

# Graph Drawing Algorithms

Vincent La

May 20, 2018

# Contents

<b>1</b>	<b>Tree Drawing Algorithms</b>	<b>2</b>
1.1	Motivation . . . . .	2
1.2	Tree Drawing as a Linear Program – Supowit and Reingold (1981) . . . . .	2
1.2.1	The Problem of Narrow Trees . . . . .	2
1.2.2	Aesthetic Trees . . . . .	2
1.2.3	Formal Description . . . . .	3
1.2.4	Implementation . . . . .	5
<b>2</b>	<b>Force Directed Algorithms</b>	<b>8</b>
2.1	Introduction . . . . .	8
2.1.1	Notation . . . . .	8
2.2	Eades’ Spring System . . . . .	9
2.3	Tutte’s Barycenter Method . . . . .	10
2.3.1	Fixed Vertices . . . . .	10
2.3.2	Linear Model . . . . .	10
2.3.3	Example: Hypercube . . . . .	11
2.3.4	Algorithms . . . . .	12
2.3.5	Case Study: Prism Graph . . . . .	13
2.4	Appendix . . . . .	19
2.4.1	Barycenter Method: Petersen Graph . . . . .	19

# Chapter 1

## Tree Drawing Algorithms

### 1.1 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.

### 1.2 Tree Drawing as a Linear Program – Supowit and Reingold (1981)

#### 1.2.1 The Problem of Narrow Trees

Unfortunately, as shown earlier (note to self: show earlier), locally minimizing the width of every subtree does not achieve the goal of a minimum width drawing. Sometimes in order to draw a tree as narrow as possible, some subtrees have to be drawn non-optimally.

#### 1.2.2 Aesthetic Trees

In the algorithm below, the problem of drawing a tree is formulated as a linear program. The problem is 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 (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 share a common vertex
6. **Isomorphic Trees Drawn Similarly** If two subtrees are isomorphic, they must be drawn identically (minus a translation)

It should be noted that aesthetic 1, 2 and 3 imply 5 (see Theorem 1.2.1).

### 1.2.3 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:

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

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

$$f_x(n) - f_x(\text{left child}(n)) \geq 1$$

For all  $n$  with a left child (Aesthetic 2)

$$f_x(\text{right child}(n)) - f_x(n) \geq 1$$

For all  $n$  with a right child (Aesthetic 2)

$$|f_x(n) - f_x(m)| \geq 2$$

For all  $m, n$  at level  $i$  (Aesthetic 3)

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

The sixth aesthetic is very difficult to implement, so I will leave it for a later report.

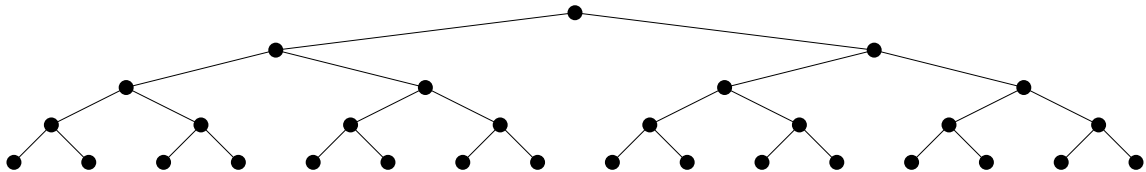


Figure 1.1: A drawing of a perfect tree of height 4

**Theorem 1.2.1** (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 1.2: 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 intersection 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. However, this violates aesthetic 3. It may also be since some node's left subtree crosses some other node's left subtree. However, this violates aesthetic 2. Lastly, it may be that some node's right subtree crosses some other node's right subtree. Again, this violates aesthetic 2.  $\square$

**Theorem 1.2.2** (Number of Constraints). *The number of constraints is  $O(n)$ .*

*Proof.* Let  $T$  be a binary tree with  $n$  nodes. First, the requirements  $x_{min} \leq f_x(n) \leq x_{max}$  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 are at most  $\frac{n}{2}$  equivalence classes 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.  $\square$

### 1.2.4 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. Each node was represented as a data structure as follows:

<code>left</code>	Pointer to left child
<code>right</code>	Pointer to right child
<code>id</code>	Corresponding column index of constraint matrix

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. During the level-order traversal, a hash table mapping each level of the tree to an array of node pointers was populated (`std::unordered_map<int, TreeNode*>`). Then, a final iteration over this auxiliary data structure was used to calculate the aesthetics. (Write about how paper uses a different method).

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 indices 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 the iterative traversal where children are added to a queue, the current level + 1 was also paired with the child pointers. (Find a better way to word this) The following pseudocode is loosely based on C++ syntax.

---

**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:      $current\_level \leftarrow current.first$  ▷  $current = (\text{level (integer)}, \text{node pointer})$ 
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:       $children \leftarrow (current\_level + 1, root \rightarrow left)$ 
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:       $children \leftarrow (current\_level + 1, root \rightarrow right)$ 
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  $l = 0, \dots, h$  do
3:     for node  $i = 0, \dots, m$  do  $\triangleright$  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  $current\_node \leq x_{max}$   $\triangleright$  Add width constraints
6:       Add constraint  $current\_node \geq x_{min}$ 
7:       if  $current\_node \rightarrow left$  is not null then
8:         Add second constraint
9:       end if
10:      if  $current\_node \rightarrow right$  is not null then
11:        Add second constraint
12:      end if
13:      if  $current\_node \rightarrow left$  and  $current\_node \rightarrow right$  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.



## Chapter 2

# Force Directed Algorithms

### 2.1 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 it's "center of mass."

#### 2.1.1 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, we saw that each vertex  $v$  in a graph gets mapped to some point  $p_v = (x_v, y_v)$ .

## 2.2 Eades' Spring System

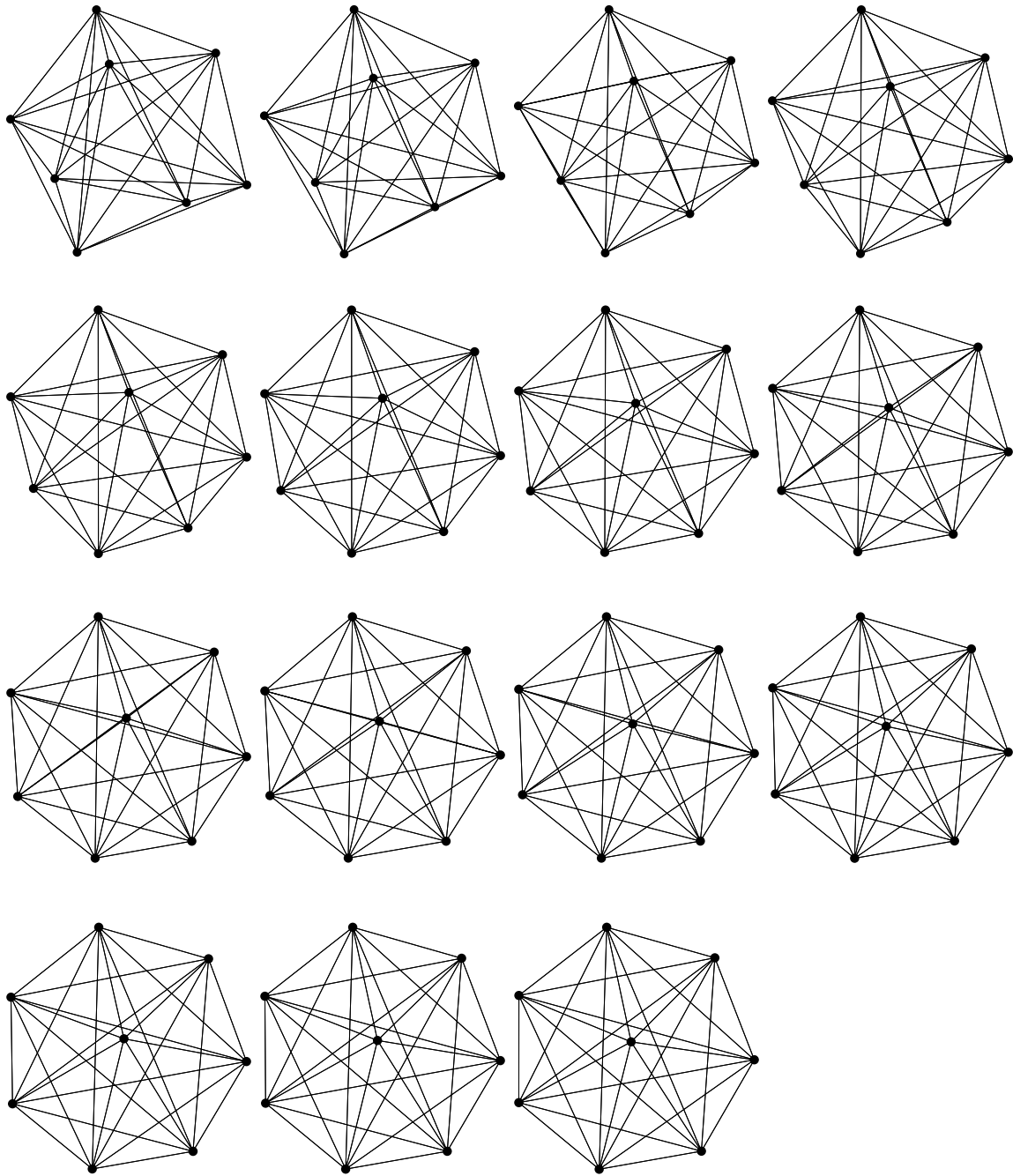


Figure 2.1: The complete graph  $K_8$

## 2.3 Tutte's Barycenter Method

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

$$F(v) = \sum_{(u,v) \in E} (p_u - p_v) \quad (2.1)$$

Hence, splitting (2.1) across the x and y dimensions we get

$$\begin{aligned} \sum_{(u,v) \in E} (x_u - x_v) &= 0 \\ \sum_{(u,v) \in E} (y_u - y_v) &= 0 \end{aligned} \quad (2.2)$$

### 2.3.1 Fixed Vertices

However, notice the system in (2.2) 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.

### 2.3.2 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

$$\begin{aligned} \deg(v)x_v - \sum_{u \in N_1(v)} x_u &= \sum_{w \in N_0(v)} x_w^* \\ \deg(v)y_v - \sum_{u \in N_1(v)} y_u &= \sum_{w \in N_0(v)} y_w^* \end{aligned} \quad (2.3)$$

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  $v_1, \dots, v_n$ , 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 2.3, 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.

$$M_{ij} = \begin{cases} \deg(v) & \text{if } i = j \\ -1 & \text{if adjacent} \\ 0 & \text{otherwise} \end{cases} \equiv \begin{bmatrix} \deg(v) & & & * \\ & \ddots & & \\ & & \ddots & \\ * & & & \deg(v) \end{bmatrix} \quad (2.4)$$

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

$$M \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{w \in N_0(x_1)} x_w^* \\ \vdots \\ \sum_{w \in N_0(x_n)} x_w^* \end{pmatrix} \quad (2.5)$$

and the y coordinates by solving

$$M \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \sum_{w \in N_0(y_1)} y_w^* \\ \vdots \\ \sum_{w \in N_0(y_n)} y_w^* \end{pmatrix} \quad (2.6)$$

### 2.3.3 Example: Hypercube

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

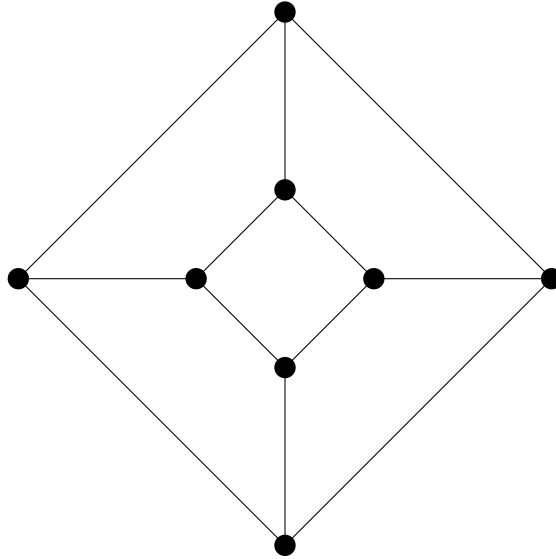


Figure 2.2: The hypercube  $Q_3$

In Figure ?? 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  $x_1 = x_3 = 0, x_2 = \frac{250}{3}, x_4 = -\frac{250}{3}$ .

### 2.3.4 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 4 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. However, this implies we need to define convergence in a way dumb enough for a computer to understand.

**Definition 2.3.1** (Convergence). *Let  $p_{i-1}, p$  be the placement of some vertex  $v$  during the  $i - 1$  and  $i^{th}$  iteration of the Newton-Raphson Iteration. We say that  $p$  has converged if  $|p - p_{i-1}| < \epsilon$ . (In practice, we can define  $\epsilon$  to be a very small positive number like 0.01).*

---

#### Algorithm 3 Barycenter Layout (Newton-Raphson)

---

```

1: procedure BARYCENTER( $t$ )
2:   Place each fixed vertex  $u \in V_0$  at a vertex of  $P$  and each free vertex at the origin.
3:    $converge \leftarrow false$ 
4:   while  $\neg converge$  do
5:      $converge \leftarrow true$ 
6:     for each free vertex  $v$  do
```

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

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

```

7:       // If this does not execute at any point in the for loop, then while loop exists
8:       if  $p$  did not converge then
9:          $converge \leftarrow false$ 
10:      end if
11:    end for
12:  end while
13: end procedure
```

---

#### Linear System

Of course, with a computer linear algebra package, one can also solve the corresponding linear system directly. In practice, this tends to be significantly faster than the previous

algorithm.

---

**Algorithm 4** Barycenter Layout (Linear Algebra)

---

- 1: **procedure** BARYCENTER( $t$ )
  - 2:   Layout  $n$  fixed vertices in a convex polygon
  - 3:   Construct a matrix  $M$  as described by (2.4)
  - 4:   Construct a vector of  $x$ -coordinates for free vertices and another for fixed vertices.  
       Along with  $M$ , use these to solve (2.5)
  - 5:   Construct a vector of  $y$ -coordinates for free vertices and another for fixed vertices.  
       Along with  $M$ , use these to solve (2.6)
  - 6: **end procedure**
- 

### 2.3.5 Case Study: Prism Graph

The prism graph  $\Pi_n$  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,  $\Pi_n$  will be drawn by using  $n$  fixed vertices (equally spaced on the perimeter of a circle).

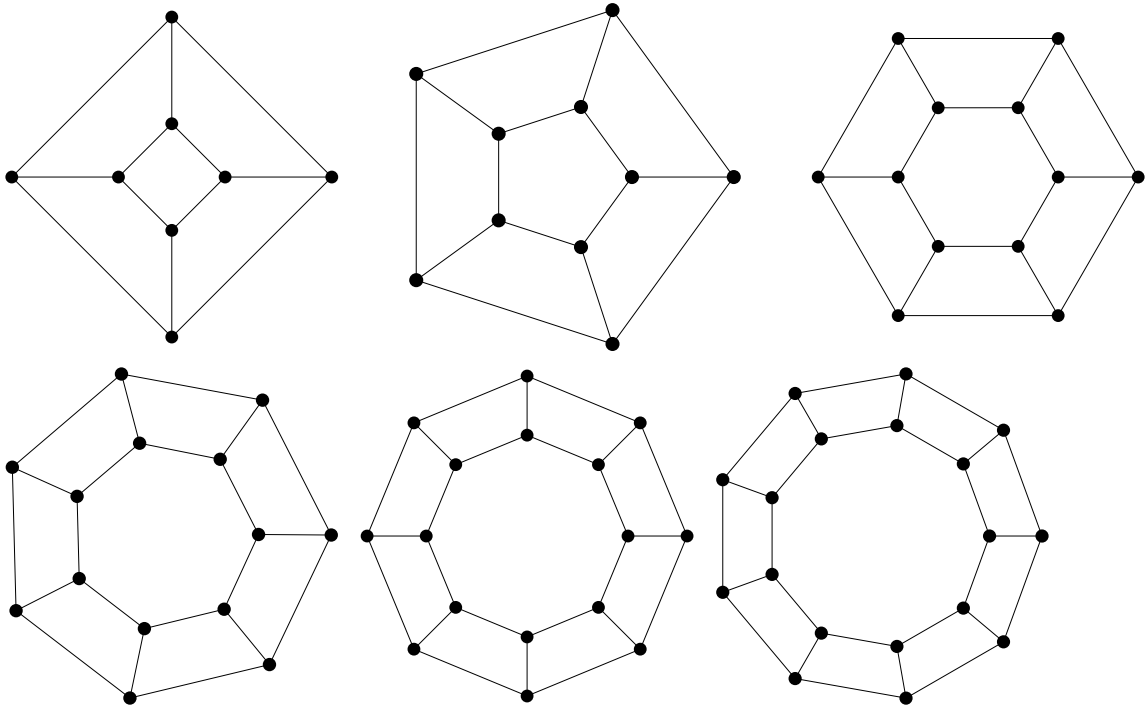


Figure 2.3:  $\Pi_4$  through  $\Pi_9$  as drawn by Tutte's algorithm. Notice that  $\Pi_4$  is isomorphic to the hypercube  $Q_3$

## Symmetry

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

**Theorem 2.3.1** (Eigenvectors of the Prism Graph). *Consider the linear system governing the coordinates of the free vertices of the prism graph  $\Pi_n$ . 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  $x = (x_1, \dots, x_n), y = (y_1, \dots, y_n)$ , where  $x_i$  is the  $x$ -coordinate of the  $i^{\text{th}}$  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  $\lambda_0 \in \mathbb{R}$ .

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

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

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

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  $\lambda_1$  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

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

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



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

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

**Corollary 2.3.1** (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.  $\square$

## 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.

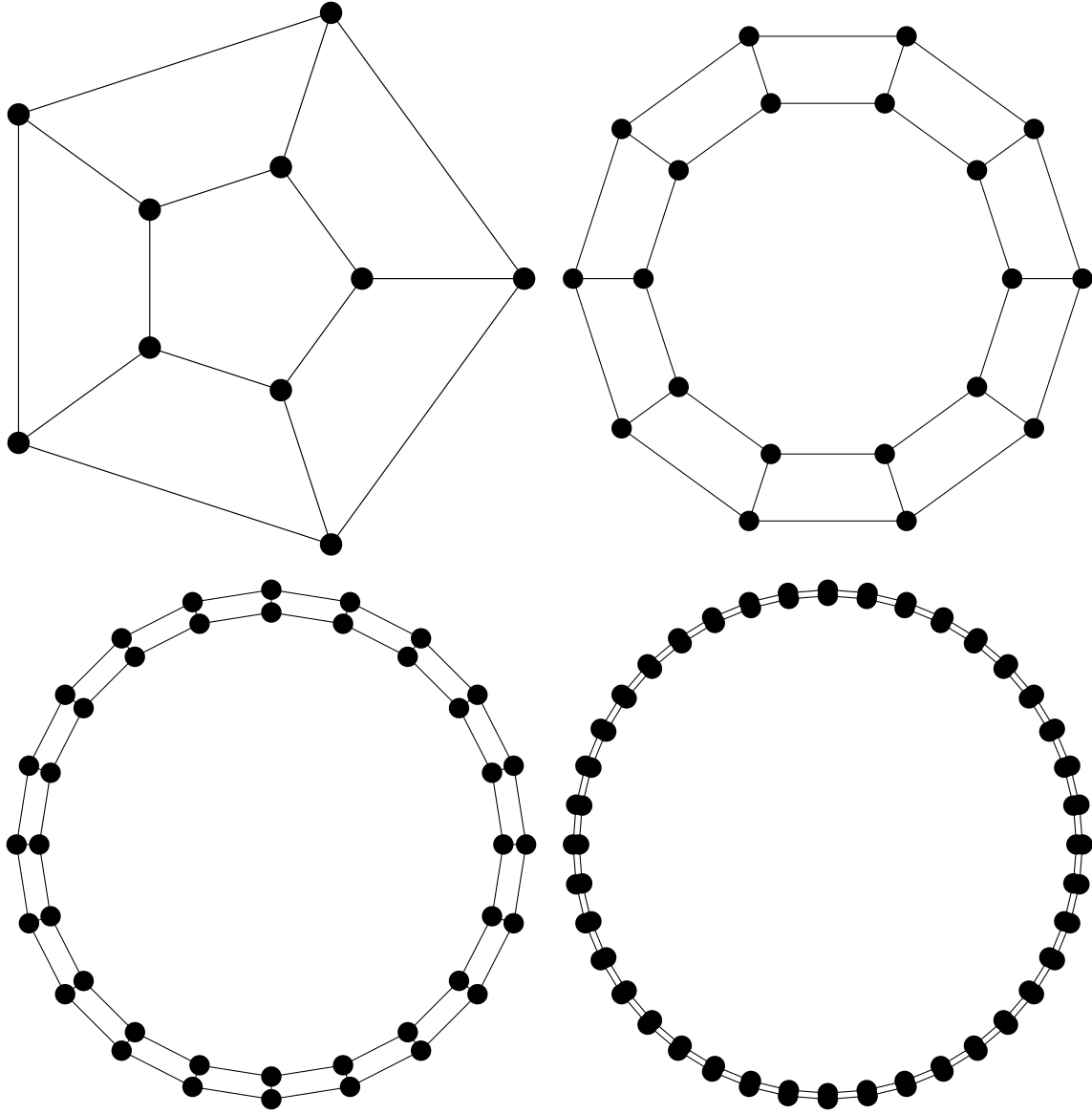


Figure 2.4:  $\Pi_5, \Pi_{10}, \Pi_{20}$  and  $\Pi_{40}$  as drawn by Tutte's algorithm

**Theorem 2.3.2** (Poor Resolution of the Prism Graph). *For every fixed vertex  $u$  in the prism graph  $\Pi_n$ , the distance between it and its adjacent free vertex  $v$  tends to 0 as  $n$  becomes large.*

*Proof.* From the theorem above, we know that

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

Hence,

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

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

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

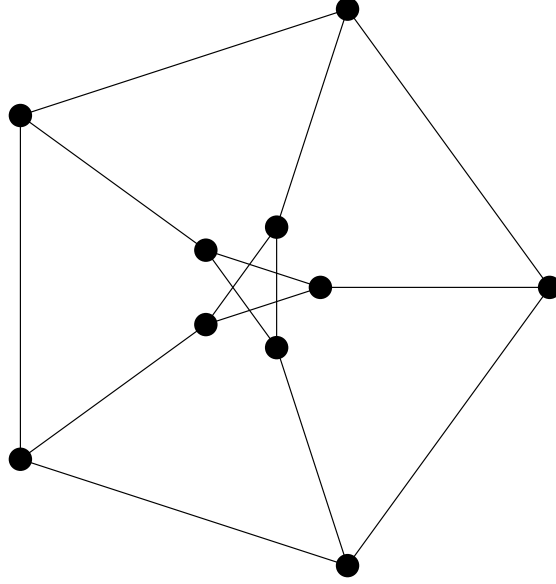
If we take the limit as  $n$  goes to infinity, we get

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

□

## 2.4 Appendix

### 2.4.1 Barycenter Method: Petersen Graph



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

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