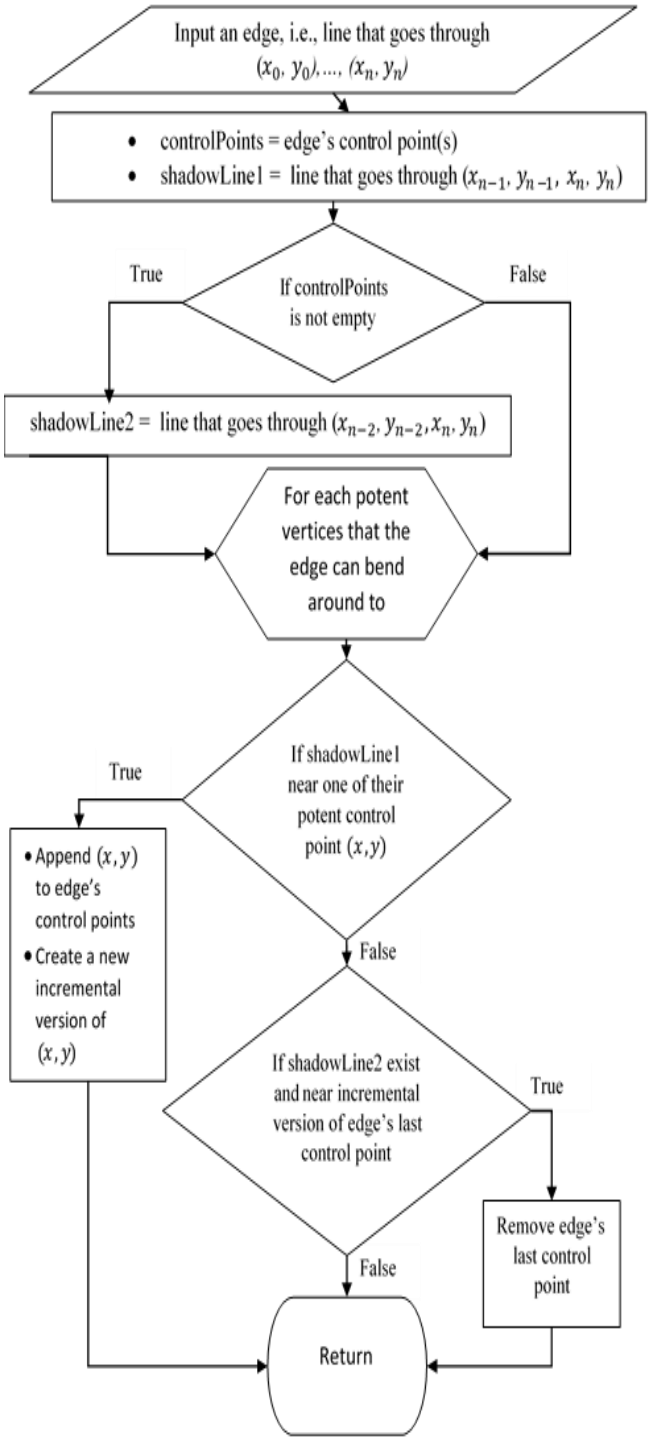
i. If the middle mouse button is clicked on any vertex, that vertex along with its incident edges(if any) are tagged as active vertex and active edges,
ii. Move the mouse while holding the middle mouse button to move the active vertex along with endpoints of the active edges
iii. Release the middle mouse button to put the active vertex and active edge in place, remove the active tags, and update the potent control points surrounding the vertex

The functions built for this feature are *chooseNode(event)*, *moveNode(event)*, and *releaseMovingNode(event)*.

4) Bendable edges around vertices

It is implemented when the edge(s) are being dragged or moved.

i. Input edge to bend or unbend, in other words, to add a control point or remove one if any.
ii. Assign the control points of the edge, including none if it not exist, to a variable "controlPoints".
iii. Assign to variable "shadowLine1" the line that goes through the last two points of the edge.
iv. If "controlPoints" is not empty, then, assign to variable "shadowLine2" the line that goes through the third-from-last point and the last point of the edge.
v. For each vertex, except the start vertex and the vertices near the end tip of the edge:
   a. If "shadowLine1" found near enough a potent control point of those vertices, then, it will be actualized to be a control point of the edge. New potent control



point is created in an outward incremental manner from the vertex.

b. If "shadowLine2" found near enough the new version of the last control point of the edge, then, the last control point of the edge will be removed and it will replace its newer version for being the potent control point again.

The function designed and built for this feature $addremoveControlPoints()$ is embedded in the function $dragEdge()$ and $moveNode()$. Following is the flowchart of adding or removing control points:

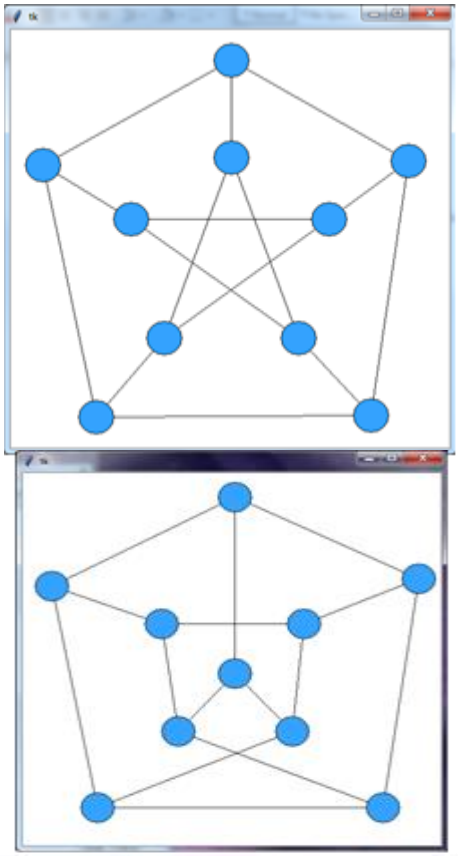
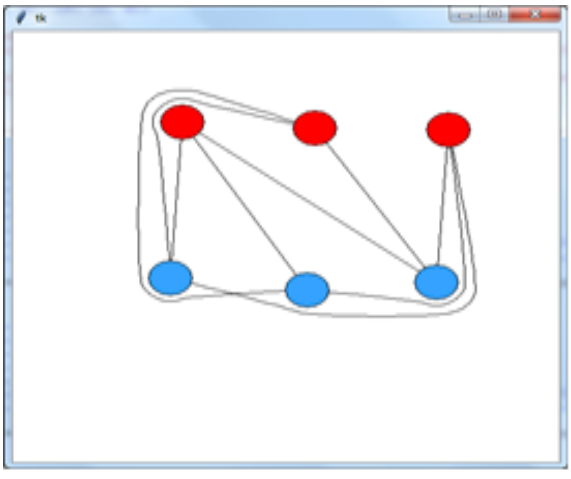


Figure 10. Adding or removing control points flowchart

4.7. Proving $cr(G) \leq k$ of some graph G using the application



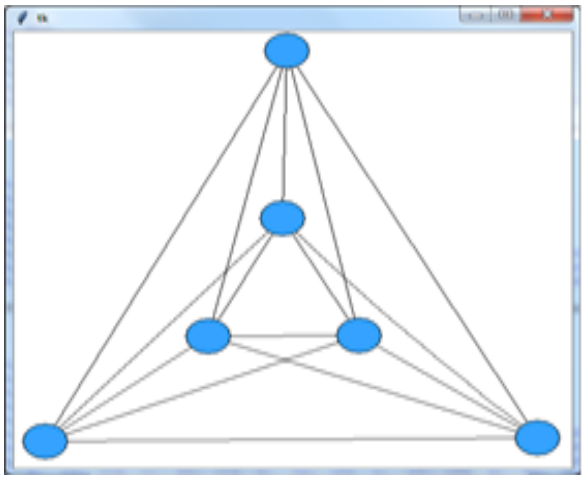


Figure 9. Proof of $K_6 \leq 3$

Figure 11. Proof of $cr(K_{3,3}) \leq 1$

The first graph is a complete bipartite graph that have two disjoint sets of three vertices each, i.e., the graph $K_{3,3}$. The graph was drawn in a way that no pair of vertices in the same set are adjacent, while every pair of vertices in the two different sets are adjacent. The drawing of the graph was with the intention to avoid crossing as much as possible, and it was found that, at most, the amount of crossing in the graph was one, proving $cr(K_{3,3}) \leq 1$.

The second graph is the complete graph with six vertices where each pair of the vertices are adjacent one another; it's the graph of K_6. The drawing of this graph was by having it in a triangular convex hull, as the result, the amount of crossing in the graph was no more than three, proving $cr(K_6) \leq 3$.

The third graph drawn in the application was the Kneser Graph $KG_{5,2}$ whose each vertex uniquely represent a 2-subset of {1,2,3,4,5}, and where two vertices are connected if and only if they correspond to disjoint subsets. $KG_{5,2}$ is also called the Petersen graph. The common and symmetric simple drawing of this graph in the plane is a pentagram within a pentagon, and it has five crossings.

Yet, there is a drawing of this graph that reduced the crossings down to two proving $cr(KG_{5,2}) \leq 2$.

5. CONCLUSION AND SUGGESTION

5.1. Conclusion

The application was developed using the canvas widget with its versatile set of tools in hand and rules in mind. The drawing of graphs is on the plane and prevented from being not simple, i.e. the application won't allow the creation of multi-edge or self-loop. The Graph setup is flexible and its layout alterable, the vertices are able to move along with its incident edges, and the edges are bendable around unincidented vertices.

The application is able to proof $cr(K_{3,3}) \leq 1$, $cr(K_6) \leq 3$, and $cr(KG_{5,2}) \leq 2$ by showing the drawing of graph $K_{3,3}$, K_5, and $KG_{5,2}$ each have crossing(s) no more than 1, 3, and 2.

5.2. Suggestion

- The application is functional to draw and setup graphs, but still lacks in-application features such as coloring, sizing, and labelling.
- Rectilinear drawing is not fully integrated.
- The number of vertices and edges of a graph the application can create with its functionality still intact are subject to the predetermined size of the canvas and vertices, and also computer's performance.

REFERENCES

[1] Abrahamsson, P., O. Salo, J. Ronkainen, and J. Warsta. 2002. Agile software development methods: Review and analysis. VTT publication 478, Espoo, Finland.
[2] Bondy, J., and U. Murty. 1982. Graph Theory with Application. Department of Combinatorics and Optimization of Waterloo University, Ontario, Canada.
[3] Beniz, and A. M. Espindola. 2016. Using Tkinter of Python to Create Graphical User Interface (GUI) fr Scripts in LNLS. Proceedings of PCaPAC2016, Campinas, Brazil.
[4] Fox J., J. Pach, and A. Suk. 2019. Approximating the Rectilinear Crossing Number. Computational Geometry **81**: 45-53, Elsevier.
[5] Garey, M. R., and D. S. Johnson. 1983. Crossing Number is NP-Complete. SIAM J. Alg. Discr. Meth. **4**:3. Society for Industrial and Applied Mathematics.
[6] Gros, J., and J. Yellen. 2004. Handbook of graph theory. CRC Press LLC, Florida, United States.
[7] Guan, X., and W. Yang. 2018. Embedding 5-Planar Graph in Three Pages. Department of Mathematics, Taiyuan University of Technology, Taiyuan Shanxi-030024, China.
[8] Hlineny, P. 2005. On crossing-critical graphs. https://www.fi.muni.cz/~hlineny/papers/crossing-sl-gems05.pdf [Accessed in May 4 2020].
[9] IBM. Software Development. https://www.ibm.com/topics/software-development [Accesed in July 13, 2020]
[10] Leighton, F. 1984. New Lower Bound Technique for VLSI. Mathematical Systems Theory 17: 47-70, Springer-Verlag New York Inc.
[11] Masuda S., K. Nakajima, T. Kawashibara, and T. Fujisawa. 1990. Crossing minimization in linear embedding of graphs. IEEE Transactions on Computers **39**: 1.
[12] Miller B., and D. Ranum. 2014. How to think like a computer scientist: interactive edition. https://runestone.academy/runestone/books/published/thinkcspy/index.html [Accessed in May 4, 2020].
[13] Palmer, S. R., and J. M. Felsing. 2002. A Practical Guide to Feature-Driven Development. Prentice Hall, United States.
[14] Pinontoan B., J. Titaley, and C.E.J.C. Montolalu. 2019. Book Embedding of 3-Crossing-Critical Graphs with Rational Average Degree between 3.5 and 4. IOP Conference Series : Materials Science and Engineering. **567** 012016.
[15] Pinontoan, B., and J. Titaley. 2019. Book Embeddings of Infinite Sequences of Extended Periodic Regular Graphs. IOP Conf. Series: Materials Science and Engineering **621** (2019) 012011, IOP publishing.
[16] Pinontoan, B., and R.B. Richter. 2003. Crossing Numbers of Sequences of Graph II: Planar Tiles. Jurnal Ilmiah Teknik Industri. **42**(**4**): 332-341.
[17] Python Software Foundation. Python Language Reference, version 3.7. Available at http://www.python.org.
[18] Srinath, K. R. 2017. Python – the faster growing programming language. IRJET. **04**:354-355.
[19] Tkinter, https://wiki.python.org/moin/TkInter
[20] Tomasek, J. 2013. Drawing graphs on surface of small genus [thesis]. Computer science institute of charles university, Prague, Czech Republic.
[21] Weisstein, E. "Graph Crossing Number." MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/GraphCrossingNumber.html [Accessed in July 13, 2020]
[22] Williams, L., and A. Cockburn. 2003. Agile Software Development: It's about Feedback and Change. IEEE Computer **36:** 39-43

Gery J. Sumual (gisumual@gmail.com)

Born in Amurang, on November 9, 1998. He pursued a Bachelor's degree in the Mathematics Department, Faculty of Mathematics and Natural Sciences in Sam Ratulangi University. 2020 is the last year of his Bachelor program. This paper is the result of the published thesis research he admitted as one of the requirements to obtain his bachelor degree.

Benny Pinontoan (bpinonto@gmail.com)

Born in Bitung, North Sulawesi, Indonesia, and lives in Manado. Completed his Bachelor's degree in Technische Informatica at Faculteit Informatica Eindhoven, the Netherlands in 1993. In 2002 he completed his Doctoral degree in Mathematics at School of Mathematics and Statistics, Carlethon University Ottwa Canada. In 1995 he was appointed as a lecturer at Sam Ratulangi University and since 1st March 2006 has become a Profesor of Mathematics at Faculty of Mathematics and Natural Science, Sam Ratulangi University.

Luther A. Latumakulita (lutherlatu@gmail.com)

Born in Ambon, September 14th, 1971, Indonesia. Completed his Bachelor and Master degree in Computer Science at the department of Computer Science of Gadjah Mada University in 1997 and 2008 respectively. In 2018, he graduated from the department of computer science and electrical engineering in Kumamoto University Japan earning his Doctoral degree. Since 2008, He has been tenuring as a lecturer in the faculty of

Mathematics and Natural Sciences, Sam Ratulangi University.