University of California, Santa Barbara

Senior Thesis

Dots and Lines

A Survey of Graph Drawing Algorithms

Vincent La

supervised by
Karel Casteels

June 15, 2018

Contents

Acknowledgements

I would like to thank Karel Casteels for his support and guidance, as well as his excellent introduction to graph theory class provided the original spark for writing this paper. Furthermore, while I am shocked that in the year 2018 it is still possible for a widely cited computer science paper (1829 citations on Google Scholar as of the time of this writing) to only be available in book format, I would like to thank the staff of the UCSB library for helping me track down a 1984 copy of *Congressus Numerantium* containing Peter Eades' "A Heuristic for Graph Drawing".

Lastly, I would like to acknowledge the 2017-2018 Golden State Warriors whose commitment to excellence inspired me to see this paper to its completion.

Chapter 1

Introduction

Graph theory is one of the most widely applicable fields of math, and as such there has been a large demand for different methods of visualizing graphs. Just as the types and uses of graphs are wide and varied, there are also a variety of different algorithms for visualizing graphs. These algorithms vary in the type of graphs they will draw, and the guarantees they may provide, e.g. planarity and symmetry. This paper will further explore some of the algorithms detailed by *Graph Drawing: Algorithms for the Visualization of Graphs.* [see 1, page 12]

All proofs, and associated errors, are my own unless otherwise noted.

1.1 Computer Science Basics

Although this is a paper on applied math,[1] because it also focuses on the analysis of algorithms, some computer science fundamentals are required.

1.1.1 Complexity

When analyzing algorithms, one feature we want to analyze is how complicated they can get as the input size n increases. For example, suppose we want to count the number of occurrences of 5 in an unsorted list. Because the list is unsorted, we do not know in general where the 5's are so we have to scan every item in the list. Hence, the running time $T(n)$ of this algorithm is based on the size of the list n and we say this algorithm runs in in linear time.

Definition 1.1.1 ($\Omega(f(n))$)**.** *We say an algorithm is $\Omega(f(n))$ if there is some positive constants a, n_0 such that $a \cdot f(n) \leq T(n)$ for all $n \geq n_0$.*

Definition 1.1.2 ($O(f(n))$)**.** *We say an algorithm is $O(f(n))$ if there is some positive constants b, n_0 such that $b \cdot f(n) \geq T(n)$ for all $n \geq n_0$.*

Definition 1.1.3 ($\Theta(f(n))$)**.** *We say an algorithm is $\Theta(f(n))$ if it is $\Omega(f(n))$ and $O(f(n))$.*

[1] As the author claims

What Gets Analyzed?

Usually, algorithms are analyzed for their time complexity (how long it takes to compute something). However, they may also be analyzed for their space complexity (memory usage), and any other aspect of their performance. For example, with respect to graph drawing, algorithms may be analyzed for the amount of screen real estate they take up.

Chapter 2

Tree Drawing Algorithms

2.1 Motivation

Many types of graphs are used in computer science, and one of the most common is the tree–especially binary trees. Binary trees–trees with at most two nodes–form the basis for many data structures in computer science. For example, when many people thinking of storing a collection of items to be searched later, the naive solution is to store them in a list. When checking to see if an item is part of this collection, a naive solution is to simply iterate through the list, comparing our item against each element of the list. In other words, for a collection of n items, we have to perform n comparisons.

However, a binary search tree is constructed such that for every node, every child node on its left is less than the value at the root, and vice versa for the right side. Hence, when searching for an item, if our item is less than the value at the current node, then we only need to keep searching in the left subtree. As a result, we can find an item we want with on average just $\log_2 n$ comparisons. As a result of the many implementations of trees in computers, many have discussed ways of visualizing these data structures.

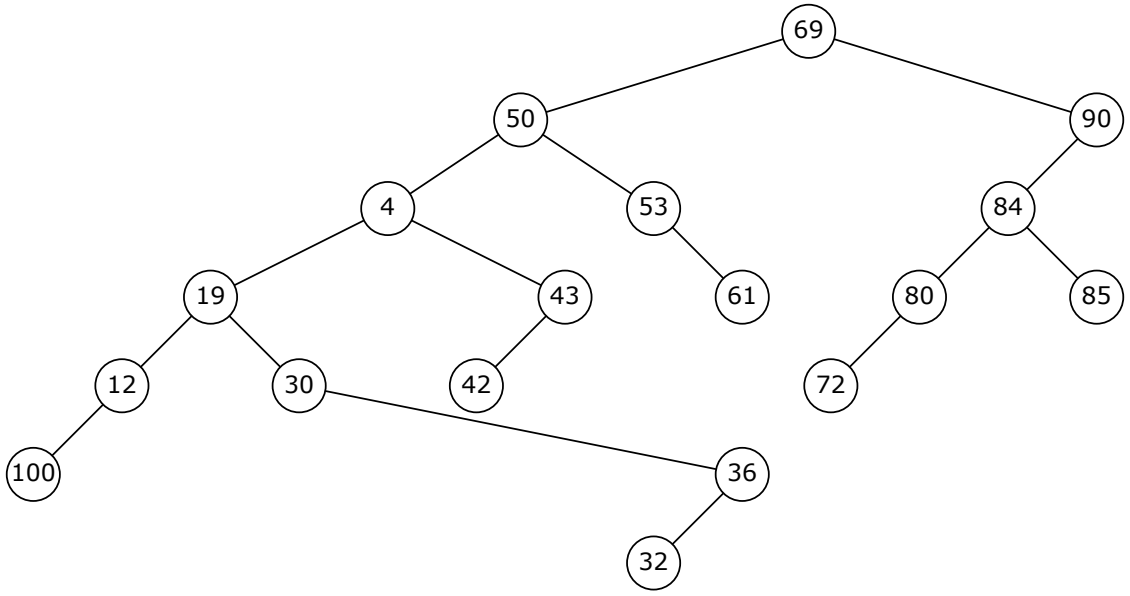


Figure 2.1: A binary search tree with 20 items

2.1.1 Definitions

Before talking about trees, it would behoove us to build a common vocabulary about them.

Definition 2.1.1 (Level (of a node))**.** *The level (or depth) of a node in a binary tree is the number of edges between that node and the root. By convention, the root is on level 0.*

Definition 2.1.2 (Height (of a tree))**.** *The height of a binary tree is the number of edges between the root and the deepest leaf.*

Definition 2.1.3 (Perfect Binary Tree)**.** *A perfect binary tree is a binary tree where all leaf nodes are on the same level, and all other nodes have two children.*

Tree Traversal

When implementing trees as computer data structures, there are a multitude of ways to traverse them.

Definition 2.1.4 (Level Order Traversal)**.** *In a level order traversal, we move from top to bottom, left to right. In other words, a level order traversal is a breadth-first traversal of a tree.*

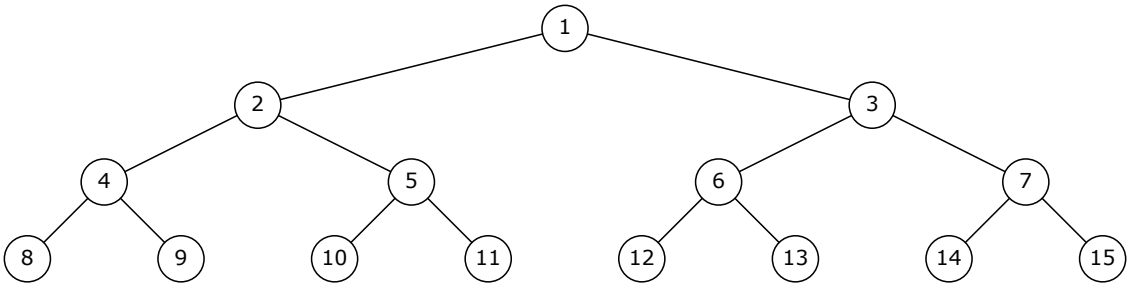


Figure 2.2: A perfect tree of height 3, where the numbers correspond to the order in which nodes are visited in a level order traversal

Definition 2.1.5 (Preorder Traversal)**.** *In a preorder traversal, the root is visited first. Then, the procedure is called recursively on the left subtree, and then recursively on the right subtree. A preorder traversal is a form of depth-first search.*

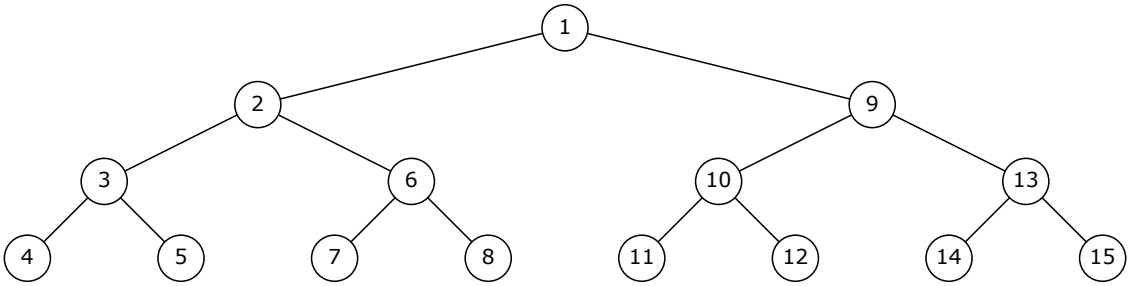


Figure 2.3: A preorder traversal in a perfect tree of height 3

2.2 Tree Drawing as a Linear Program – Supowit and Reingold (1983)

2.2.1 The Problem of Narrow Trees

Reingold and Tilford (1981) [6] gave a linear time algorithm for drawing binary trees satisfying the six aesthetics described below, while also trying to keep each subtree as compact as possible. However, it was soon discovered that locally minimizing the width of every subtree does not achieve the goal of a minimum width drawing overall. As shown in Figure 2.4, sometimes in order to draw a tree as narrow as possible, some subtrees have to be drawn non-optimally.

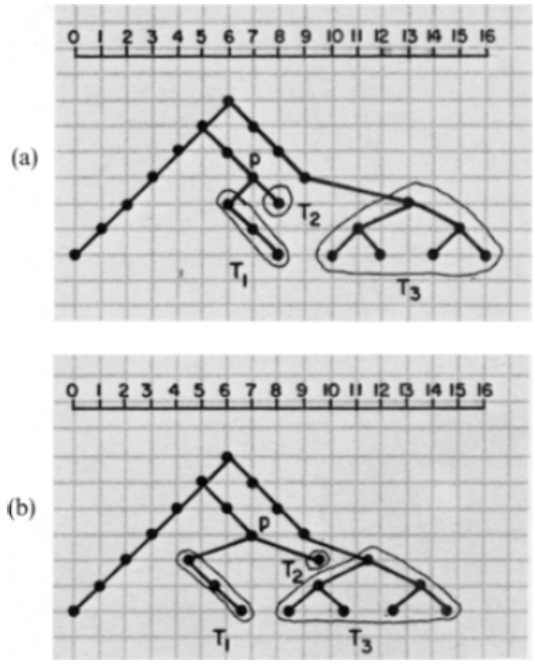


Figure 2.4: A figure from Supowit and Reingold displaying a case of suboptimality from RT81. Although the subtrees T_1 and T_2 are drawn compactly in (a) by RT81, (b) shows that a narrower drawing overall is possible if T_1 and T_2 are allowed to be farther apart.

2.2.2 Aesthetic Trees

In the algorithm below, the problem of drawing a tree is formulated as a linear program.[7] Although linear time algorithms exist for drawing trees, they do not necessarily produce "nice" drawings. In Supowit and Reingold's paper, the problem of drawing a tree can be described as finding the narrowest possible drawing such that it matches several aesthetics:

1. **Layering** All nodes at the same level, i.e. the number of edges between a node and the root, share the same y-coordinate
2. **Child Positioning** Each left child is placed strictly to the left of its parent, and each right child is placed strictly to the right of its parent
3. **Separation** For any two nodes at the same level, they must be placed at least 2 units apart.
4. **Centering** If a parent has two children, then it must be centered over them
5. **Planarity** No two tree edges may be drawn such that they intersect, unless they same a common vertex
6. **Isomorphic Trees Drawn Similarly** If two subtrees are isomorphic, they must be drawn identically (minus a translation)

As Theorem 2.2.1 shows, enforcing aesthetics 1, 2 and 3 are sufficient to guarantee 5.

By formulating the problem using linear programming, the time complexity of tree drawing can be reduced to polynomial time.[7]

2.2.3 Formal Description

First, define f to be a mapping from a tree's vertex set to $\mathbb{R}^2$. Trivially, we can satisfy Aesthetic 1 by defining $f_y(n) = -i$ where i is the level of node n. Then, we use a linear program to determine the value of $f_x(n)$. We will introduce two auxiliary variables x_{max} and x_{min} which give the left and right bounds for the x-coordinates of our drawing respectively. Hence, our goal of creating the narrowest possible drawing may be expressed as

$$\min x_{max} - x_{min}$$

subject to:

$$f_x(n) \geq x_{min}$$

$$f_x(n) \leq x_{max}$$

$$f_x(n) - f_x(\text{left child}\,(n)) \geq 1 \qquad \text{For all } n \text{ with a left child (Aesthetic 2)}$$

$$f_x(\text{right child}\,(n)) - f_x(n) \geq 1 \qquad \text{For all } n \text{ with a right child (Aesthetic 2)}$$

$$f_x(n) - f_x(m) \geq 2 \qquad n \text{ is the level-order successor of } m \text{ (Aesthetic 3)}$$

$$\frac{f_x(\text{left child}\,(n)) + f_x(\text{right child}\,(n))}{2} = f_x(n) \qquad \text{For all } n \text{ with two children (Aesthetic 4)}$$

Lastly, to implement aesthetic 6, for every node in the tree, we determine its rank and the number of in the subtree rooted at that node. Hence, we can partition every node in the tree into an equivalence classes where we say–for any nodes m, n–m is equivalent to n if they have the same size and rank. Then, for each equivalence class with more than one member, we require the constraints. Let $\{n_1, n_2, ..., n_k\}$ be the members of some equivalence class. Then, we require the constraints

$$f_x(\text{right child}\,(n_i)) - f_x(n_i) = f_x(\text{right child}\,(n_{i+1})) - f_x(n_{i+1}) \tag{2.1}$$

$$f_x(n_i) - f_x(\text{left child}\,(n_i)) = f_x(n_{i+1}) - f_x(\text{left child}\,(n_{i+1})) \tag{2.2}$$

Because, corresponding subtrees in isomorphic trees are also isomorphic, the equations above are sufficient to satisfy Aesthetic 6. (Reword this)

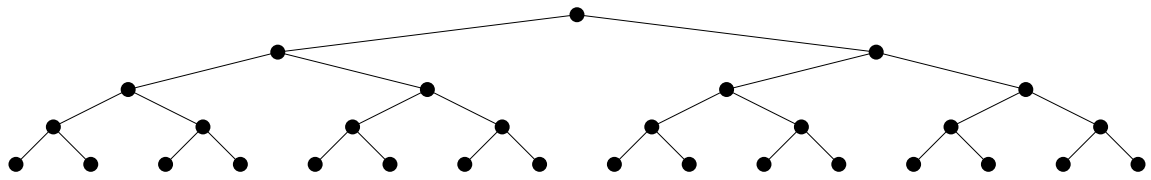


Figure 2.5: A drawing of a perfect tree of height 4

Theorem 2.2.1 (Planarity)**.** *Let T be any binary tree drawn by this algorithm. Then, aesthetics 1, 2, and 3 imply aesthetic 5.*

Proof. First, if every node n on some level i is drawn according to the rule $f_y(n) = -i$, then ensuring planarity simply means ensuring nodes on the same level do not cross. We will show via induction on the height of the tree that given aesthetic 1, aesthetics 2 and 3 then imply 5.

Base Case Trivially, this algorithm gives a planar drawing for a tree of height 0.

Figure 2.6: The full tree of height 0 as drawn by this algorithm

Inductive Hypothesis Assuming that the previous $i - 1$ levels of the tree are drawn in a planar fashion, we will show that for any node in the i^{th} level, the edge between it and its parent does not intersect any other edges.

First, suppose for a contradiction that there is some edge crossing. This implies one of three cases. It may be because some node's right subtree crosses some other node's left subtree or vice-versa.

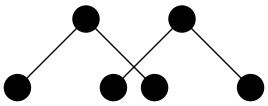


Figure 2.7: A right subtree crossing with a left subtree

However, this implies a violation of aesthetic 3 as the left subtree of the right parent should be drawn to the right of the right subtree of the left parent. Moreover, there may be a crossing since some node's left subtree crosses some other node's left subtree. However, this either implies the subtrees are drawn directly on top of each other (a violation of aesthetic 3), or that a "left" subtree is drawn to the right of its parent, which is a violation of aesthetic 2.

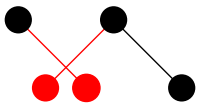


Figure 2.8: Here, two "left" subtrees (colored in red) cross each other

Lastly, it may be that some node's right subtree crosses some other node's right subtree. But, with logic similar to the case above, this either implies a violation of aesthetic 2 or 3.

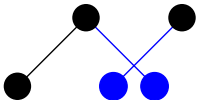


Figure 2.9: Here, two "right" subtrees (colored in blue) cross each other

□

Theorem 2.2.2 (Number of Constraints)**.** *The number of constraints is $O(n)$.*

Proof. Let T be a binary tree with n nodes. First, the requirements $x_{min} \leq f_x(n)$ and $f_x(n) \leq x_{max}$ contribute $2n$ constraints.

Aesthetic 2 For any tree, every node except the root is either a left or right son. Hence, there aesthetic 2 contributes $n - 1$ constraints.

Aesthetic 3 For every level, this aesthetic requires a constraint for every adjacent pair of nodes. There are less pairs than nodes, so this quantity is less than n.

Aesthetic 4 For any tree, there are less parents than total nodes, so this aesthetic requires no more than n constraints.

Aesthetic 6 There less than $\frac{n}{2}$ equivalence classes (excluding the equivalence class of leaves) with more than one member, so this constraint contributes no more than $\frac{n}{2}$ constraints.

Hence, we can see that there are at most $6n$ aesthetic constraints, so the number of constraints is $O(n)$ as required. □

Theorem 2.2.3 (Number of Constraints of a Perfect Tree)**.** *For a perfect tree of height h, this algorithm requires*

$$\left(4\sum_{i=0}^{h} 2^i\right) - 2h - 2^{h+1} - 3$$

constraints.

Proof. To begin, in a perfect binary tree, the number of nodes at every level doubles. Hence, in a perfect tree of height h, there are $\sum_{i=0}^{h} 2^i$ nodes. Because all of the constraints are somehow related to this number, we'll denote this quantity as t.

Width Constraints For every node in the tree, a pair of constraints is required for the auxiliary variables x_{max} and x_{min}. Hence, there are $2t$ of these constraints.

Aesthetic 2 Every node in the tree except the root is either a left or a right son, so this aesthetic requires $t - 1$ constraints.

Aesthetic 3 For each level in this tree with $k > 1$ nodes (i.e. every level except the one containing the root), this constraints necessitates $k - 1$ constraints. Hence, we need

$$(2^1 - 1) + (2^2 - 1) + ... + (2^h - 1) = t - (h + 1)$$

constraints to satisfy this aesthetic.

Aesthetic 4 Every node in a perfect tree except for those at the last level have two children, hence we need

$$2^0 + 2^1 + ... + 2^{h-1} = t - 2^h$$

constraints to satisfy this aesthetic.

Aesthetic 6 For any given level, every node in that level is isomorphic to other nodes in that level. In other words, each level of the perfect tree except for the root forms an equivalence class. Recall, for the purpose of aesthetic 6 we ignore every leaf node. Hence, the equivalence classes are composed of every node in the tree except for the root and last level. For each level, we add a constraint for every pair of adjacent nodes. Hence, we need

$$(2^1 - 1) + ... + (2^{h-1} - 1) = t - (2^h + h + 1)$$

constraints for this aesthetic.

Adding this all up, we get $4t - 2h - 2^{h+1} - 3$ constraints. □

2.2.4 Implementation

In this paper, this algorithm was implemented as a C++ program based around the GNU Linear Programming Kit (GLPK)–an open-source linear optimizer based on the simplex method.[5] Each node (in the C++ implementation–struct TreeNode) was represented as a data structure as follows:

left	Pointer to left child
right	Pointer to right child
id	Corresponding column index of constraint matrix
data	Self-explanatory

Then, aesthetics 1-5 could be computed using $2n$ iterations. First, an iterative level-order traversal was used to simultaneously assign IDs to each node, determine the number of left and right sons, and thus, determine the total number of rows and columns in the constraint matrix. By using an iterative method, our traversal took linear as opposed to quadratic time. During the level-order traversal, a hash table mapping each level of the tree to an array of node pointers was populated. [1] Then, a final iteration over this auxiliary data structure was used to calculate the aesthetics.

[1] Implemented in C++ as std :: unordered_map<int, TreeNode*>

It should be noted that regardless of method used, computing the required aesthetics requires at least two iterations, where the first iteration is used to assign column indicies to nodes. If the first iteration is a level-order traversal, then we cannot calculate any constraints related to a node's children until a second traversal–since the children do not have column indices assigned. On the other hand, if an initial preorder traversal is used, then we will still need a second iteration to calculate the third aesthetic since we are not aware of all of the nodes on one level during a preorder traversal.

Finally, in order to populate the auxiliary data structure, for every node, we also had to keep track of which level it was on. To accomplish this during an iterative level-order traversal, each node pointer was paired with its current level (e.g. (std :: pair<int, TreeNode*>)). As shown in the pseudocode below, each pointer's associated level can be determined when we examine its parent.

Algorithm 1 Tree Layout – Column Index Assignment

1: **procedure** ASSIGN_IDS(t)
2: Create a queue *children* initially populated with the pair $(0, root)$ where 0 is the level of root
3: $num\_nodes \leftarrow 0$ ▷ Information for constraint matrix
4: $left\_sons \leftarrow 0$
5: $right\_sons \leftarrow 0$
6: $current\_id \leftarrow 0$ ▷ Corresponding column of constraint matrix
7: **while** *children* is not empty **do** ▷ Level-order traversal
8: Pop the first element from *children* and assign it to *current*
9: $current\_level \leftarrow current.first$ ▷ *current* = (level (integer), node pointer)
10: $current\_node \leftarrow current.second$
11: $current\_node \rightarrow id \leftarrow current\_id$
12: **if** $current\_node \rightarrow left$ is not null **then**
13: $children \leftarrow (current\_level + 1, root \rightarrow left)$ ▷ Determine level of child
14: $left\_sons \leftarrow left\_sons + 1$ ▷ Keep count of left sons (aesthetic 2)
15: **end if**
16: **if** $current\_node \rightarrow right$ is not null **then**
17: $children \leftarrow (current\_level + 1, root \rightarrow right)$ ▷ Determine level of child
18: $right\_sons \leftarrow right\_sons + 1$ ▷ Keep count of right sons (aesthetic 2)
19: **end if**
20: $levels[current\_level].push\_back(current\_node)$ ▷ Update mapping of levels to their nodes
21: $current\_id \leftarrow current\_id + 1$
22: **end while**
23: **end procedure**

After the column indices have been assigned, we can then easily calculate aesthetics 1 through 5 using the mapping *levels* from algorithm 1.

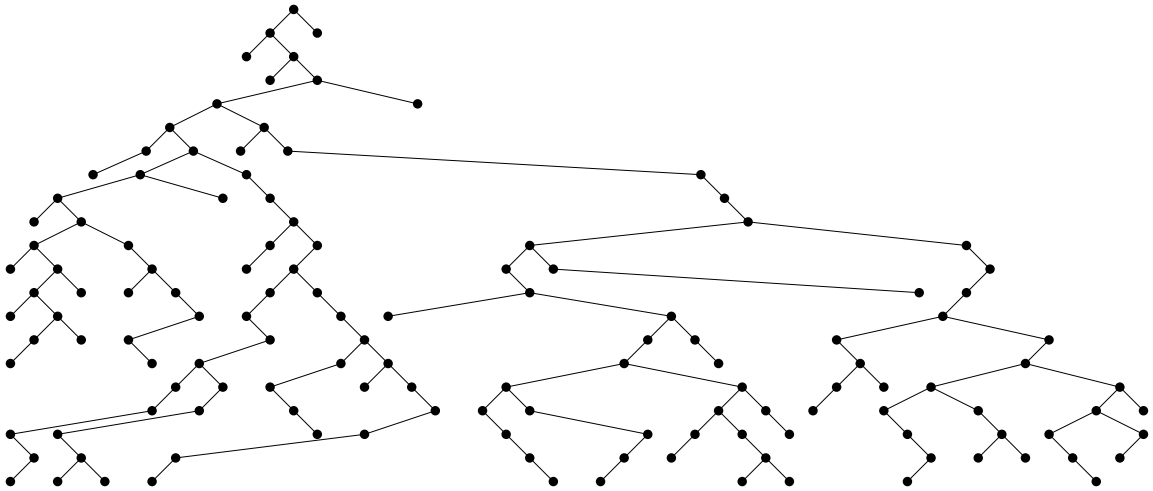
Algorithm 2 Tree Layout – Calculating Aesthetics 1-5

```
1: procedure CALCULATE_CONSTRAINTS(levels)
2:     for level l = 0, ..., h do
3:         for node i = 0, ..., m do  ▷ Iterate over nodes at level l, where m is the number of nodes at level l
4:             Let current_node refer to levels[l][i] (the i^th node at level l)
5:             Add constraint current_node ≤ x_max                    ▷ Add width constraints
6:             Add constraint current_node ≥ x_min
7:             if current_node → left is not null then
8:                 Add second constraint
9:             end if
10:            if current_node → right is not null then
11:                Add second constraint
12:            end if
13:            if current_node → left and current_node → right are not null then
14:                (Add parent centered constraint)
15:            end if
16:            if i < n then
17:                Add separation constraint
18:            end if
19:        end for
20:    end for
21: end procedure
```

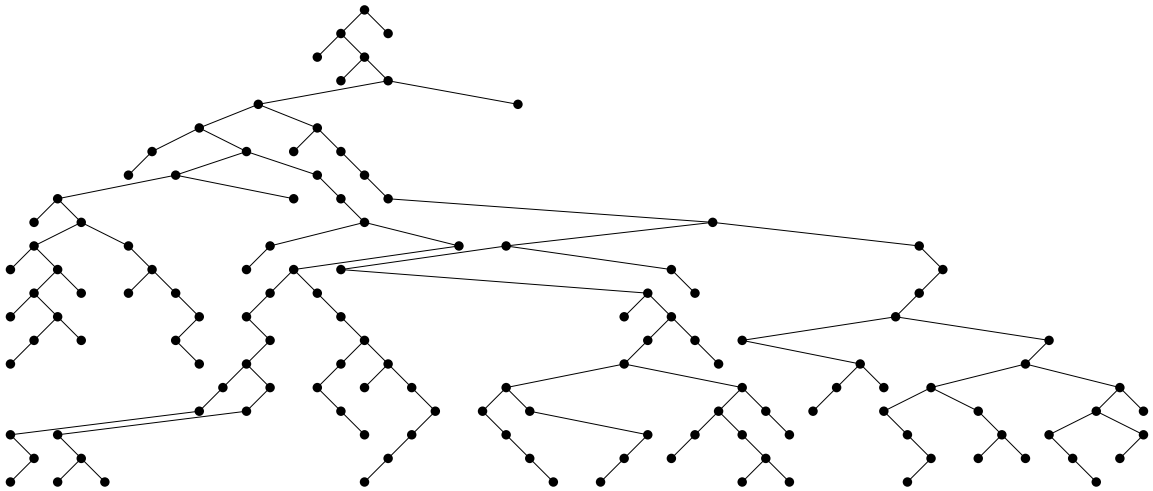
Thus, calculating the first five aesthetics can be done in linear time by iterating over each node just twice.

2.2.5 Aesthetic 6 – Drawing Isomorphic Trees Identically

As mentioned by the paper's authors, there is some debate about whether or not this should be considered an aesthetic. However, sometimes visualizations are used to find recurring patterns, and enforcing this aesthetic allows humans to more effectively use their pattern recognition abilities. In Figure 2.10, the same tree is drawn twice by this algorithm, where aesthetic 6 is relaxed in the first drawing.



(a) An incomplete tree of height 20 as drawn by this algorithm satisfying aesthetics 1-5



(b) An isomorphic tree as drawn by this algorithm satisfying aesthetics 1-6

Figure 2.10: Two drawings of isomorphic trees differing only in the application of aesthetic 6

Rank of a Tree

In order to efficiently draw isomorphic trees identically, we define a function $rank()$ which assigns a unique integer to every binary tree. We must be careful and note that this number is unique among trees of the same size, as trees of different size may be assigned the same number. Hence, we can say two subtrees are isomorphic if they have the same size and rank.

Knott showed that this function is one-to-one (among trees of the same size) and that the numbering it generated corresponded to an ordering among trees. [4] For even n, there are $B_n = \frac{\binom{2n}{n}}{n+1}$ different binary trees, so we can say $rank(t)$ is a mapping from the trees t of size n to the integers $1, 2, ..., B_n$.

Algorithm 3 Tree Layout – Calculating the Rank of Each Node

1: **procedure** RANK(t, *cache*)
2: **if** t is null **then**
3: $rank \leftarrow 1$
4: **else**
5: $rank \leftarrow B_n(r(t)) \times rank(l(t) - 1) + rank(r(t)) + \sum G_{j,n}$
6: **end if**
7: $cache[size(t)][rank].push\_back(tree)$
8: **return** $rank$
9: **end procedure**

In the algorithm above, the $rank()$ of a tree is computed recursively, moving from the bottom up. Hence, calling $rank()$ on the root node has the effect of computing the $rank()$ of every node in the tree. As a result, it behooves us to cache the calls of $rank()$. Specifically, in this implementation, the cache is a mapping of a subtree's size and rank to an array of trees of similar size and rank (i.e. isomorphic subtrees).[2] Hence, implementing the sixth aesthetic is simply a matter of iterating over the arrays of this cache.

[2]In C++, the author choose to implement the cache as
std :: unordered_map<int, std::unordered_map<int, std::vector<TreeNode∗>>>)

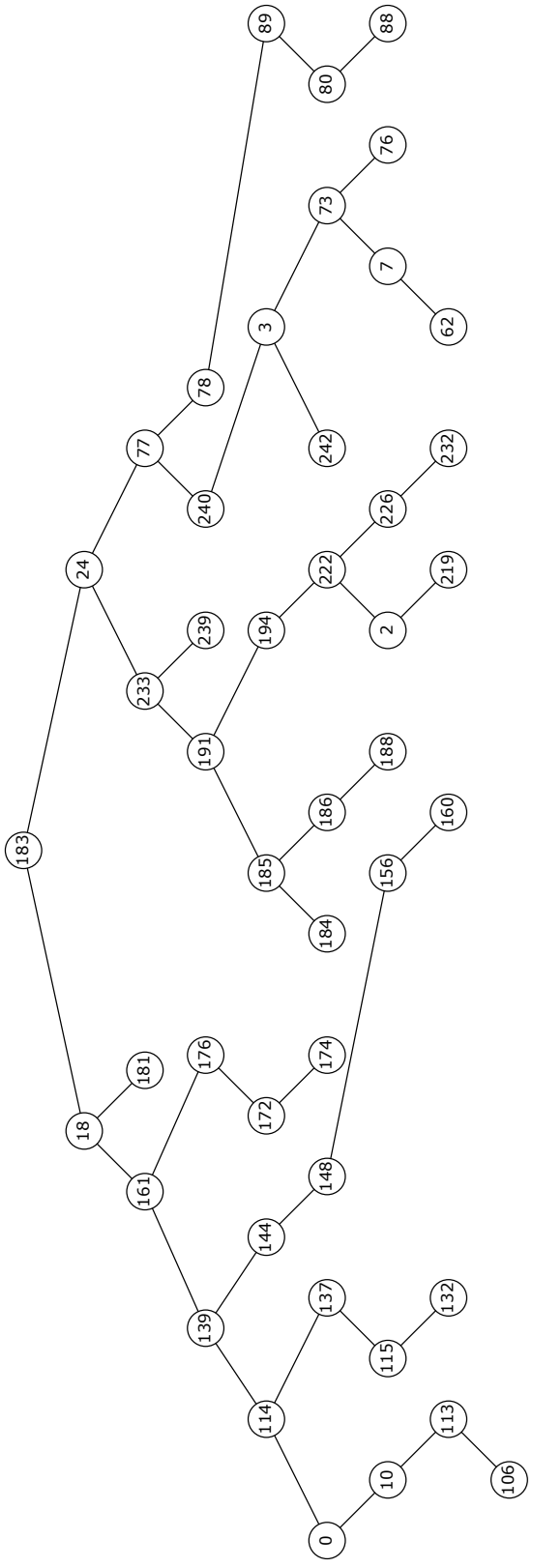


Figure 2.11: A binary search tree with 50 items

Chapter 3

Force Directed Algorithms

3.1 Introduction

Force directed algorithms attempt to draw graphs by relating them to some physical analogy. For example, we may view vertices as steel balls and the edges between them as springs. One of the earlier force directed algorithms, Tutte's Barycenter Algorithm, attempts to place a graph's nodes along its "center of mass."

3.1.1 Definitions and Notation

Although there are plenty of force-directed algorithms, they all have the final result of mapping the vertices of a graph to $\mathbb{R}^2$. Hence, for the rest of the chapter we will use the shorthand $p_v = (x_v, y_v)$.

Furthermore, force-directed algorithms may simply be viewed as setting up mathematical models which can then be solved iteratively. Using various numerical methods, such as a Newton-Raphson iteration, the positions of the vertices are iterated upon until they converge. However, this implies we need to define convergence in a way dumb enough for a computer to understand.

Definition 3.1.1 (Convergence)**.** *Suppose we are applying some force-directed algorithm f to a graph with vertex set V. For any $v \in V$, let p_{i-1}, p be the mapping of v to $\mathbb{R}^2$ during the $i-1$ and i^{th} iteration of f respectively. We say that f has converged if for all $v \in V$, $|p - p_{i-1}| < \epsilon$, where $\epsilon > 0$ is some small number. (In practice, we can define ϵ to be a very small positive number like* 0.01*).*

3.2 Eades' Spring System

3.2.1 Motivation: A Heuristic Method

Drawing graphs poses many difficulties, such as disagreement over what makes a drawing "aesthetic," and the difficulty of finding and the displaying the various isomorphisms of a graph. Furthermore, while the authors of this method desired drawings where all edges were about equally long, others have discovered satisfying that aesthetic to be NP-hard.

Because of the many difficulties of drawing graphs, the authors sought to define a heuristic method.

3.2.2 The Model

In this algorithm, edges are modeled as springs while vertices are replaced by steel rings, thus creating the analogy between graphs and physical systems. Furthermore, vertices are also given an electrical force so they repel each other. [2] Formally, the force on each vertex v is

$$F(v) = \sum_{(u,v)\in E} f_{uv} + \sum_{(u,v)\in V\times V} g_{uv} \tag{3.1}$$

where f is the force of a spring acting between adjacent vertices and g is the electrical force acting between all pairs of vertices. If we use Hooke's law springs and electrical forces which follow an inverse square law, we can write this as

$$\sum_{(u,v)\in E} \kappoint{1}_{uv}(d(p_u,p_v) - \ell_{uv})\frac{x_v - x_u}{d(p_u,p_v)} + \sum_{(u,v)\in V\times V} \frac{\kappoint{2}_{uv}}{(d(p_u,p_v))^2}\frac{x_v - x_u}{d(p_u,p_v)} \tag{3.2}$$

where $d(p_u, p_v)$ is the Euclidean distance between p_u and p_v and

- $\kappoint{1}_{uv}$ determines the stiffness of springs
- $\kappoint{2}_{uv}$ determines the strength of the electrical field between vertices
- ℓ_{uv} gives the length of a spring under zero tension

All of the spring drawings in this paper were drawn with $\kappoint{1}_{uv} = 400$, $\kappoint{2}_{uv} = 2$, and $\ell_{uv} = 1$.

3.2.3 Alternative Formulation with Logarithmic Springs

In Eades' 1984 paper, he uses logarithmic springs because "experience shows that Hookes Law (linear) springs are too strong when the vertices are far apart." [2] Under this formulation, we can rewrite the x-component of 3.1 as

$$\kappoint{1}_{uv} \log\left(\frac{d(p_u,p_v)}{\ell_{uv}}\right)\frac{x_v - x_u}{d(p_u,p_v)}$$

However, a passage in the later *Graph Drawing: Algorithms for the Visualization of Graphs* (which Eades co-authored) asserts "experience has shown that it is difficult to justify the extra computational effort by the quality of the resulting drawings" in regards to logarithmic springs.

3.2.4 Algorithm

A very simple algorithm for finding the system's equilibrium is to move each vertex in the direction of the force in a small proportion to the magnitude of the force.

Algorithm 4 Eades' Heuristic for Graph Drawing

```
1: procedure LAYOUT(G)
2:     while algorithm has not converged do
3:         for each vertex v do
4:             Compute the force f on v
5:             Move v in the direction of f by a small proportion to the magnitude of f
6:         end for
7:     end while
8: end procedure
```

In this paper, the "small proportion" used was 10%, just as it was in Eades' 1984 paper. Furthermore, the original paper also uses a fixed number of iterations (100) instead of checking for convergence, since it was observed that most graphs obtain equilibrium within 100 iterations.

Algorithm Trace for K_8

The algorithm described above, while not the most efficient, allows us to create animations of the vertices moving into place.

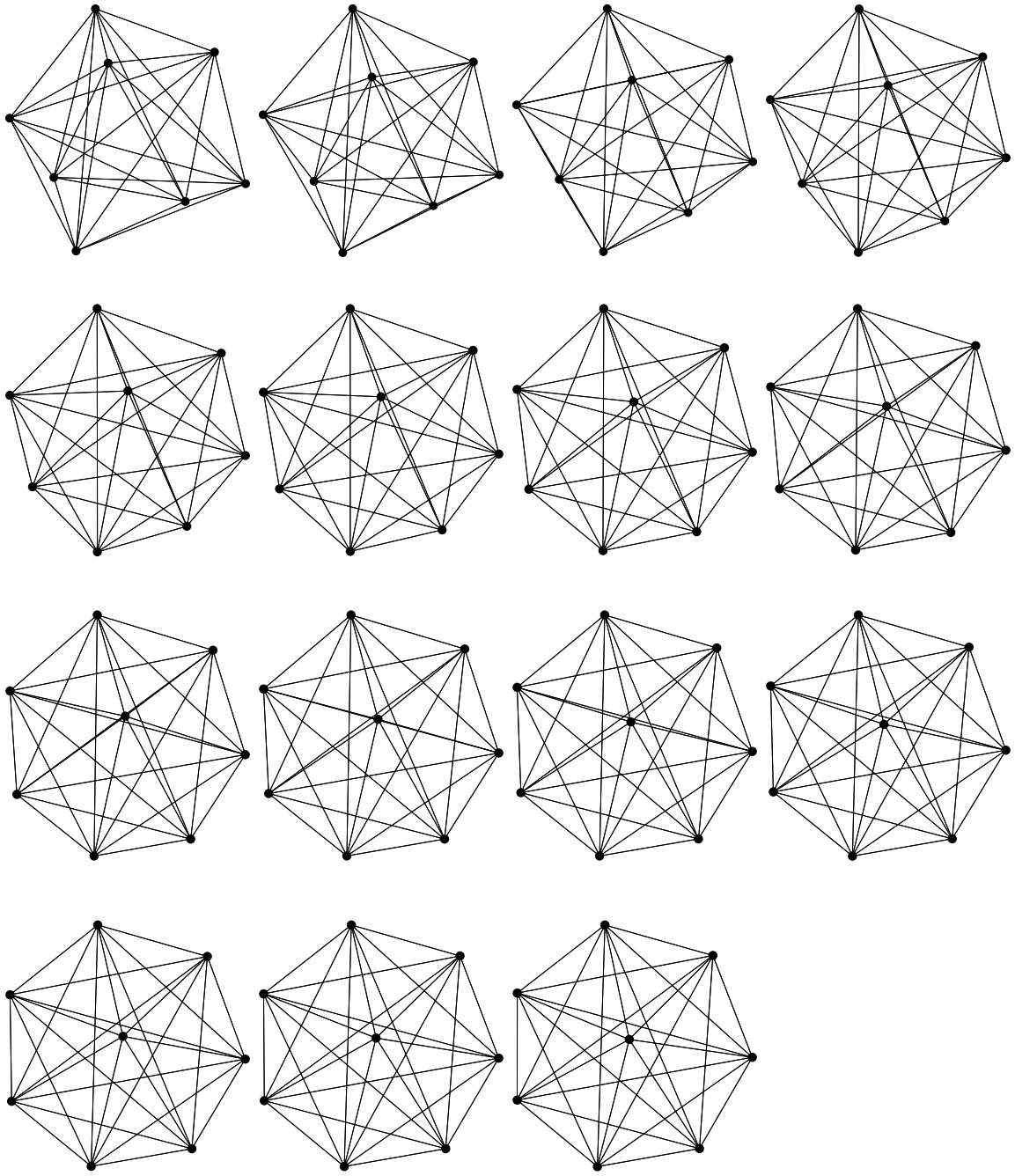


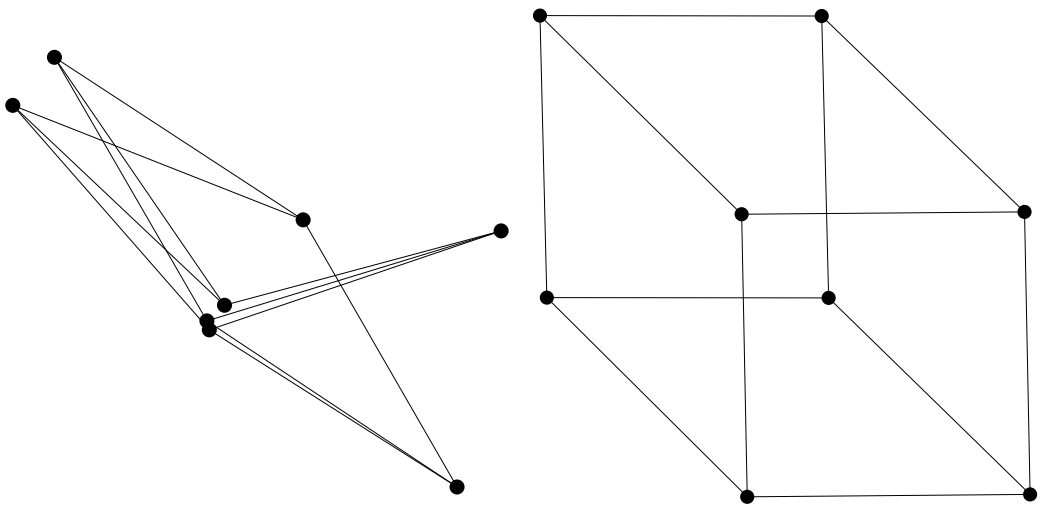
Figure 3.1: The complete graph K_8 drawn by Eades' algorithm

3.2.5 Observations

The author notes that this algorithm tends . Furthermore, this algorithm tends to do poorly with dense graphs, with one ironic exception–the complete graph K_n (when n is small).

Sensitivity to Initial Positions

As mentioned earlier, this algorithm is a heuristic, and the final drawing is heavily influenced by the initial positioning of the vertices as shown by Figure 3.2 below.



(a) Vertices are initially positioned uniformly at random about the canvas

(b) Vertices are initially positioned somewhat close to where they would be in a standard drawing of the cube

Figure 3.2: Two drawings of the cube Q_3

In the 1984 paper, Eades mentions his heuristic may be used as an approximation to a final layout, and that drawings may be fine-tuned with an accompanying editor. The figure above supports this practice, as it shows that while this algorithm may not be capable of producing aesthetic layouts on its own, it can do a great job with just a little human intervention.

3.3 Tutte's Barycenter Method

An early force directed drawing method was Tutte's Barycenter Method. [8] In this method, the force on every vertex v is given by

$$F(v) = \sum_{(u,v)\in E} (p_u - p_v) \tag{3.3}$$

Hence, splitting (3.3) across the x and y dimensions we get

$$\begin{aligned} \sum_{(u,v)\in E} (x_u - x_v) &= 0 \\ \sum_{(u,v)\in E} (y_u - y_v) &= 0 \end{aligned} \tag{3.4}$$

3.3.1 Fixed Vertices

However, notice the system in (3.4) has the trivial solution $(x, y) = (0, 0)$ for all vertices, which gives a very poor drawing! Hence, we take $n \geq 3$ vertices such that they form a convex polygon, and fix them.

In this paper's implementation of the algorithm, the fixed vertices are positioned as n equally spaced points along a circle of radius equal to half the width of the final image, centered at the origin.

3.3.2 Linear Model

Suppose for some free vertex v, we denote the set of fixed neighbors as N_0, and the set of free neighbors as N_1. Then, we may rewrite the above equations as

$$\begin{aligned} \deg(v)x_v - \sum_{u \in N_1(v)} x_u &= \sum_{w \in N_0(v)} x_w^* \\ \deg(v)y_v - \sum_{u \in N_1(v)} y_u &= \sum_{w \in N_0(v)} y_w^* \end{aligned} \tag{3.5}$$

Hence, for every free vertex, there is a pair of equations (one for x, and one for y). These equations are linear, and after labeling the free vertices $v_1, ..., v_n$, we may rewrite them as the matrix multiplications described in (5) and (6) below. Notice the M is an $n \times n$ diagonally dominant matrix. The first fact can be observed by inspecting 3.5, and the second occurs because the diagonal consists of vertex degrees, while the other entries M_{ij} are either -1's (if x_i and x_j are neighbors) or 0's if they aren't. Formally, this matrix M is known as the **Laplacian** of the graph.

$$M_{ij} = \begin{cases} \deg(v) & \text{if } i = j \\ -1 & \text{if adjacent} \\ 0 & \text{otherwise} \end{cases} \equiv \begin{bmatrix} \deg(v) & & & * \\ & \ddots & & \\ & & \ddots & \\ * & & & \deg(v) \end{bmatrix} \tag{3.6}$$

We may find the x coordinates of the free vertices by solving

$$M \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{w \in N_0(x_1)} x_w^* \\ \vdots \\ \sum_{w \in N_0(x_n)} x_w^* \end{pmatrix} \tag{3.7}$$

and the y coordinates by solving

$$M \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \sum_{w \in N_0(y_1)} y_w^* \\ \vdots \\ \sum_{w \in N_0(y_n)} y_w^* \end{pmatrix} \tag{3.8}$$

3.3.3 Example: Hypercube

A simple example for which Tutte's method gives aesthetically pleasing results is the hypercube.

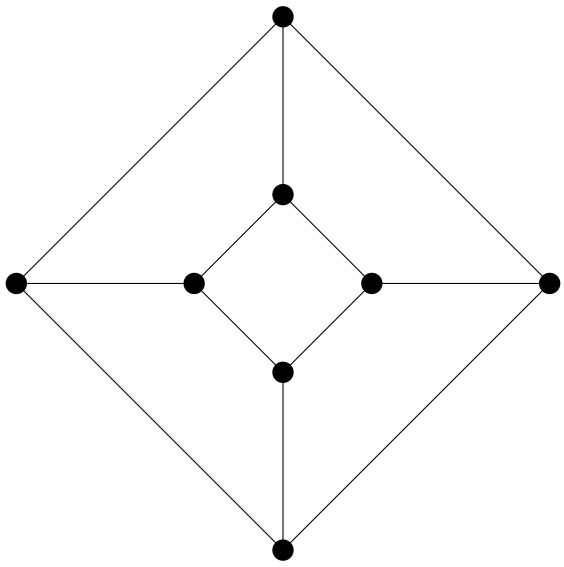


Figure 3.3: The hypercube Q_3

In Figure 3.3 the hypercube is placed in 500 x 500 pixel grid. The grid is governed by a simple Cartesian coordinate system, where the top left and bottom right corners have coordinates $(-250, 0)$ and $(250, 250)$ respectively. Four vertices are fixed and laid out into a circle of radius 250 centered at the origin. Hence, the bulk of the work performed algorithm is done in placing the center four free vertices. Labeling the free vertices as x_1, x_2, x_3, x_4, we may represent the task of laying out the free vertices with this matrix

$$\begin{bmatrix} 3 & -1 & 0 & -1 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ -1 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 250 \\ 0 \\ -250 \end{bmatrix}$$

The solution to this matrix is given by $x_1 = x_3 = 0, x_2 = \frac{250}{3}, x_4 = -\frac{250}{3}$.

3.3.4 Algorithms

Newton-Raphson Iteration

The below Newton-Raphson Iteration is easy to code and is reasonably fast. However, it is not as fast as solving Algorithm 6 below with a linear algebra package. The idea behind the algorithm is that we simply keep iterating until the x, y values of each vertex converges.

Algorithm 5 Barycenter Layout (Newton-Raphson)

procedure BARYCENTER(t)
  Place each fixed vertex $u \in V_0$ at a vertex of P and each free vertex at the origin.
  $converge \leftarrow false$
  while !$converge$ **do**
    $converge \leftarrow true$
    for each free vertex v **do**

$$x_v = \frac{1}{\deg v} \sum_{(u,v)\in E} x_u$$

$$y_v = \frac{1}{\deg v} \sum_{(u,v)\in E} y_u$$

      // If this does not execute at any point in the for loop, then while loop exists
      if p did not converge **then**
        $converge \leftarrow false$
      end if
    end for
  end while
end procedure

Linear System

Of course, with a computer linear algebra package (such as Eigen for C++), [3] one can also solve the corresponding linear system directly. In practice, this tends to be significantly faster than the previous algorithm.

Algorithm 6 Barycenter Layout (Linear Algebra)

procedure BARYCENTER(t)
  Layout n fixed vertices in a convex polygon
  Construct a matrix M as described by (3.6)
  Construct a vector of x-coordinates for free vertices and another for fixed vertices. Along with M, use these to solve (3.7)
  Construct a vector of y-coordinates for free vertices and another for fixed vertices. Along with M, use these to solve (3.8)
end procedure

3.3.5 Case Study: Prism Graph

The prism graph Π_n is a 3-connected graph constructed by taking the vertices and edges of an n-prism. From the perspective of this drawing algorithm, it allows us to investigate the symmetry and resolution properties. In this paper, and the proofs below, Π_n will be drawn by using n fixed vertices (equally spaced on the perimeter of a circle).

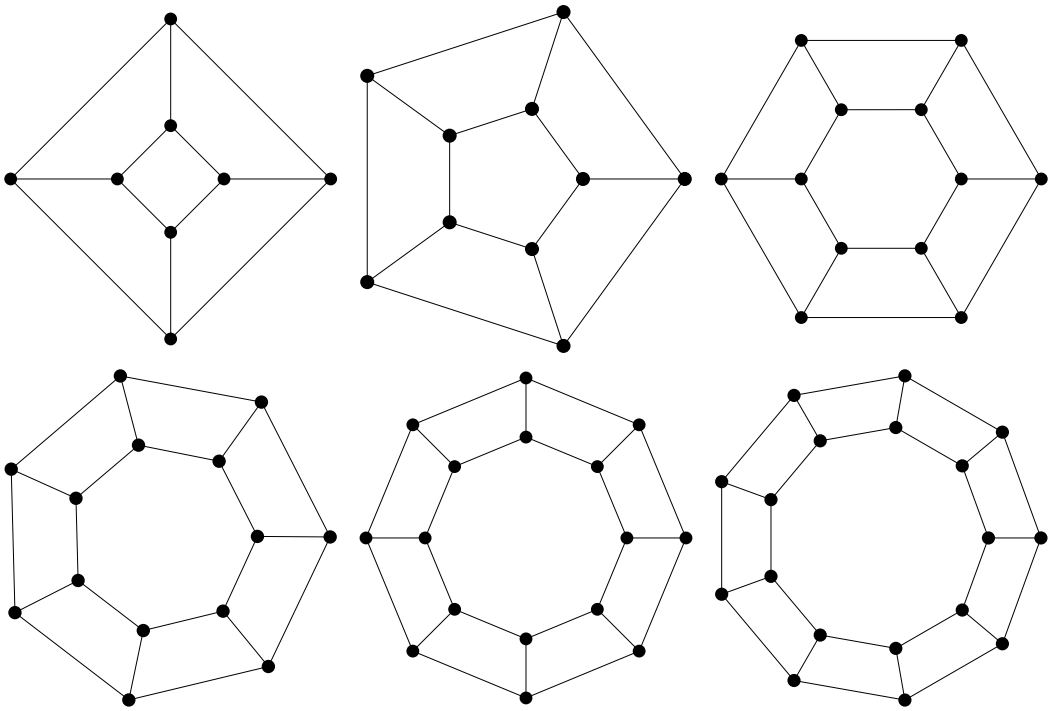


Figure 3.4: Π_4 through Π_9 as drawn by Tutte's algorithm. Notice that Π_4 is isomorphic to the hypercube Q_3

Symmetry

Under certain conditions, the barycenter method produces drawings which preserve symmetry.

Theorem 3.3.1 (Eigenvectors of the Prism Graph)**.** *Consider the linear system governing the coordinates of the free vertices of the prism graph Π_n. Now, take its corresponding matrix M and starting at $(1,0)$, place n points equally along the perimeter of the unit circle. If we create vectors $x = (x_1, ..., x_n), y = (y_1, ..., y_n)$, where x_i is the x-coordinate of the i^{th} unit circle point (and similarly for y), then x, y are eigenvectors for M with corresponding eigenvalues $\lambda_x = \lambda_y = 3 - 2\cos\frac{2\pi}{n}$.*

Proof. First, let us prove that this is true for x. Notice by the distributivity of linear maps that $Mx = (3I + N)x = 3Ix + Nx$, where N is a matrix composed of all of the $-1's$ in M (and is zero everywhere else). Hence, $N\vec{x}$ is of the form

$$\begin{bmatrix} 0 & -1 & 0 & \dots & 0 & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & 0 & \dots & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \cos 0 \\ \cos\frac{2\pi}{n} \\ \vdots \\ \cos\frac{2\pi(n-2)}{n} \\ \cos\frac{2\pi(n-1)}{n} \end{bmatrix}$$

Clearly, any vector is an eigenvector of the identity map, so we just have to show that x is an eigenvector of N. By the above matrix, showing that $Nx = \lambda_0 x$ is equivalent to showing that the following holds for some $\lambda_0 \in \mathbb{R}$.

$$\begin{cases} -\cos\frac{2\pi}{n} - \cos\frac{2\pi(n-1)}{n} = \lambda_0 \cos 0 & \text{Equation for the first row} \\ -\cos\frac{2\pi(i-2)}{n} - \cos\frac{2\pi i}{n} = \lambda_0 \cos\frac{2\pi(i-1)}{n} & \text{Equation for the } i^{th} \text{ row} \end{cases}$$

Now, the first equation implies that

$$\begin{aligned} \lambda_0 &= -\left[\cos\frac{2\pi}{n} + \cos\frac{2\pi n - 2\pi}{n}\right] \\ &= -2\left[\cos\frac{2\pi + 2\pi n - 2\pi}{2n}\cos\frac{2\pi - 2\pi n + 2\pi}{2n}\right] && \text{Using sum-product identity} \\ &= -2\left[\cos\frac{2\pi n}{2n}\cos\frac{4\pi - 2\pi n}{2n}\right] \\ &= -2\left[\cos\pi\cos\frac{2\pi}{n} - \pi\right] \\ &= 2\left[\cos\pi - \frac{2\pi}{n}\right] && \cos \text{ is an even function} \\ &= -2\cos -\frac{2\pi}{n} = -2\cos\frac{2\pi}{n} && \text{Supplementary angles} \end{aligned}$$

implying that $\lambda_x = 3 - 2\cos\frac{2\pi}{n}$ as desired. Now, we just need to show the equation for the i^{th} row holds. Notice that

$$\begin{aligned} &-\left[\cos\frac{2\pi(i-2)}{n} + \cos\frac{2\pi i}{n}\right] \\ &= -2\left[\cos\frac{2\pi(i-2) + 2\pi i}{2n}\cos\frac{2\pi(i-2) - 2\pi i}{2n}\right] && \text{Sum-product identity} \\ &= -2\left[\cos\frac{\pi i - 2\pi + \pi i}{n}\cos\frac{\pi i - 2\pi - \pi i}{n}\right] \\ &= -2\left[\cos\frac{\pi i - 2\pi + \pi i}{n}\cos\frac{2\pi}{n}\right] && \cos\frac{-2\pi}{n} = \cos\frac{2\pi}{n} \\ &= \lambda_0 \cos\frac{2\pi(i-1)}{n} \end{aligned}$$

as desired.

Now, let us prove that this is true for y. The proof is very similar to the proof for x.

Here, $N\vec{y}$ is of the form

$$\begin{bmatrix} 0 & -1 & 0 & \dots & 0 & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & 0 & \dots & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \sin 0 \\ \sin \frac{2\pi}{n} \\ \vdots \\ \sin \frac{2\pi(n-2)}{n} \\ \sin \frac{2\pi(n-1)}{n} \end{bmatrix}$$

Equivalently, we want to show that the following holds for some $\lambda_1 \in \mathbb{R}$.

$$\begin{cases} -\sin \frac{2\pi}{n} - \sin \frac{2\pi(n-1)}{n} = \lambda_1 \sin 0 & \text{Equation for the first row} \\ -\sin 0 - \sin \frac{2\pi \cdot 2}{n} = \lambda_1 \sin \frac{2\pi}{n} & \text{Equation for the second row} \\ -\sin \frac{2\pi(i-2)}{n} - \sin \frac{2\pi i}{n} = \lambda_1 \sin \frac{2\pi(i-1)}{n} & \text{Equation for the } i^{th} \text{ row} \end{cases}$$

Note that here, we'll fix λ_1 by using the equation for the 2nd row since in the first row, the right hand side is equal to 0. Now, the second equation implies that

$$\begin{aligned} \lambda_1 \sin \frac{2\pi}{n} &= -\left(\sin 0 + \sin \frac{4\pi n}{n}\right) \\ &= -\sin \frac{4\pi}{n} \\ &= -2 \sin \frac{2\pi}{n} \cos \frac{2\pi}{n} && \text{Double angle identity} \end{aligned}$$

Simplifying we get $\lambda_1 = -2\cos \frac{2\pi}{n}$, implying $\lambda_y = 3I - 2\cos \frac{2\pi}{n}$ as desired. Now, we just need to show that the equation for the i^{th} row holds. Notice that

$$\begin{aligned} & -\left[\sin \frac{2\pi(i-2)}{n} + \sin \frac{2\pi i}{n}\right] \\ &= -2\left[\sin \frac{2\pi(i-2) + 2\pi i}{2n} \cos \frac{2\pi(i-2) - 2\pi i}{2n}\right] && \text{Sum-product identity} \\ &= -2\left[\sin \frac{\pi i - 2\pi + \pi i}{n} \cos \frac{\pi i - 2\pi - \pi i}{n}\right] \\ &= -2\left[\sin \frac{2\pi(i-1)}{n} \cos \frac{2\pi}{n}\right] && \cos \frac{-2\pi}{n} = \cos \frac{2\pi}{n} \\ &= \lambda_1 \sin \frac{2\pi(i-1)}{n} \end{aligned}$$

as desired. Hence, we have shown that our vectors x, y of points along the unit circle are eigenvectors for M indeed. □

Corollary 3.3.1 (Reflectional Symmetry)**.** *The barycenter method gives a reflectionally symmetric drawing of the prism graph.*

Proof. From the theorem above, because the equally spaced points of a circle form an eigenvector of the linear system for the prism graph, each free vertex is a scalar multiple of some fixed vertex lying on said circle. Hence, the axes of symmetry lie on a line between each fixed vertex and its associated free vertex. Because the edges of connecting each fixed vertex to its associated free vertex also lie on these axes of symmetry, the barycenter method gives a symmetric drawing as required. □

Resolution

One the the main drawbacks of this algorithm is potentially poor resolution, i.e. the more edges and vertices we add to our graph, the harder it becomes to distinguish the different features of our graph. This is demonstrated best by the prism graph.

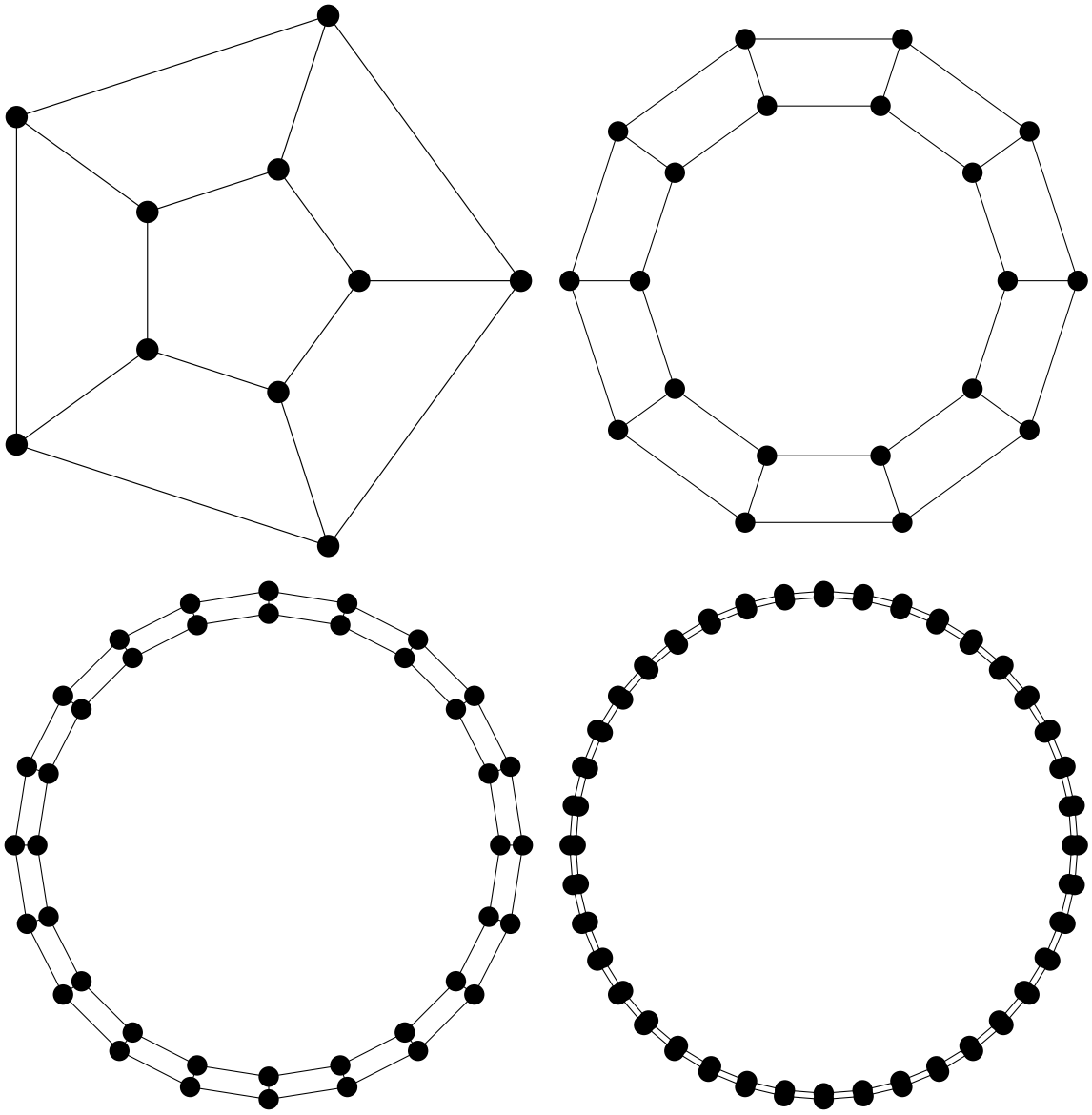


Figure 3.5: $\Pi_5, \Pi_{10}, \Pi_{20}$ and Π_{40} as drawn by Tutte's algorithm

Theorem 3.3.2 (Poor Resolution of the Prism Graph)**.** *For every fixed vertex u in the prism graph Π_n, the distance between it and its adjacent free vertex v tends to 0 as n becomes large.*

Proof. Without loss of generality, assume that this graph is drawn in the unit square. From the theorem above, we know that

$$v = u \cdot \frac{1}{3 - 2\cos\frac{2\pi}{n}}$$

Hence,

$$\begin{aligned}
\operatorname{dist}(u,v) &= \sqrt{\left(u_x - u_x \cdot \frac{1}{3-2\cos\frac{2\pi}{n}}\right)^2 + \left(u_y - u_y \cdot \frac{1}{3-2\cos\frac{2\pi}{n}}\right)^2} \\
&= \sqrt{\left[u_x\left(1 - \frac{1}{3-2\cos\frac{2\pi}{n}}\right)\right]^2 + \left[u_y\left(1 - \frac{1}{3-2\cos\frac{2\pi}{n}}\right)\right]^2 +} \\
&= \sqrt{u_x^2\left(1 - \frac{1}{3-2\cos\frac{2\pi}{n}}\right)^2 + u_y^2\left(1 - \frac{1}{3-2\cos\frac{2\pi}{n}}\right)^2}
\end{aligned}$$

Using the fact that u is a point on the unit circle,

$$\begin{aligned}
\operatorname{dist}(u,v) &= \sqrt{\left(u_x^2 + u_y^2\right)\left(1 - \frac{1}{3-2\cos\frac{2\pi}{n}}\right)^2} \\
&= \sqrt{\left(u_x^2 + u_y^2\right)} \cdot \sqrt{\left(1 - \frac{1}{3-2\cos\frac{2\pi}{n}}\right)^2} \\
&= 1 - \frac{1}{3-2\cos\frac{2\pi}{n}}
\end{aligned}$$

If we take the limit as n goes to infinity, we get

$$\operatorname{dist}(u,v) = 1 - \frac{1}{3-2\cos 0} = 1 - \frac{1}{3-2} = 0$$

□

Chapter 4

Appendix

4.1 Eades' Method Gallery

4.1.1 Complete Graphs

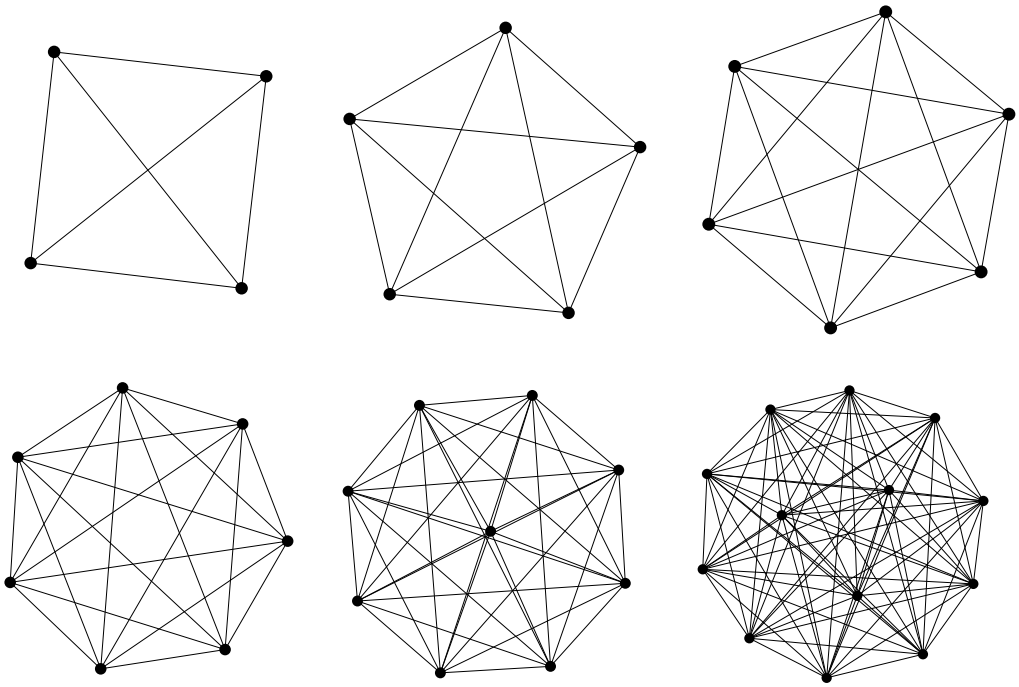
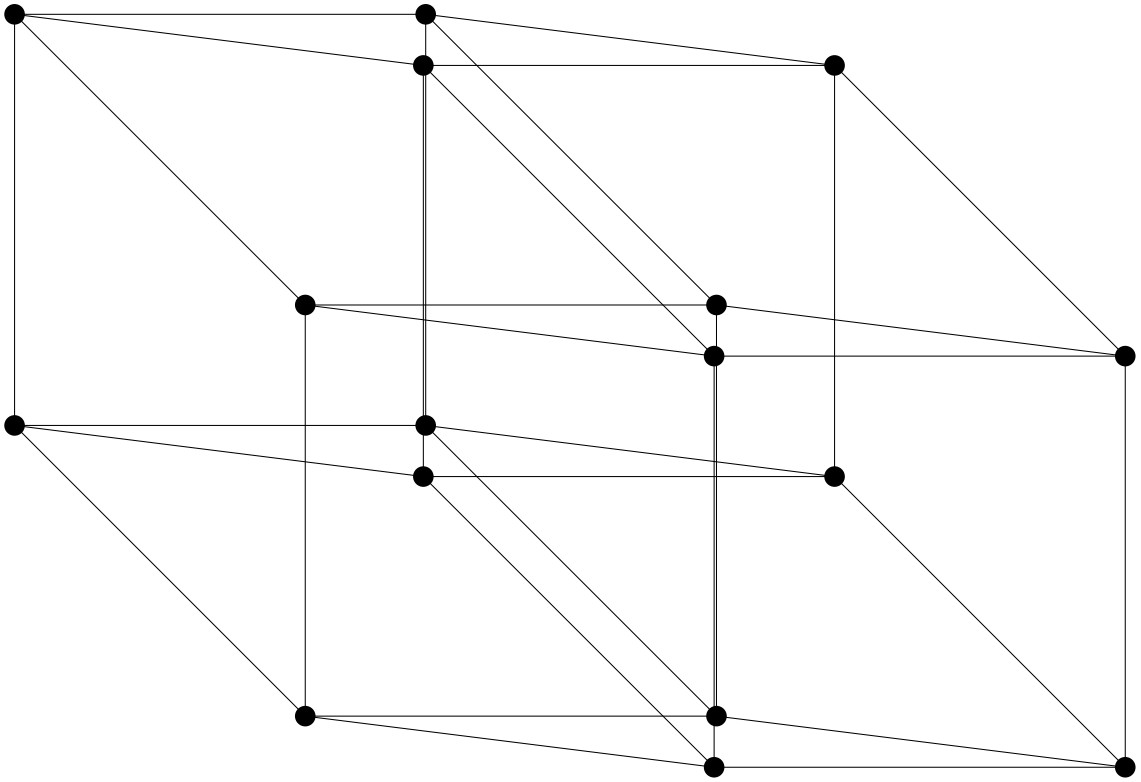


Figure 4.1: The graphs K_4, K_5, K_6, K_7, K_9 and K_{13} as drawn by Eades' spring algorithm

4.1.2 Tesseract Q_4



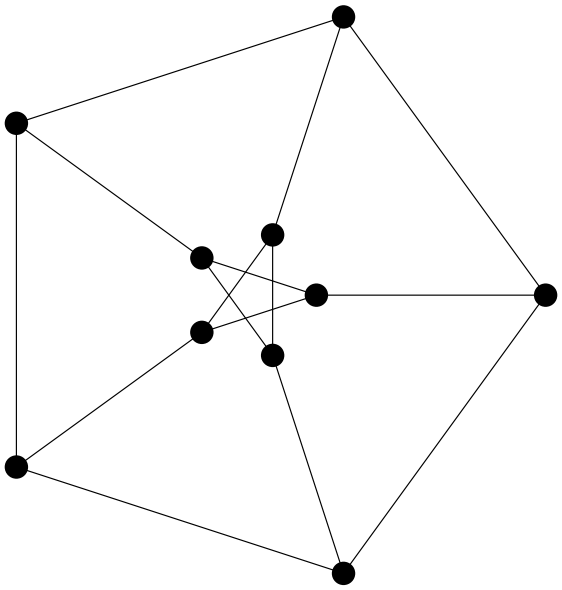
4.2 Barycenter Method: Generalized Petersen Graphs

One can extend the notion of a Petersen graph to a family of graphs formed by taking the vertices of a $n-$cycle and connecting to them to the corresponding vertices of a star polygon. Tutte's barycenter method described above draws this family of graphs very well. Watkins [9] gives a simple description of a Generalized Petersen graph $GP(n,k)$ as a graph with

$$V(GP(n,k)) = \{u_0, u_1, ..., u_{n-1}, v_0, v_1, ..., v_{n-1}\}$$
$$E(GP(n,k)) = \{[u_i, u_{i+1}], [u_i, v_i], [v_i, v_{i+k}]\}$$

for $i = 0, ..., n-1$ and $k \leq n-1, 2k \neq n$. Note that the subscripts for the edge set should be read modulo n. Using this definition, the Petersen graph can be described as $GP(5,2)$.

4.2.1 Petersen Graph (GP(5, 2))



In this image above, the x-coordinates are governed by

$$\begin{bmatrix} 3 & 0 & -1 & -1 & 0 \\ 0 & 3 & 0 & -1 & -1 \\ -1 & 0 & 3 & 0 & -1 \\ -1 & -1 & 0 & 3 & 0 \\ 0 & -1 & -1 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 250 \\ 77.25 \\ -202.25 \\ -202.25 \\ 77.25 \end{bmatrix}$$

and the y-coordinates are governed by

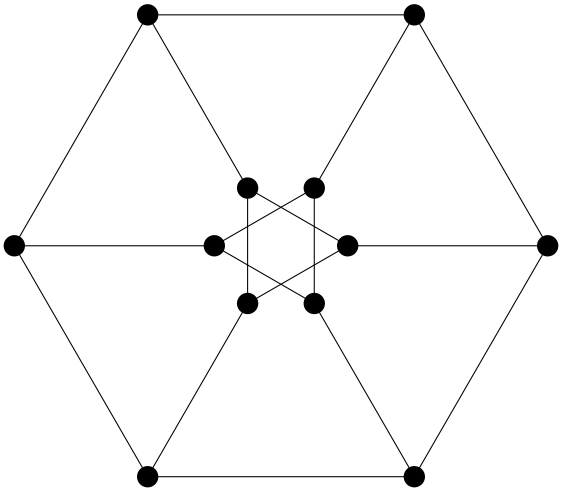
$$\begin{bmatrix} 3 & 0 & -1 & -1 & 0 \\ 0 & 3 & 0 & -1 & -1 \\ -1 & 0 & 3 & 0 & -1 \\ -1 & -1 & 0 & 3 & 0 \\ 0 & -1 & -1 & 0 & 3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 237.76 \\ 146.95 \\ -146.95 \\ -237.76 \end{bmatrix}$$

with

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 54.14 \\ 16.73 \\ -43.8 \\ -43.8 \\ 16.73 \end{bmatrix}, \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} -0 \\ 51.49 \\ 31.82 \\ -31.82 \\ -51.49 \end{bmatrix}$$

Watkins showed that the Petersen graph is the only member of the Generalized Petersen family of graphs to not have a 3-edge-coloring. [9]

4.2.2 Dürer Graph (GP(6, 2))



This graph has Laplacian

$$M = \begin{bmatrix} 3 & 0 & -1 & 0 & -1 & 0 \\ 0 & 3 & 0 & -1 & 0 & -1 \\ -1 & 0 & 3 & 0 & -1 & 0 \\ 0 & -1 & 0 & 3 & 0 & -1 \\ -1 & 0 & -1 & 0 & 3 & 0 \\ 0 & -1 & 0 & -1 & 0 & 3 \end{bmatrix}$$

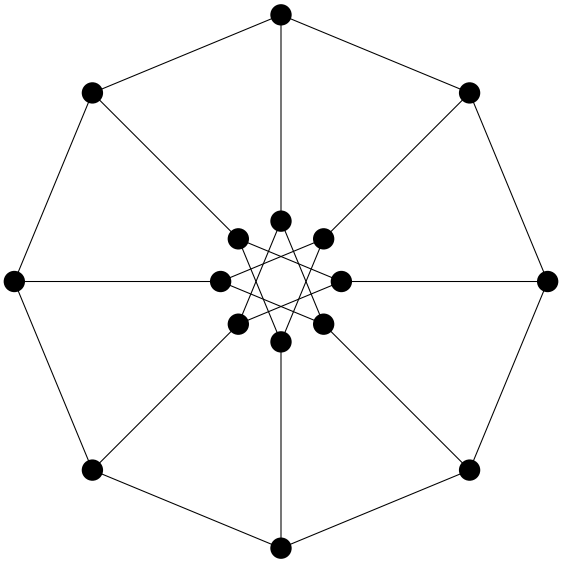
The coordinates of the free vertices are governed by

$$M \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 250 \\ 125 \\ -125 \\ -250 \\ -125 \\ 125 \end{bmatrix}, M \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 217 \\ 217 \\ 0 \\ -217 \\ -217 \end{bmatrix}$$

with solutions

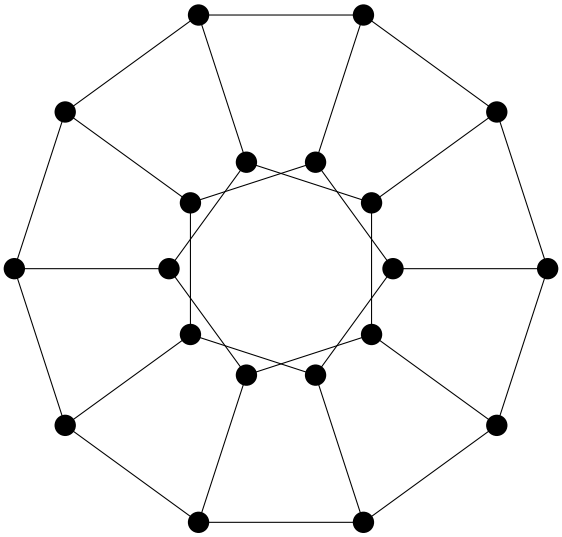
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 62.5 \\ 31.2 \\ -31.3 \\ -62.5 \\ -31.3 \\ 31.2 \end{bmatrix}, \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 54.1 \\ 54.1 \\ 0 \\ -54.1 \\ -54.1 \end{bmatrix}$$

4.2.3 Möbius-Kantor Graph (GP(8, 3))

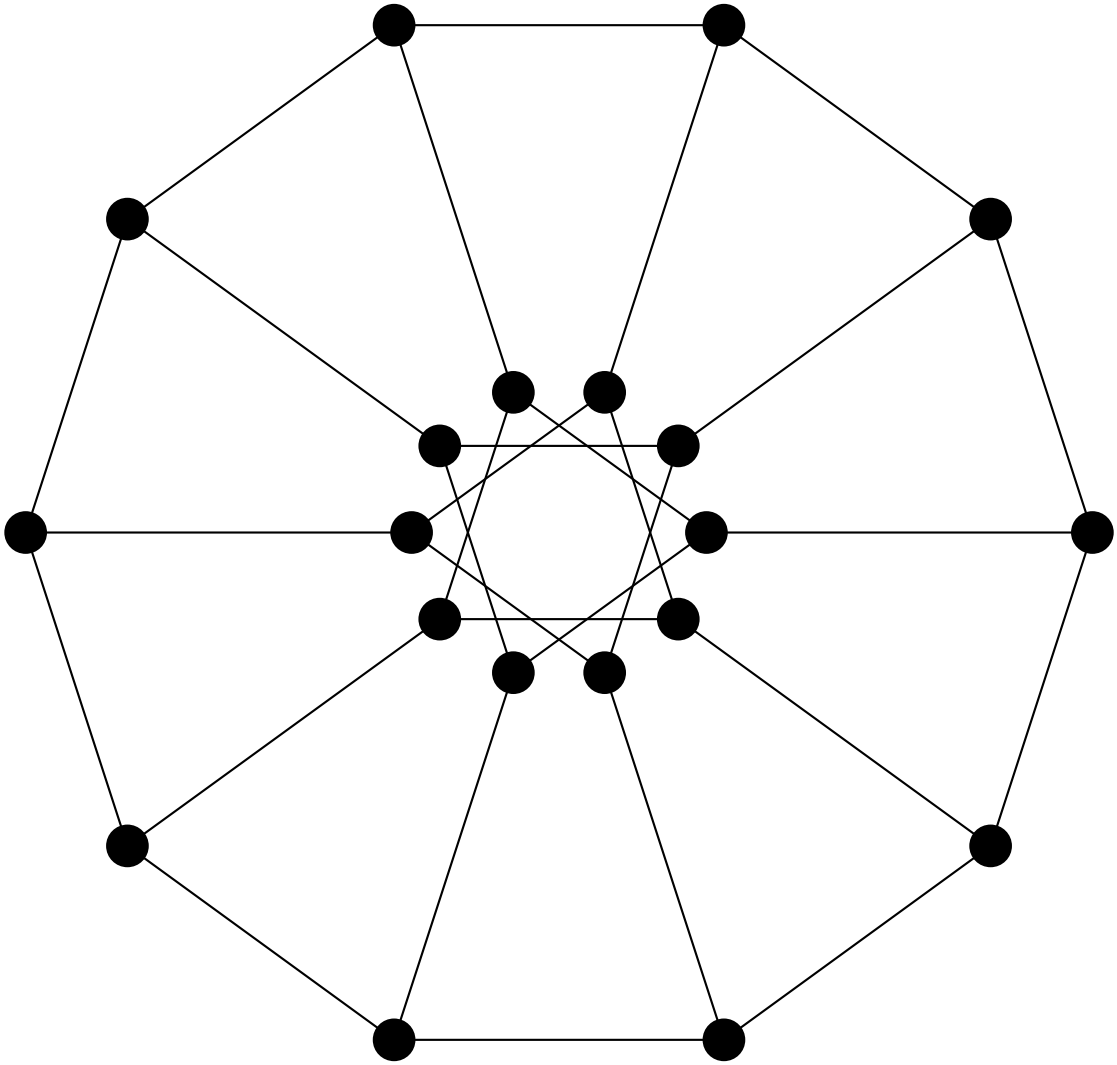


4.2.4 Dodecahedral Graph (GP(10, 2))

This graph can be thought of the skeleton of the dodecahedron.

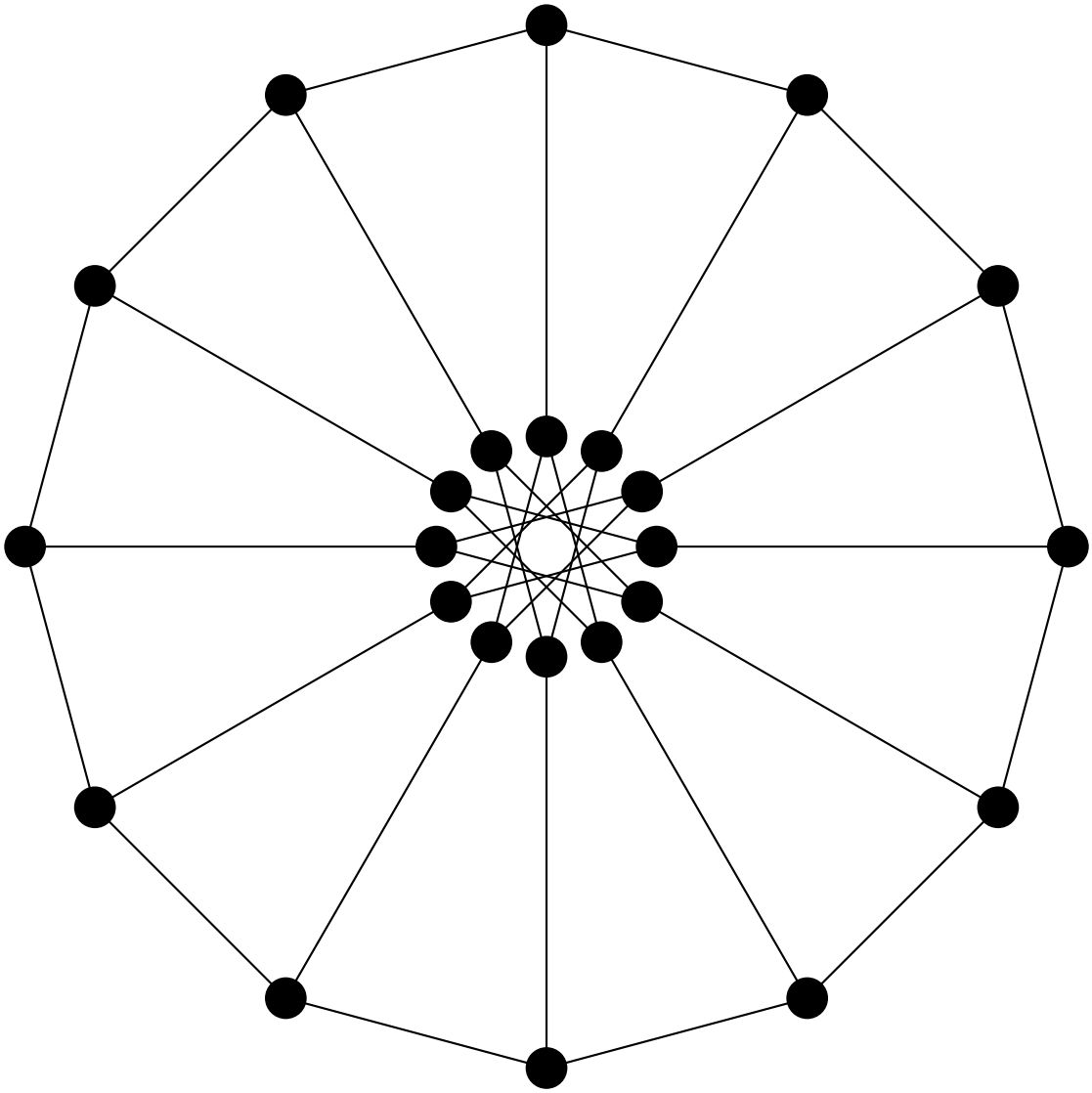


4.2.5 Desargues Graph (GP(10, 3))

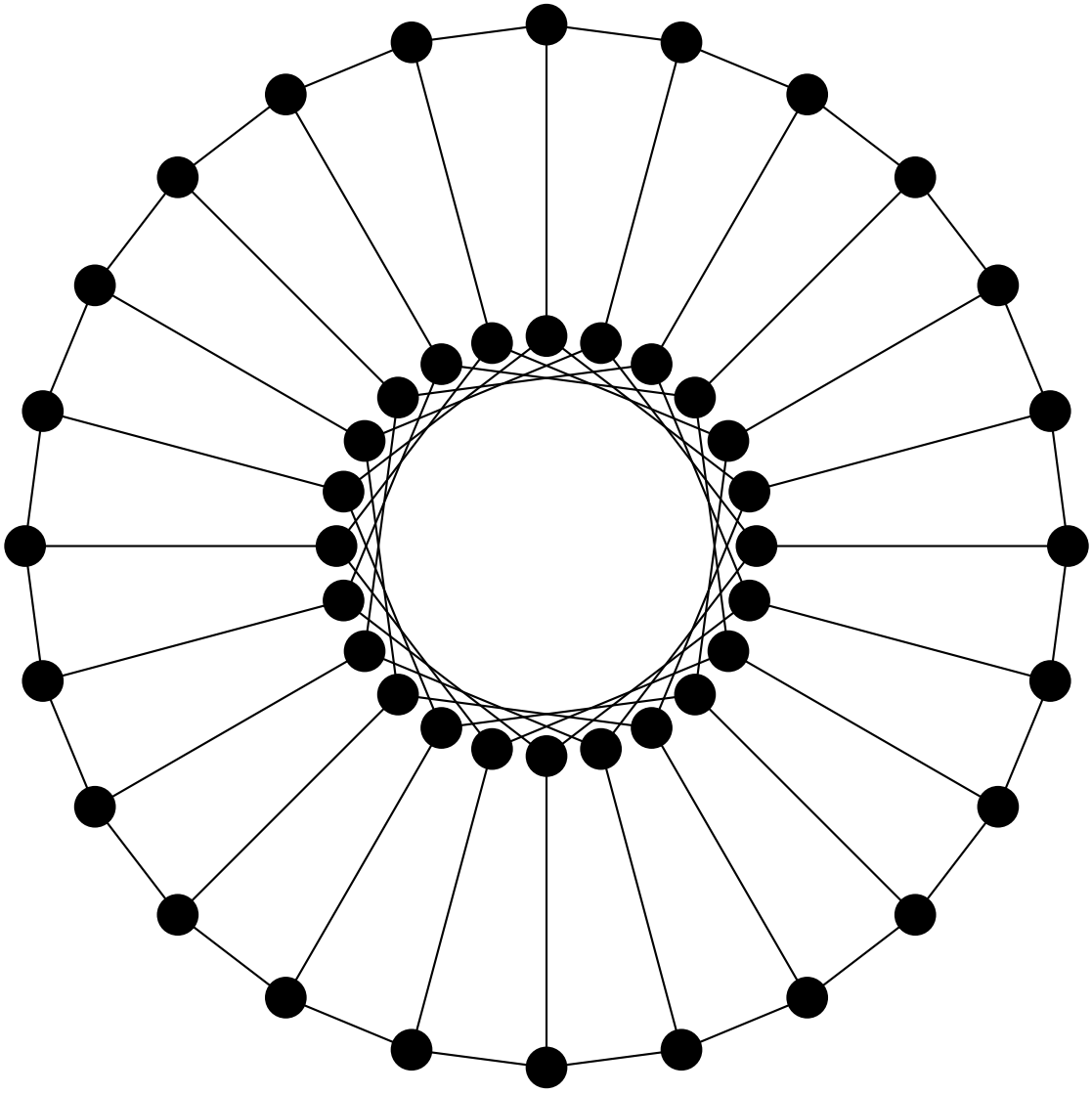


4.2.6 Nauru Graph (GP(12, 5))

The inner star polygon of this graph resembles the star on the flag of the Pacific island of Nauru, hence the name. [10]



4.2.7 Cubic Symmetric Graph $F_{048}A$ (GP(25, 4))



Bibliography

[1] Giuseppe Di Battista et al. *Graph Drawing: Algorithms for the Visualization of Graphs*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN: 0133016153.

[2] Peter Eades. "A heuristic for graph drawing". In: *Congressus numerantium* 42 (1984), pp. 149–160.

[3] Gael Guennebaud, Benoit Jacob, et al. *Eigen: a C++ linear algebra library*. Version 3.3.4. 2017-06-17. URL: `http://eigen.tuxfamily.org/`.

[4] Gary D. Knott. "A Numbering System for Binary Trees". In: *Commun. ACM* 20.2 (Feb. 1977), pp. 113–115. ISSN: 0001-0782. DOI: `10.1145/359423.359434`. URL: `http://doi.acm.org/10.1145/359423.359434`.

[5] Andrew Makhorin. *GNU Linear Programming Kit*. Version 4.64. Feb. 16, 2018. URL: `https://www.gnu.org/software/glpk/`.

[6] Edward M. Reingold and John S. Tilford. "Tidier drawings of trees". In: *IEEE Transactions on software Engineering* 2 (1981), pp. 223–228.

[7] Kenneth J Supowit and Edward M Reingold. "The complexity of drawing trees nicely". In: *Acta Informatica* 18.4 (1983), pp. 377–392.

[8] William Thomas Tutte. "How to draw a graph". In: *Proceedings of the London Mathematical Society* 3.1 (1963), pp. 743–767.

[9] Mark E Watkins. "A theorem on Tait colorings with an application to the generalized Petersen graphs". In: *Journal of Combinatorial Theory* 6.2 (1969), pp. 152–164.

[10] Eric W. Weisstein. *Nauru Graph*. URL: `http://mathworld.wolfram.com/NauruGraph.html` (visited on 06/15/2018).