# Graph Drawing Algorithms

Vincent La

May 11, 2018

# Contents

# Chapter 1

# Tree Drawing Algorithms

## 1.1 Motivation

Trees with at most two child nodes are used widely in computer science. The simplicity of their structure easily lends to mathematical analysis about algorithms operating on binary trees. Given the vast utility of this tree, many have tried to define algorithms which draw binary trees.

## 1.2 Reingold-Tillford (1981)

One classic algorithm used to layout binary trees is described by Reingold and Tillford.

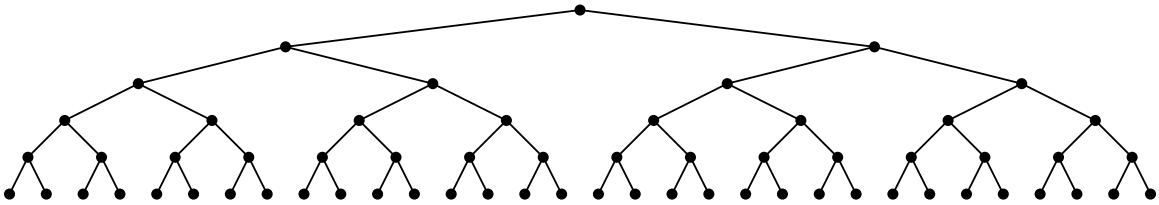
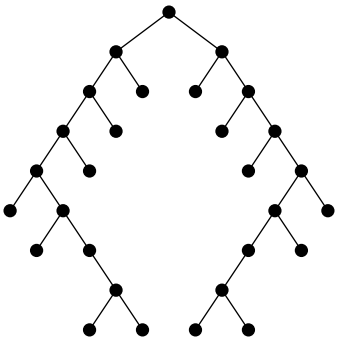
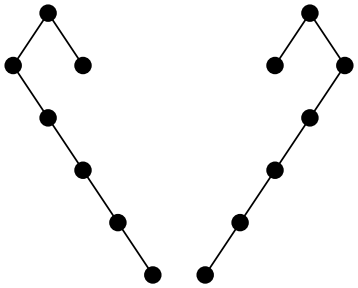


Figure 1.1: A complete binary tree



(a) An example of a tree generated by RT 81



(b) Unlike other algorithms, RT 81 draws a tree and its mirror symetrically

Figure 1.2: A reproduction of some figures from RT's original paper

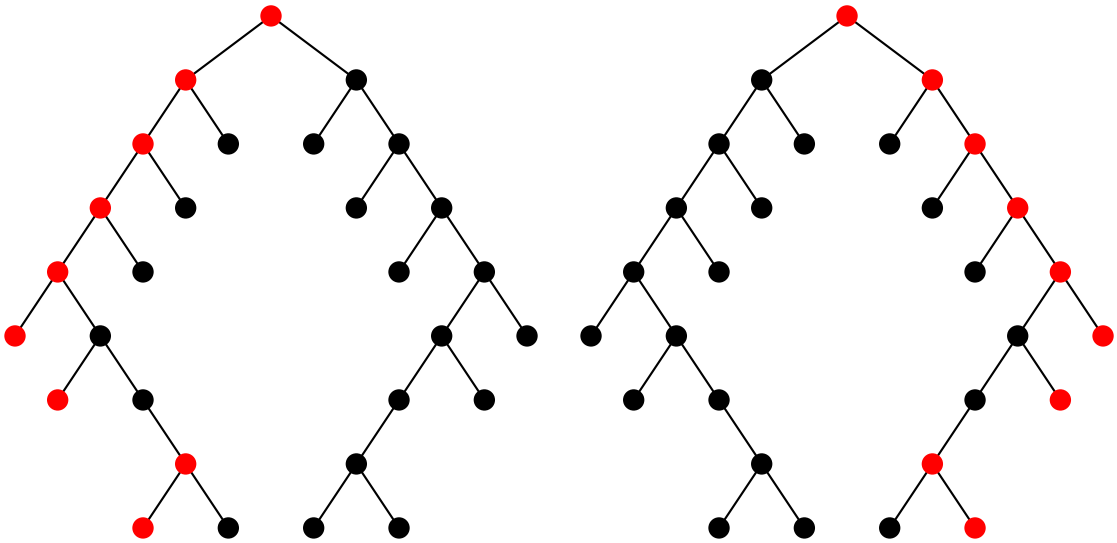


Figure 1.3: The left contour and right contour of the same tree

### 1.2.1 Algorithm Description

#### Informal Description

First, we calculate the displacements of the nodes relative to each other.

1. Base Case: Trivial
2. Apply this algorithm to subtrees via a postorder traversal
3. For each subtree, merge them horizontally such that they are two units apart horizontally

### Formal Description

**Node Type** To implement the tree in a programming language, we will need to creature a structure that represents each node in the tree.

left Pointer to left subtree

right Pointer to right subtree

offset Offset relative to parent

By using offsets relative to a node's parent, as opposed to absolute offsets, we avoid having to reposition all of the nodes in a subtree when the root of that subtree gets moved.

### Threading

For every node in a given tree, the algorithm traverses through the left contour of the right subtree, and the right contour of the left subtree. In this traversal, the goal is to find the appropriate number units to separate the trees by. However, for any given node, its right contour may not be contained entirely within the same subtree. Hence, we require "threads", or connections between nodes in different subtrees in order to follow a tree's contour. Per the visualization below, we may think of threads as temporary edges.

**Algorithm 1** Reingold and Tilford's Algorithm

Constants:
*minsep*: The smallest distance any two subtrees can be separated by [units]

Input:
$t$: A binary tree

```
1: procedure SETUP(t)
2:     cursep = minsep                ▷ Used to keep track of how far apart subtrees are
3:     setup(t→left)                  ▷ Post-order traversal on tree
4:     setup(t→right)
5:     left ← t → left
6:     right ← t → right
7:     while left is not NULL right is not NULL do   ▷ We only have to traverse as deep as the shortest subtree
8:         if cursep < minsep then    ▷ Trees too close, so push them apart
9:             left_dist ← left_dist + (minsep − cursep)/2
10:            right_dist ← right_dist + (minsep − cursep)/2
11:            cursep = minsep
12:        end if
13:        if left → right not null then   ▷ Traverse left subtree
14:            left ← left → right
15:            cursep ← cursep − left.offset
16:        else
17:            left ← left → right
18:            cursep ← cursep − left.offset
19:        end if
20:        if right → left not null then   ▷ Traverse right subtree
21:            right ← right → left
22:            cursep ← cursep − right.offset
23:        end if
24:        if left → right not null then
25:            right ← right → right
26:            cursep ← cursep − right.offset
27:        end if
28:    end while
29:    if left then                   ▷ The left subtree was taller
30:        Insert a thread from the right-most item of the right subtree to right
31:    end if
32:    if right then                  ▷ The right subtree was taller
33:        Insert a thread from the left-most item of the left subtree to right
34:    end if
35: end procedure
```

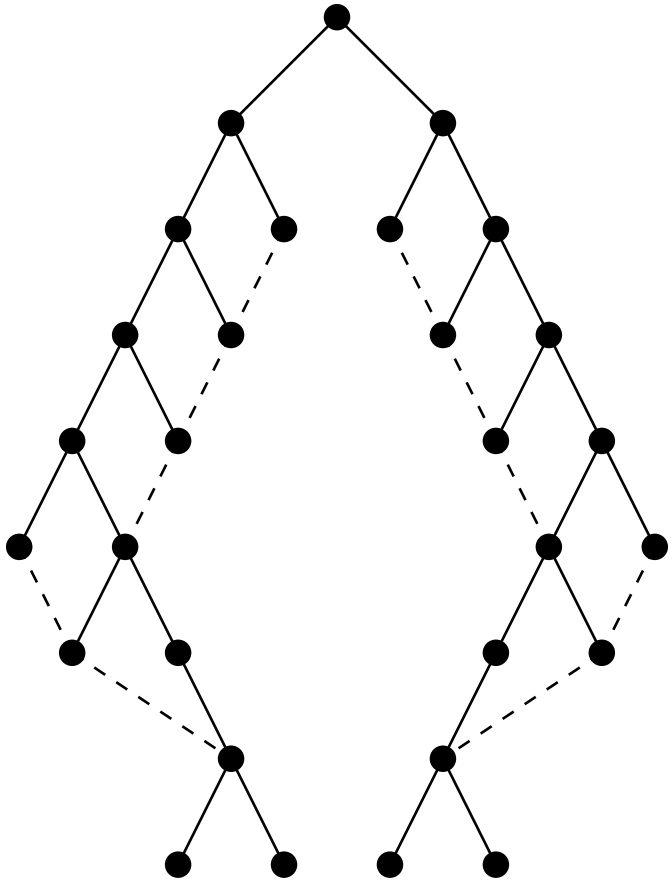


Figure 1.4: Threads (dashed) created by the algorithm while traversing the tree in Figure 1.2a

### 1.2.2 Algorithm Trace for Complete Binary Trees

The figures below show the displacements set by the algorithm for each node. From these figures, we can see how the algorithm achieves symmetry between subtrees.

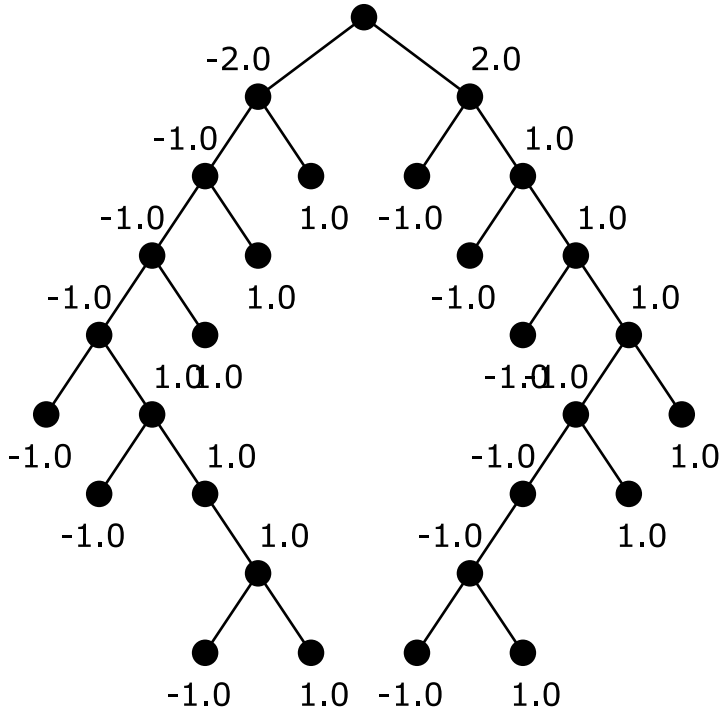



Figure 1.5: Algorithm trace for example tree

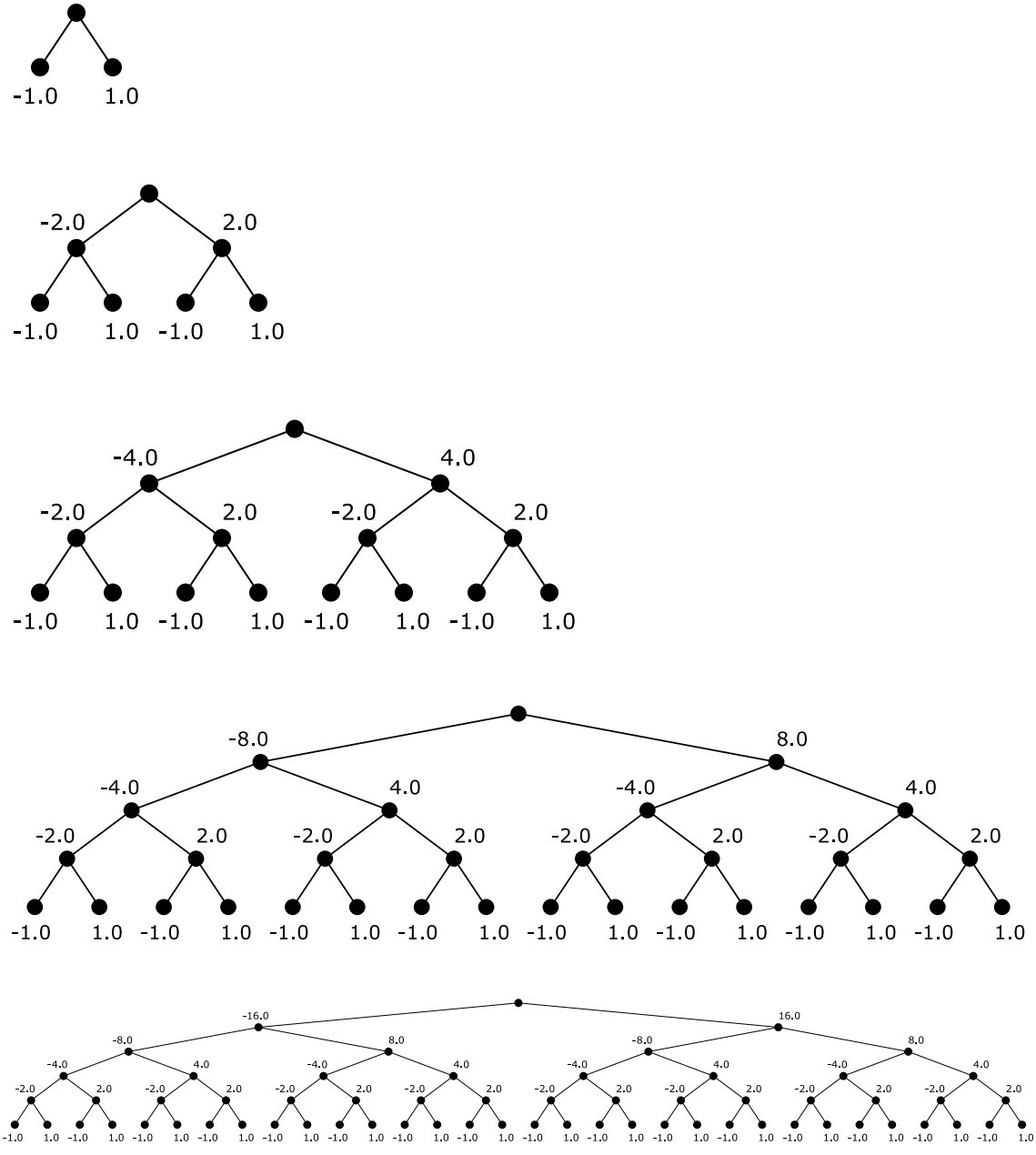



Figure 1.6: Algorithm trace for complete binary trees of heights 2 through 6

# 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

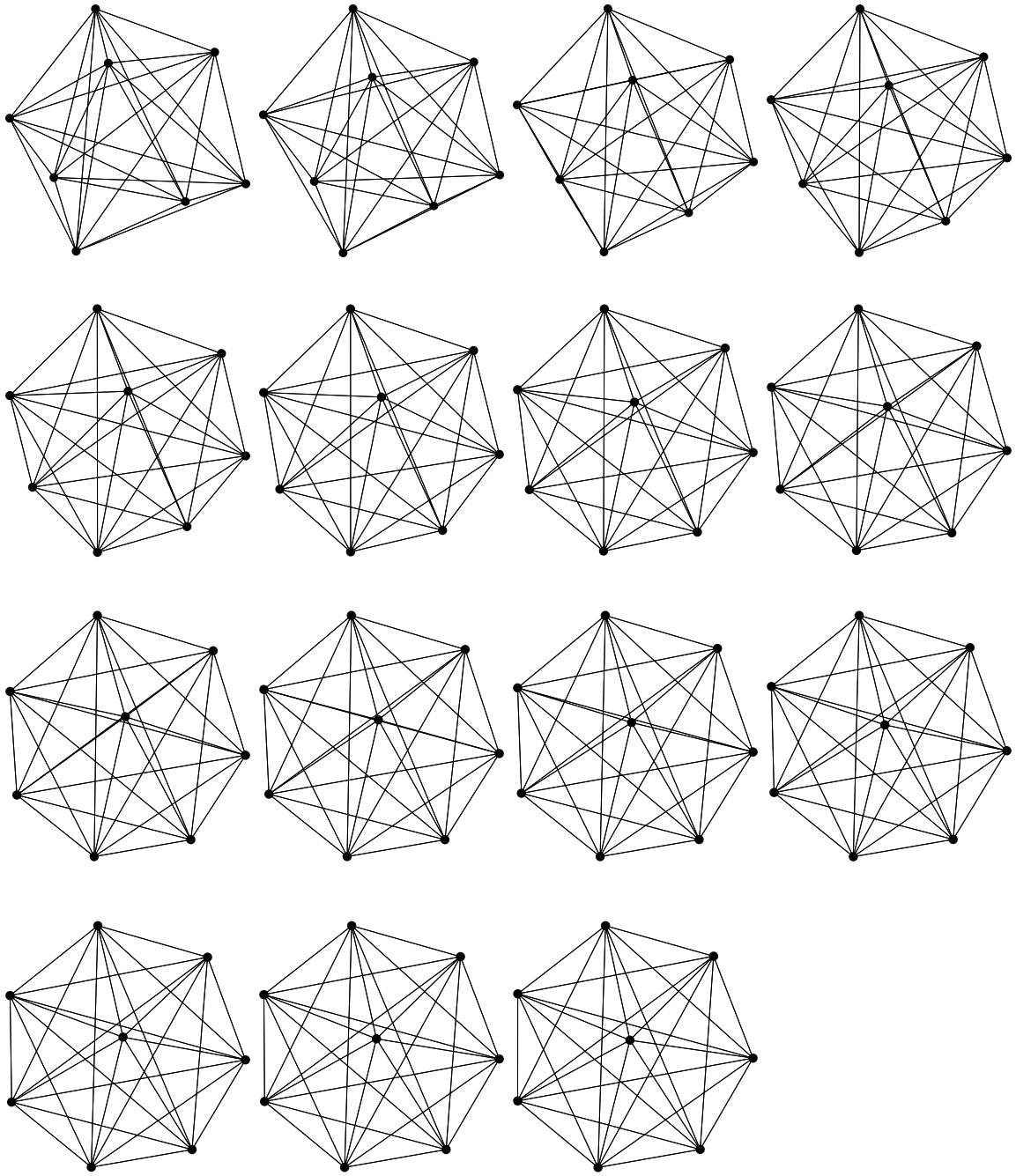


Figure 2.1: The complete graph $K_8$

## 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

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

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

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

### 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

$$\begin{aligned} \deg(v)x_v - \sum_{u\in N_1(v)} x_u = \sum_{w\in N_0(v)} x^*_w \\ \deg(v)y_v - \sum_{u\in N_1(v)} y_u = \sum_{w\in N_0(v)} y^*_w \end{aligned} \tag{2.3}$$

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 $v_1, ..., v_n$, 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.

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

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

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

and the y coordinates by solving

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

### 2.3.3 Example: Hypercube

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

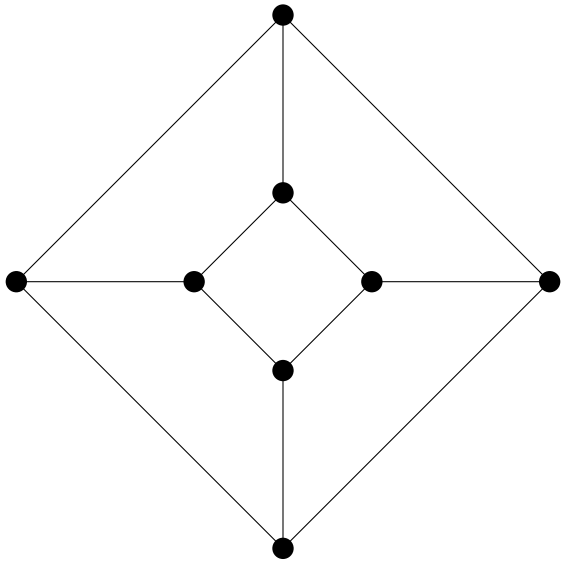


Figure 2.2: The hypercube $Q_3$

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 3 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 2** 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, one can also solve the corresponding linear system directly. In practice, this tends to be significantly faster than the previous

algorithm.

**Algorithm 3** 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).

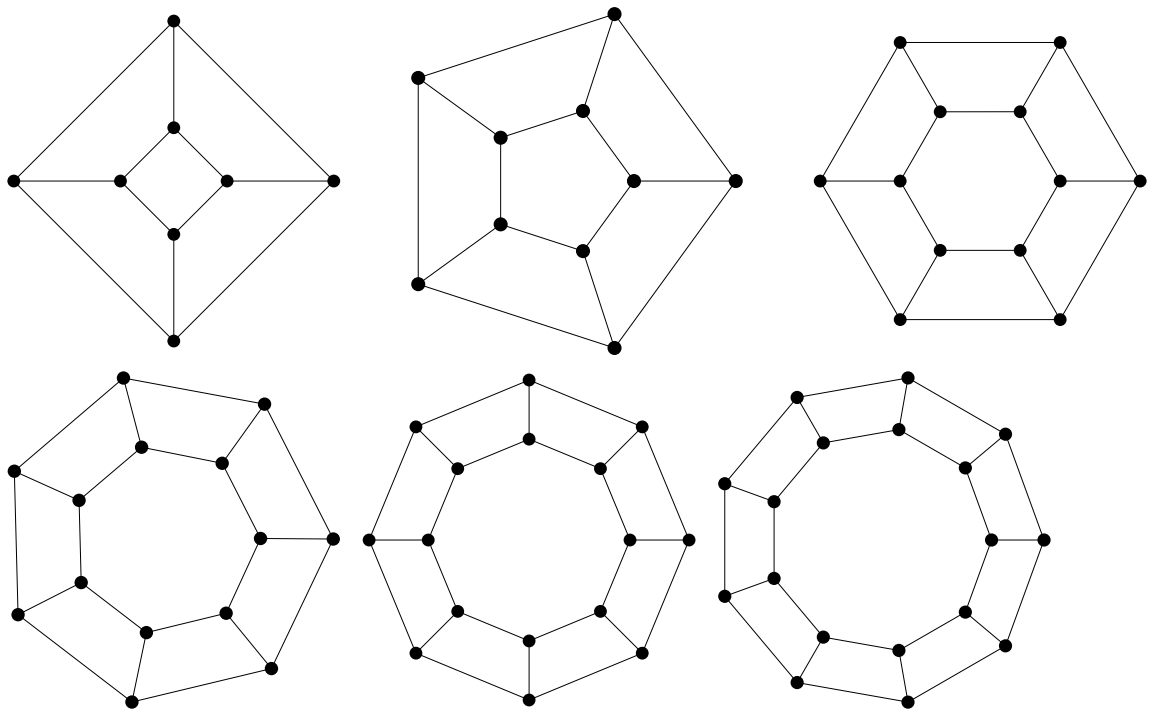


Figure 2.3: $\Pi_4$ through $\Pi_9$ as drawn by Tutte's algorithm. Notice that $\Pi_4$ is isomorphic to the hypercube $Q_3$

**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 $x = (x_1, ..., x_n), y = (y_1, ..., y_n)$, 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 $\lambda_1$ 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

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

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 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. □

### 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.

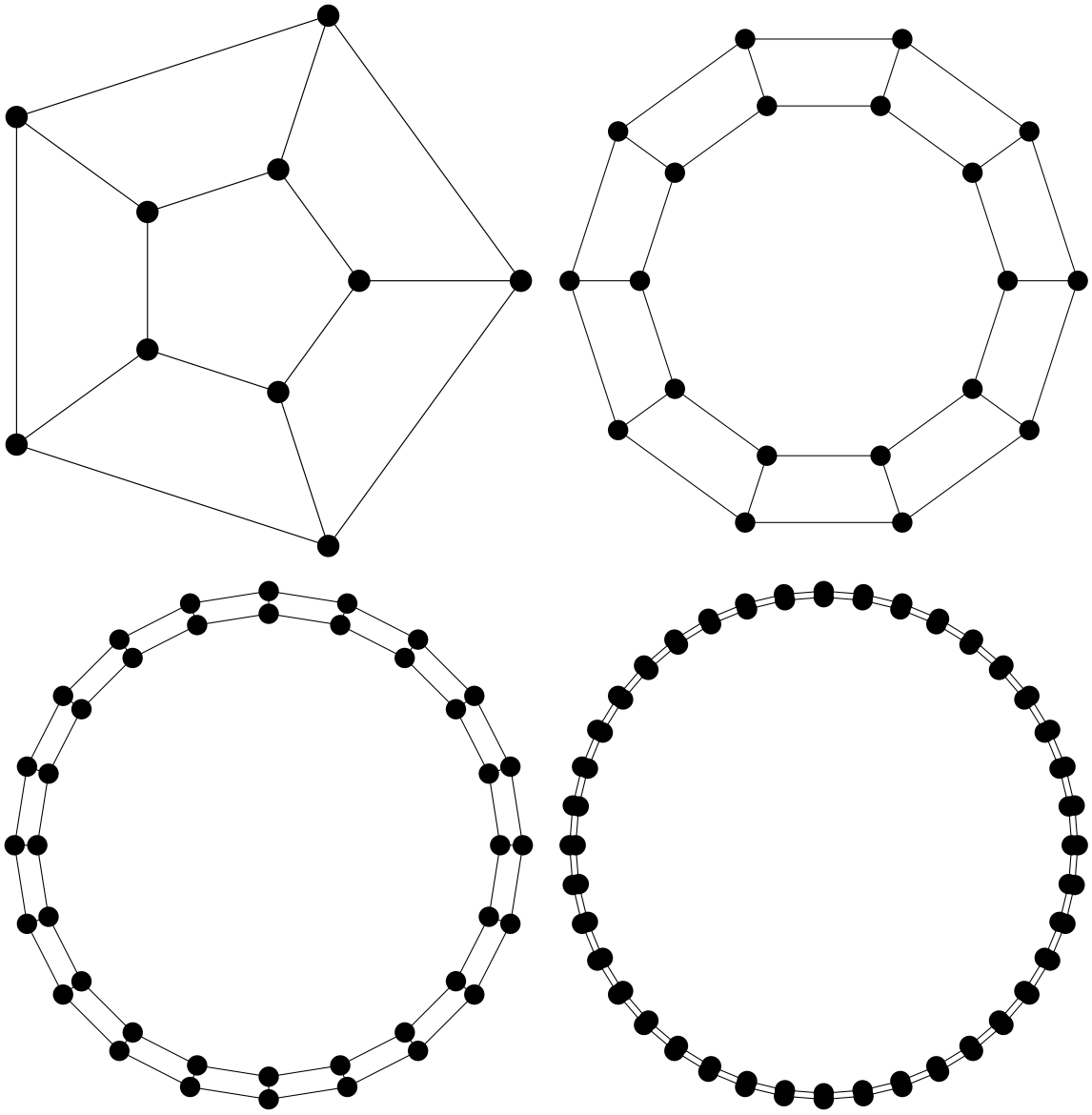


Figure 2.4: $\Pi_5, \Pi_{10}, \Pi_{20}$ and $\Pi_{40}$ as drawn by Tutte's algorithm

**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,

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

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

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

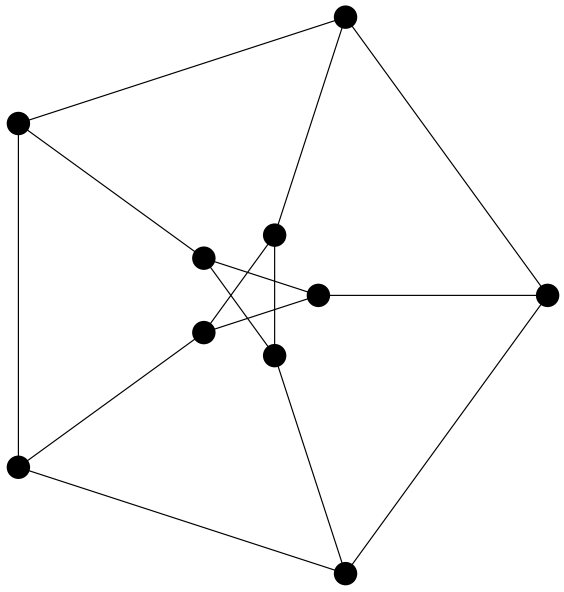
If we take the limit as $n$ goes to infinity, we get

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

□

## 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

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