Multilevel Distance Labeling

1 Introduction and Background

Multilevel distance labeling, also known as radio labeling, is used to model a system of radio stations to have maximum coverage and minimum station to station interference. For a given set of radio stations, the task is to assign each station a non-negative integer channel so that both interference among channels and the span of the channels assigned are minimized.

Usually, the level of interference between any two stations is based upon their geographic locations – the closer the stations are, the stronger the interference. Let us model each station with a node on a graph G, and draw an edge between two nodes if they are geographically "close." Then, the *interference level* between two stations is the distance between the two corresponding nodes of the graph.

Interference among channels can occur at multiple levels. Multilevel distance labeling looks at interference levels ranging from the smallest distance, one, to the largest distance, the diameter of G, denoted $diam(G)$. Then, a *multilevel distance labeling with span* k is a function $F : V(G) \to 0, 1, 2, \ldots, k$, so that for any vertices u and v,

$$|F(u) - F(v)| \geq diam(G) - d(u, v) + 1$$

where $d(u, v)$ is the distance between vertices u and v. Then the radio number of G, denoted $rn(G)$, is the minimum span of the distance labeling of G. The *solution* of G is then a labeling which has $span(G) = rn(G)$

This paper outlines explorations of the multilevel distance labeling problem. The main results include a method to find a lower bound for $rn(G)$, the upper bound for three classes of graphs, and computer programs to aid in calculations and proofs of the distance labeling problem. There is also a discussion on the types of solutions and the number of solutions of graphs.

Note: For the remainder of the paper, $diam(G)$ and D will be used interchangeably to denote the diameter of graph G.

2 Preliminary Investigations and Terminology

2.1 Do not repeat labels

Claim: No two vertices can have the same labeling.

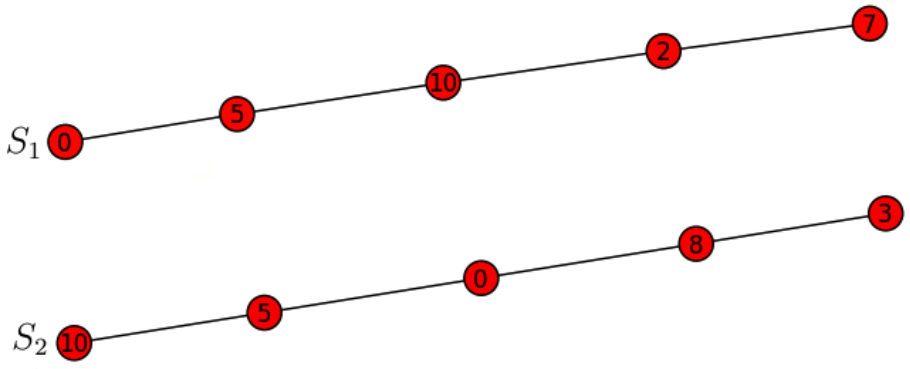
Proof by Contradiction: Assume that there exists two vertices such that $F(x) = F(y)$. By definition, $d(x, y) + |F(x) - F(y)| > diam(G) \Rightarrow d(x, y) > diam(G)$. However, this is a contradiction, as $d(x, y) \leq diam(G)$.

2.2 The Weakest Upper Bound

The loosest upper bound we may give $rn(G)$ is $rn(G) \leq (n-1) \cdot D$. We construct this bound as follows: Arbitrarily assign the terms of the arithmetic sequence $\{0, D, 2D, \ldots (n-1)D\}$ to vertices of G. We verify that this is a distance labeling, as $d(x,y)+|F(x)-F(y)| \geq D+1$, and we have $|F(x)-F(y)| \geq D$ and $d(x,y) \geq 1$.

2.3 The Inverse Solution

Given a solution S_1 of G, we may construct another solution S_2 of G by relabeling each vertex v_i of G as $rn(G) - F(v_i)$. This new assignment is also valid, as $d(x,y) + |(rn(G) - F(x)) - (rn(G) - F(y))| = d(x,y) + |F(x) - F(y)| > diam(G)$, and the maximum label vertex of S_1 is the same as the maximum label vertex of S_2. We call this equivalent solution S_2 the **inverse solution** of S_1. Below is an example of an inverse solution.



2.4 Wiggle Room, Tightness, and Tightness Graphs

Suppose for two vertices x and y, we have $|F(x)-F(y)|+d(x,y) = D+1+k$ with $k \geq 0$. Then, we call k the *wiggle room* between labelings $F(x)$ and $F(y)$, denoted $wr(x,y)$, as this is the extra difference in labeling. We then call situations in which $wr(x,y) = 0$ *tight*.

Using this terminology, we define the tightness graph G_T of a labeled graph G to be the graph of G with all edges removed, and edges added for all pairs of tight vertices. This idea is further explored in Section 5.

2.5 A Flawed Labeling

In initial investigations of the problem, the following labeling algorithm was conceived for the path graph with vertices, in order $\{v_1, v_2, \ldots v_n\}$: Label $F(v_{\lfloor \frac{n+1}{2} \rfloor}) = 0$. Then, for all vertices v_i, with $i < \lfloor \frac{n+1}{2} \rfloor$, label v_i such that $wr(v_i, v_{i+1}) = 0$ and $v_i > v_{i+1}$. Then, label $v_{\lfloor \frac{n+1}{2} \rfloor + 1}$ such that $wr(v_1, v_{\lfloor \frac{n+1}{2} \rfloor + 1}) = 0$. Finally, for the remaining vertices, label v_i such that $wr(v_i, v_{i+1}) = 0$ for $\lfloor \frac{n+1}{2} \rfloor + 1 \leq i \leq n-1$ and $v_i > v_{i+1}$. (S_1 in Section 2.3 is an example of this algorithm.)

It is easily verified that this labeling is valid. We also see that the maximum labled vertex is $v_{\lfloor \frac{n+1}{2} \rfloor + 1}$. After some calculation, we see that

$$F(v_{\lfloor \frac{n+1}{2} \rfloor + 1}) = (n-1)\lfloor \frac{n+1}{2} \rfloor + 1 \approx \binom{n}{2}$$

Although this algorithm gives $rn(G)$ for all paths with at most 6 vertices, it no longer works for 7 or more vertices, as it was disproven by the computer (see Section 6). The insight, however, about this labeling is that it recognizes that $rn(G)$ grows approximately as $\binom{n}{2}$, implying that $rn(G)$ is likely a quadratic function.

2.6 Tight Hopping

We define a hopping to be a sequence of all vertices $\{h_1, h_2, \ldots, h_n\}$ on a graph G, such that $F(h_i) < F(h_{i+1})$ for $1 \leq i \leq n-1$.

A special instance of a hopping is the case where $wr(h_i, h_{i+1}) = 0$ for all $1 \leq i \leq n-1$. We see that there are $n!$ tight hoppings, but not all such tight hoppings are valid distance labelings. Thus, we establish some rules below to ensure that our hopping sequence generates a valid labeling for the case of a path graph P_n.

2.6.1 Tight Hopping Rules on Paths

We cannot hop any arbitrary sequence on a path and expect to have a valid distance labeling. Thus, the following restrictions must be placed on tight hoppings for paths:

1. If our hopping sequence is $h_i \to h_{i+1} \to h_{i+2}$, with $d(h_i, h_{i+1}) > d(h_{i+1}, h_{i+2})$ and $d(x_i, x_{i+2}) = |d(h_i, h_{i+1}) - d(h_{i+1}, h_{i+2})|$, then $d(h_{i+1}, h_{i+2}) \leq \frac{n}{2}$.
2. If our hopping sequence is $h_i \to h_{i+1} \to h_{i+2}$, with $d(h_i, h_{i+1}) < d(h_{i+1}, h_{i+2})$ and $d(x_i, x_{i+2}) = |d(h_i, h_{i+1}) - d(h_{i+1}, h_{i+2})|$, then $d(h_i, h_{i+1}) \leq \frac{n}{2}$.

Notice that if our hopping sequence is $h_i \to h_{i+1} \to h_{i+2}$, with $d(x_i, x_{i+2}) = d(h_i, h_{i+1}) + d(h_{i+1}, h_{i+2})$, then $\min(d(h_i, h_{i+1}), d(h_{i+1}, h_{i+2})) \leq \frac{n}{2}$, (as we cannot partition a path of length n into two parts with length greater than $\frac{n}{2}$.). With this, we notice that the restrictions placed on tight hoppings are equivalent to the following restriction:

$$\min(d(h_i, h_{i+1}), d(h_{i+1}, h_{i+2})) \leq \frac{n}{2}$$

Let us prove that this rule will make the tight hopping on a path a valid distance labeling, by showing that this labeling satisfies the definition $|F(u) - F(v)| \geq diam(G) - d(u, v) + 1$ for any two vertices u and v.

First, we note that any two vertices h_i and h_j with $|i - j| = 1$ will satisfy the relationship, as $wr(h_i, h_j) = 0$.

Further, any two vertices h_i and h_j with $|i - j| = 2$ satisfy. Let $d(h_i, h_{i+1}) = d_1$ and $d(h_{i+1}, h_{i+2}) = d_2$. Now, we may express $d(h_i, h_{i+2}) = d_3$ in terms of d_1 and d_2. First, we establish the value of h_{i+2} compared to h_i.

$$F(h_{i+1}) = D + 1 - d_1 + F(h_i) \tag{1}$$

$$F(h_{i+2}) = D + 1 - d_2 + F(h_{i+1}) \tag{2}$$

From equations (1) and (2) we get $F(h_{i+2}) - F(h_i) = 2D + 2 - d_1 - d_2$.

Now there are two cases:

Case 1 - $d_3 = d_1 + d_2$**:** Clearly, if two hops are in the same direction, we have $d_3 = d_1 + d_2$. Then, by the distance labeling definition, $F(h_{i+2}) - F(h_i) \geq D + 1 - d_3$. Since we already know $F(h_{i+2}) - F(h_i) = 2D + 2 - d_1 - d_2$, we have

$$2D + 2 - d_1 - d_2 \geq D + 1 - (d_1 + d_2).$$

Then, since $D + 1 = n$ we have $n \geq 0$, which is obviously true. Thus, when hopping twice on a path in the same direction, there are no restrictions on the values of d_1 and d_2, other than $d_1 + d_2 \leq n \Rightarrow \min(d_1, d_2) \leq \frac{n}{2}$ as desired.

Case 2 - $d_3 = |d_1 - d_2|$**:** In this case, the two hops are in opposite directions. Without loss of generality, we may let $d_1 > d_2$. Then, by definition we have $F(h_{i+2}) - F(h_i) \geq D + 1 - d_3$. Again, since $F(h_{i+2}) - F(h_i) = 2D + 2 - d_1 - d_2$, we have

$$2D + 2 - d_1 - d_2 \geq D + 1 - |d_1 + d_2|.$$

Since $d_1 > d_2$, we get $n \geq 2d_1 \Rightarrow \frac{n}{2} \geq d_1$ as desired.

Finally, for any two vertices h_i and h_j with $i - j \geq 3$ we satisfy the relationship, as the increase in h_i is great enough to not conflict with h_j.

2.6.2 Verifying Path Labelings

With Section 2.6.1, we have shown that any tight hopping on a path that satisfies

$$\min(d(h_i, h_{i+1}), d(h_{i+1}, h_{i+2})) \leq \frac{n}{2}$$

for $1 \leq i \leq n - 2$ is a valid distance labeling.

Thus, if we can verify that a labeling on a path P_n is a tight hopping that satisfies this condition, we know that it is a valid distance labeling. However, as will be later discovered, not all solutions are tight hoppings. If a solution is not a tight hopping, the condition $\min(d(h_i, h_{i+1}), d(h_{i+1}, h_{i+2})) \leq \frac{n}{2}$ will no longer be true for all $1 \leq i \leq n - 2$. This problem is solved with the *contribution* of a vertex $cb(x_i)$ found in the next section.

3 The Lower Bound

There are two methods by which we may establish a lower bound for a graph G.

3.1 Total Hopping Distance

Let us consider the vertices of G in order of increasing label. Then, if G has vertices $V(G) = \{v_1, v_2 \dots v_n\}$, let $\{x_1, x_2 \dots x_n\}$ be a permutation of $V(G)$ such that $F(x_{i+1}) > F(x_i)$ for all $1 \leq i \leq n - 1$.

For convenience, let $F(x_{i+1}) - F(x_i) = f_i$ and $d(x_{i+1}, x_i) = d_i$. By definition we have $F(x_1) \geq 0$ and $f_i \geq D + 1 - d_i$. We also define the *contribution* of a vertex $cb(x_i) = f_i + d_i - D - 1$. We see that f_i is minimized when $cb(x_i) = 0$ (This part may belong in either Section 2 or Section 4).

Now, we note that the maximum labeled vertex $F(x_n) = \sum_{i=1}^{n-1} f_i$. Then, by summing up the inequalities, we have

$$F(x_n) = \sum_{i=1}^{n-1} f_i \geq \sum_{i=1}^{n-1} [D + 1 - d_i]$$

$$\Rightarrow F(x_n) \geq (n-1)(D+1) - \sum_{i=1}^{n-1} d_i$$

Thus, we see that if we can maximize $\sum_{i=1}^{n-1} d_i$ on a graph G, we will have a lower bound for $rn(G)$.

We call $\sum_{i=1}^{n-1} d_i$ the *total hopping distance*, as it is the sum of all the distances as we *hop* over a sequence of the n vertices of G. Then we see that our lower bound makes sense, as increasing the total hopping distance decreases the amount we must increment the labelings.

This total hopping distance can be maximized for several classes of graphs, two of which will be explored in Section 4.

However, for a general graph G, finding the maximum total hopping distance seems to be an NP-complete problem.*

3.2 Minimum Increase

This method will be completed later.

4 The Upper Bound - Proofs for Paths, Cycles, and Lollipops

There are certain classes of graphs for which we may compute the lower bound and construct an upper bound matching the lower bound. This ability to compute the lower bound is related to the ability to find the maximum hopping distance. It follows that paths and cycles are special cases when trying to find the maximum hopping distance, as distances are found by subtracting vertex numbers.

4.1 Paths

A *path* graph P_n is defined as a graph G with vertices $V(G) = \{v_1, v_2 \dots v_n\}$ and edges (v_i, v_{i+1}) for all $1 \geq i \geq n-1$. We proceed to prove that for any $n \geq 4$,

$$rn(P_n) = \begin{cases} (k+1)^2 + (k-1)^2 & : n = 2k+1 \\ (k-1)^2 + k^2 & : n = 2k \end{cases}$$

4.1.1 Lower Bound

From Section 3, we know that $F(x_n) \geq (n-1)(D+1) - \sum_{i=1}^{n-1} d_i$.

It happens that the maximum hopping distance is different for even length and odd length paths.

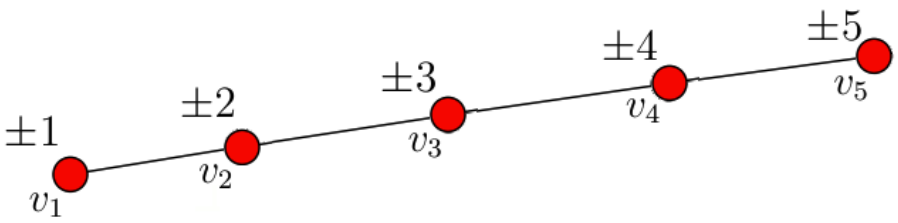
a) Odd Length Paths

Note: if we have $\sum_{i=1}^{(2k+1)-1} d_i \leq 2k^2 + 2k - 2$, then we are done, as

$$F(x_n) \geq ((2k+1)-1)(2k+1) - (2k^2+2k-2) \geq (k+1)^2 + (k-1)^2.$$

Claim: if $\sum_{i=1}^{2k} d_i > 2k^2+2k-2$, then $\sum_{i=1}^{2k} d_i = 2k^2+2k-1$, and there exists a vertex v_i such that $wr(v_i) = 1$.

Proof of Claim: Since each $d(x_i, x_{i+1}) = |j - j'|$ if $x_i = v_j$ and $x_{i+1} = v_{j'}$, $\sum_{i=1}^{2k} d_i$ is the sum of $4k$ j's where half of the j's are positive, the other half are negative, and $1 \leq j \leq 2k+1$. Furthermore, $2k-1$ terms appear twice, and 2 terms appear once. (The terms appearing once represent the min and max labeled vertex).



Then, to maximize $\sum_{i=1}^{2k} d_i$, we need to minimize the absolute values of the negative terms and maximize the values of the positive terms. There are two cases achieving this maximum summation:

Case 1- We have positive values of j for $\{k+2, k+3, \ldots 2k+1\}$, each of which appears twice (note that this is 2k terms), negative values of j for $\{1, 2, \ldots k-1\}$, each of which appears twice, and negative values for k and $k+1$ both appearing once.

Case 2- We have positive values of j for $\{k+3, k+4, \ldots 2k+1\}$, each of which appears twice, positive values for $k+1$ and $k+2$, and negative values of j for $\{1, 2, \ldots k\}$, each of which appears twice.

In both cases, we get

$$\sum_{i=1}^{2k} d_i = 2k^2 + 2k - 1.$$

In Case 1, we have $v_{k+1} = x_1$ and $v_k = x_{2k+1}$. (It does not matter if the positions of x_1 and x_{2k+1} are switched due to the inverse solution). Since each d_i is composed of a positive component and a negative component, we know that if $x_i \geq k+2$ (in part B below), then $x_{i+1} \leq k+1$.

$$\underbrace{\overbrace{1,2,3\ldots k-1}^{\text{A}}}_{2(k-1)},\ \overbrace{x_1\ x_{2k+1}}^{}\ k,k+1,\underbrace{\overbrace{k+2,k+3,\ldots 2k+1}^{\text{B}}}_{2k}$$

Now, consider $x_i = 1$. Then, $x_{i-1} \geq k+2$ and $x_{i+1} \geq k+2$. (The previous and next vertex in the sequence are both in part B above.) However, this contradicts our tight hopping criterion above (Section 2.6.2). Thus, since both distances $d_{i-1}, d_i \geq \frac{n}{2}$, we know that either $cb(x_{i-1}) = 1$ or $cb(x_{i+1}) = 1$, which forces our maximum value to increment by at least 1.

In Case 2, we have $v_{k+1} = x_1$ and $v_{k+2} = x_{2k+1}$. Like the previous case, we know that if $x_i \geq k+1$, then $x_{i+1} \leq k$. Now, consider $x_i = 2k+1$. Then, $x_{i-1} \leq k$ and $x_{i+1} \leq k$. This again contradicts our hopping criterion above, as both distances $d_{i-1}, d_i \geq \frac{n}{2}$, we know that either $cb(x_{i-1}) = 1$ or $cb(x_{i+1}) = 1$, which forces our maximum value to increment by at least 1.

By taking the sum of all inequalities and accounting for the contribution of 1, we have

$$F(x_{2k+1}) \geq ((2k+1)-1)(2k+1) - (2k^2+2k-1) - 1 \geq (k+1)^2 + (k-1)^2.$$

and we have shown the lower bound for odd paths.

b) Even Length Paths Thankfully, finding this lower bound is simpler than for odd length paths, as there is only one way to maximize the hopping distance.

Claim: $\sum_{i=1}^{2k-1} d_i \leq 2k^2 - 1$

Proof of Claim: Using the above logic, we know that we have $4k-2$ terms of $1 \leq j \leq 2k$, $2k-2$ of which occur twice and 2 of which occur once. Again, half of these terms are positive and the other half are negative. The maximization of this sum occurs when we have positive values of j for $\{k+2, k+3, \ldots 2k\}$, each of which appears twice, positive values for k and $k+1$ each appearing once, and negative values of j for $\{1, 2, \ldots k-1\}$, each of which appears twice.

From this, we get

$$\sum_{i=1}^{2k-1} d_i \leq 2k^2 - 1.$$

It follows that

$$F(x_{2k}) \geq 2k(2k-1) - (2k^2-1) = (k-1)^2 + k^2,$$

as desired.

4.1.2 Upper Bound

a) Odd Length Paths We label the vertices of G as follows:

x_1	v_k
x_2	v_{k+k}
x_3	v_1
x_4	v_{1+k}
x_5	v_{1+k+k}
x_6	v_3
x_7	v_{3+k}
x_8	v_4
x_9	v_{4+k}
x_{10}	v_5
x_{11}	v_{5+k}
...	...
x_{2k-2}	v_{k-1}
x_{2k-1}	v_{k-1+k}
x_{2k}	v_2
x_{2k+1}	v_{2+k}

where $x_1 = 0$, and each x_i is tight with x_{i+1} for $1 \leq i \leq n-1$. We proceed to show that this tight hopping is a distance labeling by checking it against the restrictions placed in Sections 2.6.1 and 2.6.2.

Note: This labeling only works for odd paths with $n \geq 7$. In the case of $n = 5$, x_{2k+1} is too great with the above labeling. However, the case $n = 5$ does follow the formula given above (check Appendix A for all solutions to the P_5). Furthermore, all solutions generated with the above labeling have G_T which are also path graphs. (See Section 5).

Proof of Upper Bound: Using our distance labeling verification method for paths from Section 2.6.1 and Section 2.6.2, we need $min(d_i, d_{i+1}) \leq \frac{n}{2}$. Checking the labeling above, for every three consecutive vertices, at least one adjacent pair is k apart. As $k < \frac{n}{2}$, we see that the above labeling method is valid.

Now, we need to show that the above method achieves the upper bound $x_{2k+1} = (k+1)^2 + (k-1)^2$. Since this is a tight hopping, there are no contributions from any vertices. Therefore,

$$x_{2k+1} = \sum_{i=1}^{2k} [D + 1 - d_i].$$

Then we get $x_{2k+1} = 2k(2k+1) - \sum_{i=1}^{2k} d_i$, and once again it comes down to find the hopping distance of this specific labeling scheme.

However, finding the hopping distance of a labeling algorithm is not so difficult, and some algebra verifies that $\sum_{i=1}^{2k} d_i = 2k^2 + 2k - 2$, giving $x_{2k+1} = (k+1)^2 + (k-1)^2$.

a) Even Length Paths We label the vertices of G as follows:

x_1	v_k
x_2	v_{k+k}
x_3	v_2
x_4	v_{2+k}
x_6	v_3
x_7	v_{3+k}
x_8	v_4
x_9	v_{4+k}
x_{10}	v_5
x_{11}	v_{5+k}
...	...
x_{2k-2}	v_{k-1}
x_{2k-1}	v_{k-1+k}
x_{2k}	v_1
x_{2k+1}	v_{1+k}

where $x_1 = 0$, and each x_i is tight with x_{i+1} for $1 \leq i \leq n-1$. We again check against the restrictions placed in Sections 2.6.1 and 2.6.2.

Proof of Upper Bound: We need $min(d_i, d_{i+1}) \leq \frac{n}{2}$. Checking the labeling above, for every three consecutive vertices, at least one adjacent pair is k apart. As $k < \frac{n}{2}$, we see that the above labeling method is valid.

Now, we need to show that the above method achieves the upper bound $x_{2k+1} = (k-1)^2 + k^2$. Since this is a tight hopping, there are no contributions from any vertices. Therefore,

$$x_{2k} = \sum_{i=1}^{2k-1} [D + 1 - d_i].$$

Then we get $x_{2k+1} = (2k-1)2k - \sum_{i=1}^{2k-1} d_i$. Some algebra reveals that $\sum_{i=1}^{2k} d_i = 2k^2 - 1$, giving $x_{2k+1} = (k+1)^2 + (k-1)^2$.

With this, we have solved the problem of finding the minimum span radio labeling for the path graph.

5 Further Work - Tightness Graphs

5.1 Introduction

Tightness graphs, G_T, are an interesting way to categorize solutions, as they reveal the underlying structure of a solution. One nice application of the tightness graph is its use in generating a labeling algorithm for new graph types. This could help generate an upper bound for the graph type in general. Using the Python code in Section 6, we may compare the tightness graphs of various numbers of vertices of the same graph type. So far, it appears that solutions with similar tightness graphs follow the same labeling algorithm. Thus, by looking at the solutions that generate tightness graphs that follow a pattern, we may devise new labeling algorithms. This idea is further discussed and explained in Appendix C. (See me about lollipop work).

5.2 Structures and Examples

Look in my email for various structures and examples of tightness graphs for both paths and lollipops. Another way to obtain these tightness graphs is to generate them on your own computer using the Python code below in Section 6 (and in the email attachment). A read-me with instructions on how to use the code is included in the email as well.

Notice that there are recurring trends in odd length and even length tightness graphs for paths. A notable trend is that all odd length path solutions have tightness graphs which are also paths. Incidentally, the construction of the upper bound in Section 4 created exactly these solutions. The construction of the upper bound for even length paths constructed the tightness graph that looked like a ladder. Thus, when given solutions to a new type of graph, inspecting the tightness structures is a quick way to identify and organize the solutions.

5.3 G_T on Odd and Even Paths

Although there are recurring trends on odd length path tightness graphs, there are also irregularities throughout the solutions. Notably, there is usually one "limb" sticking out of the tightness graph for odd paths. However, this phenomena can be explained by our discussion of maximum hopping distance of odd-length paths in Section 4. From the claim on page 6, we see that the maximum hopping distance involves one vertex having $cb(x_i) = 1$. Precisely the vertex x_i for which $cb(x_i) = 1$ is the vertex that is "sticking out" of the rest of the graph. Although exactly why this is the case is not known, it does show that G_T reflects trends in the maximum hopping distance. I am fairly certain that the sum of the maximum hopping distance and the longest path that does not visit the same vertex twice between the min and the max is a constant value.

5.4 Future Work

My hope is, if we prove certain properties about G_T, then finding the lower bound by using maximum hopping distance from Section 3 will result in $rn(G)$ itself, and not just a lower bound. (in other words, proving some nice properties might make it such that labeling the vertices based on a maximum hopping distance always generates a valid distance labeling. If our lower bound is a valid labeling, we clearly win).

First, we prove a quality of G_T: **Every solution on any graph G has a tightness *chain* from min to max**. (By chain, we mean that there is a walk from the min vertex to the max vertex on G_T).

Proof by Contradiction: Assume there is not a chain between min (x_0) and max (x_n) in G_T if the labeling on G is a solution.

Let us define the sets $L_0, L_1, L_2, \ldots L_m$ such that $x_1 \in L_0$ and $x_n \in L_m$. Furthermore, let L_i be the set of all vertices which are tight with the vertices of L_{i-1}. Now, suppose that L_1 is empty. Then, we have a contradiction, as the labeling on G is certainly not a solution, as we can decrease the span of the labeling by 1 by adding 1 to x_0. By induction, if any L_i with $0 \leq i \leq m$ is empty, we know that the labeling on G is not a solution. Thus, we see that our initial assumption was false, and that every solution has a chain

linking min to max on G_T.

Even though we have proved this simple result, there is still a whole lot of properties up in the air about tightness graphs. First of all, we don't know if the chain must have increasing labels from min to max. We don't even know that every vertex from G is on G_T, although I am almost certain that this is the case. I am also fairly certain that $d(x_0, x_n) = diam(G_T)$.

In conclusion, there is still quite a bit of work to be done on G_T, in addition to other ways of looking at the problem. The main advantage that I have found for G_T is its ability to categorize different types of solutions to help find labeling algorithms.

6 Finding Solutions

6.1 Solutions by Hand

Sometimes excitement and eagerness gets in the way of trying to code a solution: you have to try it out by hand!
Here is the fastest method I have found to do so.

Fix the min and the max If we have a conjecture for the value of $rn(G)$, we may find two vertices on G and label one the min (0) and the other the max (the conjectured value). Having two values to start each case reduces the computations by giving a range of possible values for all other vertices. In general, one must deal with around $\frac{1}{2} \cdot \binom{n}{2}$ cases to exhaust all possibilities. This is not too bad for graphs G with less than 7 or 8 vertices.

6.2 Using a Computer

The multilevel distance labeling problem is computationally heavy when it comes to finding solutions. It is somewhat illogical to not use a computer here.

6.2.1 Python Code

The following Python Code takes a value n for the number of vertices on the path, and outputs all computed solutions to the case. Another more versatile and useful program is in the email.

```
def solution(n):
    """prec: give solution a number n.
    postc: it prints out all solutions with n vertices as lists"""
    #Set the number of vertices on the graph with respect to entered n
    vertexNumber=n
    #The output list
    out=[0]*vertexNumber
    #Set the diameter!
    diam=n-1
    #This is the loosest lower bound for rn(G) that applies for all graphs.
```

```
    maxVal=diam*(vertexNumber-1)
    optionLibrary=[]
    #Set the distance matrix! This can be customized for different graphs.
    #This is the path graph distance matrix.
    dist=[[abs(i-j) for i in range(n+1)] for j in range(n+1)]
    def checkItem(v,idx,maxVal):
        """prec: takes current list progress, index, and maximum value
        postc: checks to see if value at index agrees with previous
        vertices"""
        n=len(v)
        for prev in range(idx):
            if v[prev]>maxVal:
                return False
                #Checks to see if previous vertices check with labeling
                #at current node, if not, we return false.
        for prev in range(0,idx):
            if (abs(v[prev]-v[idx])<(n-dist[prev][idx])):
                return False
        return True
    def generateOption(idx,maxVal):
        """prec: takes current index and maximum value
        postc: generates a list of the  possible next vertices and puts in
        library"""
        if idx==0:
            #to generate option for first list
            nextOption=list(range(maxVal+1))
            return nextOption
        else:
            #to generate options for all other lists
            nextOption=[]
            for label in range(maxVal+1):
                out[idx]=label
                if checkItem(out,idx,maxVal):
                    nextOption.append(label)
            return nextOption
    def libraryCleaner(maximumValue):
        """prec: takes a new value for maxVal
        postc: cleans up the library to account for change in maxVal"""
        for i in range(len(optionLibrary)):
            j=0
            while j in range(len(optionLibrary[i])):
                if optionLibrary[i][j]>maximumValue:
                    del optionLibrary[i][j]
                    j-=1
                j+=1
    idx=0
#######THE DEPTH FIRST SEARCHER#######
#While our searcher is in the list, it will generate next option if needed
```

```
    while idx in range(vertexNumber):
        if len(optionLibrary)==idx:
            optionLibrary.append(generateOption(idx,maxVal))
        #Prune off all empty option arrays at end of library.
        #These occur when the "searcher" is stuck
        while optionLibrary[-1]==[]:
            del optionLibrary[-1]
            idx-=1
            #When the complete case is exhausted,
            #the option library will be empty.
            if optionLibrary==[]:
                return ''
        #Removes a value from the option library and tries it out.
        out[idx]=optionLibrary[idx][0]
        del optionLibrary[idx][0]
        idx+=1
#If we are at the last index, put the potential solution to stdout
        if idx==vertexNumber:
            print(out)
            if maxVal>max(out):
                maxVal=max(out)
                libraryCleaner(maxVal)
            idx-=1
    return
```

6.2.2 JavaScript Code

This code is provided for readers without Python who would like to verify the solutions provided in Appendix A, or to use as a convenient tool to check the solutions that they have found by hand. JavaScript is particularly easy to use, as it will run on any major web browser (by using Developer Tools on Google Chrome for example).

The following two JavaScript code excerpts are used to collect data about the path case. The path is represented as an array.

This function verifies whether a given path labeling is a valid distance labeling. Copy this function into JavaScript, and try "isgood(someArray)" to see if your array represents a valid labeling. If valid, the function returns True. Otherwise, it returns False.

```
function isgood(arr) {
        var n = arr.length - 1;
        var result = true;

        for (var i=0; i<=n; i++) {
                for (var j=0; j<=n; j++) {
                        if (j <= i) continue;
                        var d = arr[j] - arr[i];
```

```
                        if (d<0) d = -d;
                        if (d + j - i <= n) result = false;
                }
        }
        return result;
}
```

This function prints out every pair of vertices that are tight on a path graph. Again, copy the function into JavaScript, and try "tight(someArray)" and the function will return a list of the tight vertices

```
function tight(arr) {
        var n = arr.length;
        var m = 0;
        var t = "";
        for (var i=0; i<n; i++) {
                for (var j=0; j<n; j++) {
                        var d = Math.abs(arr[j] - arr[i]);
                        if(d+j-i-n == 0) t = t + '('+arr[i]+','+arr[j]+')='
                        + (d + j - i - n) + '\n';
                                m++;
                }
        }
        return t;
}
```

Appendix A: Path Solutions

These are all the solutions to paths of various lengths, generated by the Python code in Section 6. Each array below represents a solution on a path.

```
n=1 path solutions
[0]

n=2 path solutions
[0, 1]
[1, 0]

n=3 path solutions
[0, 3, 1]
[1, 3, 0]
[2, 0, 3]
[3, 0, 2]

n=4 path solutions
[2, 5, 0, 3]
[3, 0, 5, 2]

n=5 path solutions
[0, 5, 10, 2, 7]
[2, 6, 10, 0, 4]
[2, 10, 5, 0, 8]
[3, 7, 0, 10, 4]
[3, 7, 0, 10, 5]
[3, 8, 0, 5, 10]
[4, 0, 10, 6, 2]
[4, 10, 0, 7, 3]
[5, 0, 10, 3, 7]
[5, 10, 0, 7, 3]
[6, 0, 10, 3, 7]
[6, 10, 0, 4, 8]
[7, 2, 10, 5, 0]
[7, 3, 10, 0, 5]
[7, 3, 10, 0, 6]
[8, 0, 5, 10, 2]
[8, 4, 0, 10, 6]
[10, 5, 0, 8, 3]

n=6 path solutions
[3, 8, 13, 0, 5, 10]
[10, 5, 0, 13, 8, 3]

n=7 path solutions
[3, 15, 8, 20, 0, 12, 5]
[4, 16, 9, 0, 20, 13, 5]
```

```
[4, 16, 9, 0, 20, 13, 6]
[4, 16, 10, 0, 20, 6, 13]
[4, 16, 10, 0, 20, 6, 14]
[5, 12, 0, 20, 8, 15, 3]
[5, 13, 20, 0, 9, 16, 4]
[6, 13, 20, 0, 9, 16, 4]
[6, 14, 0, 20, 10, 4, 16]
[6, 16, 0, 10, 20, 4, 14]
[7, 14, 0, 20, 10, 4, 16]
[13, 6, 20, 0, 10, 16, 4]
[14, 4, 20, 10, 0, 16, 6]
[14, 6, 20, 0, 10, 16, 4]
[14, 7, 0, 20, 11, 4, 16]
[15, 7, 0, 20, 11, 4, 16]
[15, 8, 20, 0, 12, 5, 17]
[16, 4, 10, 20, 0, 14, 6]
[16, 4, 10, 20, 0, 14, 7]
[16, 4, 11, 20, 0, 7, 14]
[16, 4, 11, 20, 0, 7, 15]
[17, 5, 12, 0, 20, 8, 15]

n=8 path solutions
[4, 11, 18, 25, 0, 7, 14, 21]
[4, 19, 10, 25, 0, 15, 6, 21]
[21, 6, 15, 0, 25, 10, 19, 4]
[21, 14, 7, 0, 25, 18, 11, 4]

n=9 path solutions
[4, 28, 19, 10, 34, 0, 24, 15, 6]
[5, 19, 28, 12, 0, 34, 23, 7, 16]
[5, 29, 13, 21, 0, 34, 8, 26, 16]
[5, 29, 13, 21, 0, 34, 8, 26, 17]
[5, 29, 13, 22, 0, 34, 8, 17, 26]
[5, 29, 13, 22, 0, 34, 8, 17, 27]
[5, 29, 20, 11, 0, 34, 25, 16, 6]
[5, 29, 20, 11, 0, 34, 25, 16, 7]
[5, 29, 20, 12, 0, 34, 25, 7, 16]
[5, 29, 20, 12, 0, 34, 25, 7, 17]
[6, 15, 24, 0, 34, 10, 19, 28, 4]
[6, 16, 25, 34, 0, 11, 20, 29, 5]
[7, 16, 25, 34, 0, 11, 20, 29, 5]
[7, 17, 26, 0, 34, 12, 21, 5, 29]
[7, 29, 20, 0, 12, 34, 25, 5, 17]
[8, 17, 26, 0, 34, 12, 21, 5, 29]
[16, 7, 23, 34, 0, 12, 28, 19, 5]
[16, 7, 25, 34, 0, 12, 20, 29, 5]
[16, 26, 8, 34, 0, 21, 13, 29, 5]
[17, 5, 25, 34, 12, 0, 20, 29, 7]
```

```
[17, 7, 25, 34, 0, 12, 20, 29, 5]
[17, 8, 26, 0, 34, 13, 21, 5, 29]
[17, 26, 8, 34, 0, 21, 13, 29, 5]
[17, 27, 9, 0, 34, 22, 14, 5, 29]
[17, 29, 9, 0, 22, 34, 14, 5, 27]
[18, 8, 26, 0, 34, 13, 21, 5, 29]
[18, 27, 9, 0, 34, 22, 14, 5, 29]
[18, 27, 11, 0, 34, 22, 6, 15, 29]
[26, 17, 8, 34, 0, 22, 13, 29, 5]
[27, 5, 14, 34, 22, 0, 9, 29, 17]
[27, 17, 8, 34, 0, 22, 13, 29, 5]
[27, 18, 9, 0, 34, 23, 14, 5, 29]
[28, 18, 9, 0, 34, 23, 14, 5, 29]
[28, 19, 10, 34, 0, 24, 15, 6, 30]
[29, 5, 14, 22, 34, 0, 9, 27, 17]
[29, 5, 14, 22, 34, 0, 9, 27, 18]
[29, 5, 14, 23, 34, 0, 9, 18, 27]
[29, 5, 14, 23, 34, 0, 9, 18, 28]
[29, 5, 21, 12, 34, 0, 26, 17, 7]
[29, 5, 21, 12, 34, 0, 26, 17, 8]
[29, 5, 21, 13, 34, 0, 26, 8, 17]
[29, 5, 21, 13, 34, 0, 26, 8, 18]
[29, 15, 6, 22, 34, 0, 11, 27, 18]
[30, 6, 15, 24, 0, 34, 10, 19, 28]

n=10 path solutions
[5, 14, 23, 32, 41, 0, 9, 18, 27, 36]
[5, 14, 33, 22, 41, 0, 9, 28, 17, 36]
[5, 22, 33, 12, 41, 0, 27, 18, 7, 36]
[5, 22, 33, 14, 41, 0, 27, 8, 19, 36]
[5, 24, 13, 32, 41, 0, 19, 8, 27, 36]
[5, 24, 33, 12, 41, 0, 19, 28, 7, 36]
[5, 34, 13, 22, 41, 0, 29, 8, 17, 36]
[5, 34, 23, 12, 41, 0, 29, 18, 7, 36]
[5, 34, 23, 14, 41, 0, 29, 8, 19, 36]
[36, 7, 18, 27, 0, 41, 12, 33, 22, 5]
[36, 7, 18, 29, 0, 41, 12, 23, 34, 5]
[36, 7, 28, 19, 0, 41, 12, 33, 24, 5]
[36, 17, 8, 29, 0, 41, 22, 13, 34, 5]
[36, 17, 28, 9, 0, 41, 22, 33, 14, 5]
[36, 19, 8, 27, 0, 41, 14, 33, 22, 5]
[36, 19, 8, 29, 0, 41, 14, 23, 34, 5]
[36, 27, 8, 19, 0, 41, 32, 13, 24, 5]
[36, 27, 18, 9, 0, 41, 32, 23, 14, 5]

n=11 path solutions
[5, 45, 22, 35, 12, 52, 0, 28, 41, 18, 7]
[5, 45, 34, 23, 12, 52, 0, 40, 29, 18, 7]
```

[6, 22, 33, 44, 14, 0, 52, 27, 38, 8, 19]
[6, 22, 45, 34, 14, 0, 52, 27, 40, 8, 19]
[6, 27, 45, 15, 35, 0, 52, 22, 9, 41, 30]
[6, 34, 45, 15, 26, 0, 52, 39, 9, 20, 31]
[6, 46, 16, 27, 37, 0, 52, 10, 21, 43, 31]
[6, 46, 16, 27, 37, 0, 52, 10, 21, 43, 32]
[6, 46, 16, 27, 38, 0, 52, 10, 21, 32, 43]
[6, 46, 16, 27, 38, 0, 52, 10, 21, 32, 44]
[6, 46, 16, 36, 25, 0, 52, 10, 42, 31, 19]
[6, 46, 16, 36, 25, 0, 52, 10, 42, 31, 20]
[6, 46, 16, 36, 26, 0, 52, 10, 42, 20, 31]
[6, 46, 16, 36, 26, 0, 52, 10, 42, 20, 32]
[6, 46, 23, 34, 14, 0, 52, 41, 28, 8, 19]
[6, 46, 23, 36, 13, 0, 52, 29, 42, 19, 7]
[6, 46, 23, 36, 13, 0, 52, 29, 42, 19, 8]
[6, 46, 23, 36, 14, 0, 52, 29, 42, 8, 19]
[6, 46, 23, 36, 14, 0, 52, 29, 42, 8, 20]
[6, 46, 28, 15, 37, 0, 52, 22, 9, 43, 31]
[6, 46, 28, 15, 37, 0, 52, 22, 9, 43, 32]
[6, 46, 28, 15, 38, 0, 52, 22, 9, 32, 43]
[6, 46, 28, 15, 38, 0, 52, 22, 9, 32, 44]
[6, 46, 28, 17, 37, 0, 52, 10, 23, 43, 32]
[6, 46, 35, 15, 25, 0, 52, 41, 9, 31, 19]
[6, 46, 35, 15, 25, 0, 52, 41, 9, 31, 20]
[6, 46, 35, 15, 26, 0, 52, 41, 9, 20, 31]
[6, 46, 35, 15, 26, 0, 52, 41, 9, 20, 32]
[6, 46, 35, 24, 13, 0, 52, 41, 30, 19, 7]
[6, 46, 35, 24, 13, 0, 52, 41, 30, 19, 8]
[6, 46, 35, 24, 14, 0, 52, 41, 30, 8, 19]
[6, 46, 35, 24, 14, 0, 52, 41, 30, 8, 20]
[7, 18, 29, 40, 0, 52, 12, 23, 34, 45, 5]
[7, 18, 41, 28, 0, 52, 12, 35, 22, 45, 5]
[7, 19, 30, 41, 52, 0, 13, 24, 35, 46, 6]
[7, 19, 42, 29, 52, 0, 13, 36, 23, 46, 6]
[8, 19, 30, 41, 52, 0, 13, 24, 35, 46, 6]
[8, 19, 42, 29, 52, 0, 13, 36, 23, 46, 6]
[8, 20, 31, 42, 0, 52, 14, 25, 36, 6, 46]
[8, 20, 43, 30, 0, 52, 14, 37, 24, 6, 46]
[8, 46, 23, 36, 0, 14, 52, 29, 42, 6, 20]
[8, 46, 35, 24, 0, 14, 52, 41, 30, 6, 20]
[9, 20, 31, 42, 0, 52, 14, 25, 36, 6, 46]
[9, 20, 43, 30, 0, 52, 14, 37, 24, 6, 46]
[19, 8, 28, 41, 52, 0, 14, 34, 23, 46, 6]
[19, 8, 30, 41, 52, 0, 14, 24, 35, 46, 6]
[19, 8, 38, 27, 52, 0, 14, 44, 33, 22, 6]
[19, 8, 40, 27, 52, 0, 14, 34, 45, 22, 6]
[19, 8, 42, 29, 52, 0, 14, 36, 23, 46, 6]
[19, 31, 9, 41, 52, 0, 25, 15, 35, 46, 6]

```
[19, 31, 42, 10, 52, 0, 25, 36, 16, 46, 6]
[20, 6, 30, 41, 52, 14, 0, 24, 35, 46, 8]
[20, 6, 42, 29, 52, 14, 0, 36, 23, 46, 8]
[20, 8, 30, 41, 52, 0, 14, 24, 35, 46, 6]
[20, 8, 42, 29, 52, 0, 14, 36, 23, 46, 6]
[20, 9, 29, 42, 0, 52, 15, 35, 24, 6, 46]
[20, 9, 31, 42, 0, 52, 15, 25, 36, 6, 46]
[20, 9, 43, 30, 0, 52, 15, 37, 24, 6, 46]
[20, 31, 9, 41, 52, 0, 25, 15, 35, 46, 6]
[20, 31, 42, 10, 52, 0, 25, 36, 16, 46, 6]
[20, 32, 10, 42, 0, 52, 26, 16, 36, 6, 46]
[20, 32, 43, 11, 0, 52, 26, 37, 17, 6, 46]
[20, 46, 10, 36, 0, 26, 52, 16, 42, 6, 32]
[20, 46, 35, 11, 0, 26, 52, 41, 17, 6, 32]
[21, 9, 31, 42, 0, 52, 15, 25, 36, 6, 46]
[21, 9, 43, 30, 0, 52, 15, 37, 24, 6, 46]
[21, 32, 10, 42, 0, 52, 26, 16, 36, 6, 46]
[21, 32, 43, 11, 0, 52, 26, 37, 17, 6, 46]
[21, 32, 43, 13, 0, 52, 26, 37, 7, 18, 46]
[22, 11, 43, 30, 0, 52, 17, 37, 7, 25, 46]
[30, 41, 9, 22, 52, 0, 35, 15, 45, 27, 6]
[31, 20, 9, 39, 52, 0, 26, 15, 45, 34, 6]
[31, 20, 9, 41, 52, 0, 26, 15, 35, 46, 6]
[31, 20, 42, 10, 52, 0, 26, 36, 16, 46, 6]
[31, 43, 9, 22, 52, 0, 37, 15, 28, 46, 6]
[31, 43, 21, 10, 52, 0, 37, 27, 16, 46, 6]
[32, 6, 17, 41, 52, 26, 0, 11, 35, 46, 20]
[32, 6, 42, 16, 52, 26, 0, 36, 10, 46, 20]
[32, 20, 9, 41, 52, 0, 26, 15, 35, 46, 6]
[32, 20, 42, 10, 52, 0, 26, 36, 16, 46, 6]
[32, 21, 10, 42, 0, 52, 27, 16, 36, 6, 46]
[32, 21, 43, 11, 0, 52, 27, 37, 17, 6, 46]
[32, 43, 9, 22, 52, 0, 37, 15, 28, 46, 6]
[32, 43, 21, 10, 52, 0, 37, 27, 16, 46, 6]
[32, 43, 23, 10, 52, 0, 37, 17, 28, 46, 6]
[32, 44, 10, 23, 0, 52, 38, 16, 29, 6, 46]
[32, 44, 22, 11, 0, 52, 38, 28, 17, 6, 46]
[32, 46, 10, 23, 0, 38, 52, 16, 29, 6, 44]
[32, 46, 22, 11, 0, 38, 52, 28, 17, 6, 44]
[33, 21, 10, 42, 0, 52, 27, 16, 36, 6, 46]
[33, 21, 43, 11, 0, 52, 27, 37, 17, 6, 46]
[33, 44, 10, 23, 0, 52, 38, 16, 29, 6, 46]
[33, 44, 12, 25, 0, 52, 38, 18, 7, 30, 46]
[33, 44, 14, 25, 0, 52, 38, 8, 19, 30, 46]
[33, 44, 22, 11, 0, 52, 38, 28, 17, 6, 46]
[33, 44, 24, 11, 0, 52, 38, 18, 29, 6, 46]
[43, 32, 9, 22, 52, 0, 38, 15, 28, 46, 6]
[43, 32, 21, 10, 52, 0, 38, 27, 16, 46, 6]
```

[44, 6, 17, 28, 52, 38, 0, 11, 22, 46, 32]
[44, 6, 29, 16, 52, 38, 0, 23, 10, 46, 32]
[44, 32, 9, 22, 52, 0, 38, 15, 28, 46, 6]
[44, 32, 21, 10, 52, 0, 38, 27, 16, 46, 6]
[44, 33, 10, 23, 0, 52, 39, 16, 29, 6, 46]
[44, 33, 22, 11, 0, 52, 39, 28, 17, 6, 46]
[45, 33, 10, 23, 0, 52, 39, 16, 29, 6, 46]
[45, 33, 22, 11, 0, 52, 39, 28, 17, 6, 46]
[45, 34, 11, 24, 52, 0, 40, 17, 30, 7, 47]
[45, 34, 23, 12, 52, 0, 40, 29, 18, 7, 47]
[46, 6, 17, 28, 38, 52, 0, 11, 22, 44, 32]
[46, 6, 17, 28, 38, 52, 0, 11, 22, 44, 33]
[46, 6, 17, 28, 39, 52, 0, 11, 22, 33, 44]
[46, 6, 17, 28, 39, 52, 0, 11, 22, 33, 45]
[46, 6, 17, 37, 26, 52, 0, 11, 43, 32, 20]
[46, 6, 17, 37, 26, 52, 0, 11, 43, 32, 21]
[46, 6, 17, 37, 27, 52, 0, 11, 43, 21, 32]
[46, 6, 17, 37, 27, 52, 0, 11, 43, 21, 33]
[46, 6, 24, 35, 15, 52, 0, 42, 29, 9, 20]
[46, 6, 24, 37, 14, 52, 0, 30, 43, 20, 8]
[46, 6, 24, 37, 14, 52, 0, 30, 43, 20, 9]
[46, 6, 24, 37, 15, 52, 0, 30, 43, 9, 20]
[46, 6, 24, 37, 15, 52, 0, 30, 43, 9, 21]
[46, 6, 29, 16, 38, 52, 0, 23, 10, 44, 32]
[46, 6, 29, 16, 38, 52, 0, 23, 10, 44, 33]
[46, 6, 29, 16, 39, 52, 0, 23, 10, 33, 44]
[46, 6, 29, 16, 39, 52, 0, 23, 10, 33, 45]
[46, 6, 29, 18, 38, 52, 0, 11, 24, 44, 33]
[46, 6, 36, 16, 26, 52, 0, 42, 10, 32, 20]
[46, 6, 36, 16, 26, 52, 0, 42, 10, 32, 21]
[46, 6, 36, 16, 27, 52, 0, 42, 10, 21, 32]
[46, 6, 36, 16, 27, 52, 0, 42, 10, 21, 33]
[46, 6, 36, 25, 14, 52, 0, 42, 31, 20, 8]
[46, 6, 36, 25, 14, 52, 0, 42, 31, 20, 9]
[46, 6, 36, 25, 15, 52, 0, 42, 31, 9, 20]
[46, 6, 36, 25, 15, 52, 0, 42, 31, 9, 21]
[46, 18, 7, 37, 26, 52, 0, 13, 43, 32, 21]
[46, 25, 7, 37, 17, 52, 0, 30, 43, 11, 22]
[46, 30, 7, 18, 38, 52, 0, 25, 12, 44, 33]
[46, 30, 19, 8, 38, 52, 0, 25, 14, 44, 33]
[47, 7, 18, 29, 40, 0, 52, 12, 23, 34, 45]
[47, 7, 30, 17, 40, 0, 52, 24, 11, 34, 45]

Appendix B: Lollipop Solutions

These are all the solutions to lollipops of various lengths, generated by the Python code in Section 6. Each array below represents a solution on a lollipop. Read the solution the same way as you would a path, except make the last two vertices part of the "lollipop."

```
4 vertices --> 4
[1, 4, 0, 2]
[1, 4, 2, 0]
[3, 0, 2, 4]
[3, 0, 4, 2]

5 vertices --> 7
[2, 5, 0, 3, 7]
[2, 5, 0, 7, 3]
[5, 2, 7, 0, 4]
[5, 2, 7, 4, 0]

6 vertices --> 11
[2, 6, 11, 0, 4, 8]
[2, 6, 11, 0, 8, 4]
[9, 5, 0, 11, 3, 7]
[9, 5, 0, 11, 7, 3]

7 vertices --> 15
[3, 12, 7, 0, 15, 4, 10]
[3, 12, 7, 0, 15, 10, 4]
[12, 3, 8, 15, 0, 5, 11]
[12, 3, 8, 15, 0, 11, 5]

8 vertices --> 22
[3, 9, 15, 22, 0, 6, 12, 18]
[3, 9, 15, 22, 0, 6, 18, 12]
[3, 14, 8, 22, 0, 17, 5, 11]
[3, 14, 8, 22, 0, 17, 11, 5]
[3, 15, 8, 22, 0, 12, 5, 18]
[3, 15, 8, 22, 0, 12, 18, 5]
[3, 16, 8, 22, 0, 12, 5, 18]
[3, 16, 8, 22, 0, 12, 18, 5]
[9, 16, 4, 22, 12, 0, 7, 18]
[9, 16, 4, 22, 12, 0, 18, 7]
[10, 3, 15, 22, 7, 0, 12, 18]
[10, 3, 15, 22, 7, 0, 18, 12]
[12, 19, 7, 0, 15, 22, 4, 10]
[12, 19, 7, 0, 15, 22, 10, 4]
[13, 6, 18, 0, 10, 22, 4, 15]
[13, 6, 18, 0, 10, 22, 15, 4]
[19, 6, 14, 0, 22, 10, 4, 17]
[19, 6, 14, 0, 22, 10, 17, 4]
```

```
[19, 7, 14, 0, 22, 10, 4, 17]
[19, 7, 14, 0, 22, 10, 17, 4]
[19, 8, 14, 0, 22, 5, 11, 17]
[19, 8, 14, 0, 22, 5, 17, 11]
[19, 13, 7, 0, 22, 16, 4, 10]
[19, 13, 7, 0, 22, 16, 10, 4]

9 vertices --> 27
[4, 23, 10, 17, 0, 27, 6, 13, 21]
[4, 23, 10, 17, 0, 27, 6, 21, 13]
[4, 23, 16, 9, 0, 27, 20, 5, 13]
[4, 23, 16, 9, 0, 27, 20, 13, 5]
[23, 4, 11, 18, 27, 0, 7, 14, 22]
[23, 4, 11, 18, 27, 0, 7, 22, 14]
[23, 4, 17, 10, 27, 0, 21, 6, 14]
[23, 4, 17, 10, 27, 0, 21, 14, 6]

10 vertices --> 36
[4, 17, 26, 10, 36, 0, 21, 30, 6, 14]
[4, 17, 26, 10, 36, 0, 21, 30, 14, 6]
[4, 18, 28, 10, 36, 0, 23, 15, 6, 31]
[4, 18, 28, 10, 36, 0, 23, 15, 31, 6]
[4, 19, 28, 10, 36, 0, 23, 15, 6, 31]
[4, 19, 28, 10, 36, 0, 23, 15, 31, 6]
[4, 19, 28, 12, 36, 0, 23, 7, 16, 31]
[4, 19, 28, 12, 36, 0, 23, 7, 31, 16]
[4, 20, 11, 27, 36, 0, 16, 7, 23, 31]
[4, 20, 11, 27, 36, 0, 16, 7, 31, 23]
[4, 28, 19, 10, 36, 0, 24, 15, 6, 31]
[4, 28, 19, 10, 36, 0, 24, 15, 31, 6]
[4, 29, 19, 10, 36, 0, 24, 15, 6, 31]
[4, 29, 19, 10, 36, 0, 24, 15, 31, 6]
[5, 19, 28, 12, 36, 0, 23, 7, 16, 31]
[5, 19, 28, 12, 36, 0, 23, 7, 31, 16]
[16, 7, 31, 22, 0, 12, 36, 27, 5, 18]
[16, 7, 31, 22, 0, 12, 36, 27, 18, 5]
[20, 29, 5, 14, 36, 24, 0, 9, 18, 31]
[20, 29, 5, 14, 36, 24, 0, 9, 31, 18]
[31, 17, 8, 24, 0, 36, 13, 29, 5, 20]
[31, 17, 8, 24, 0, 36, 13, 29, 20, 5]
[32, 7, 17, 26, 0, 36, 12, 21, 5, 30]
[32, 7, 17, 26, 0, 36, 12, 21, 30, 5]
[32, 8, 17, 26, 0, 36, 12, 21, 5, 30]
[32, 8, 17, 26, 0, 36, 12, 21, 30, 5]
[32, 16, 25, 9, 0, 36, 20, 29, 5, 13]
[32, 16, 25, 9, 0, 36, 20, 29, 13, 5]
[32, 17, 8, 24, 0, 36, 13, 29, 5, 20]
[32, 17, 8, 24, 0, 36, 13, 29, 20, 5]
```

```
[32, 17, 8, 26, 0, 36, 13, 21, 5, 30]
[32, 17, 8, 26, 0, 36, 13, 21, 30, 5]
[32, 18, 8, 26, 0, 36, 13, 21, 5, 30]
[32, 18, 8, 26, 0, 36, 13, 21, 30, 5]
[32, 19, 10, 26, 0, 36, 15, 6, 22, 30]
[32, 19, 10, 26, 0, 36, 15, 6, 30, 22]

11 vertices --> 43
[5, 38, 13, 22, 31, 0, 43, 8, 17, 26, 36]
[5, 38, 13, 22, 31, 0, 43, 8, 17, 36, 26]
[5, 38, 13, 30, 21, 0, 43, 8, 35, 16, 26]
[5, 38, 13, 30, 21, 0, 43, 8, 35, 26, 16]
[5, 38, 19, 30, 11, 0, 43, 24, 35, 6, 16]
[5, 38, 19, 30, 11, 0, 43, 24, 35, 16, 6]
[5, 38, 23, 12, 31, 0, 43, 18, 7, 26, 36]
[5, 38, 23, 12, 31, 0, 43, 18, 7, 36, 26]
[5, 38, 29, 12, 21, 0, 43, 34, 7, 16, 26]
[5, 38, 29, 12, 21, 0, 43, 34, 7, 26, 16]
[5, 38, 29, 20, 11, 0, 43, 34, 25, 6, 16]
[5, 38, 29, 20, 11, 0, 43, 34, 25, 16, 6]
[38, 5, 14, 23, 32, 43, 0, 9, 18, 27, 37]
[38, 5, 14, 23, 32, 43, 0, 9, 18, 37, 27]
[38, 5, 14, 31, 22, 43, 0, 9, 36, 17, 27]
[38, 5, 14, 31, 22, 43, 0, 9, 36, 27, 17]
[38, 5, 20, 31, 12, 43, 0, 25, 36, 7, 17]
[38, 5, 20, 31, 12, 43, 0, 25, 36, 17, 7]
[38, 5, 24, 13, 32, 43, 0, 19, 8, 27, 37]
[38, 5, 24, 13, 32, 43, 0, 19, 8, 37, 27]
[38, 5, 30, 13, 22, 43, 0, 35, 8, 17, 27]
[38, 5, 30, 13, 22, 43, 0, 35, 8, 27, 17]
[38, 5, 30, 21, 12, 43, 0, 35, 26, 7, 17]
[38, 5, 30, 21, 12, 43, 0, 35, 26, 17, 7]

12 vertices --> 54
[5, 20, 31, 42, 12, 54, 0, 25, 36, 47, 7, 17]
[5, 20, 31, 42, 12, 54, 0, 25, 36, 47, 17, 7]
[5, 21, 33, 44, 12, 54, 0, 27, 38, 18, 7, 48]
[5, 21, 33, 44, 12, 54, 0, 27, 38, 18, 48, 7]
[5, 22, 33, 44, 12, 54, 0, 27, 38, 18, 7, 48]
[5, 22, 33, 44, 12, 54, 0, 27, 38, 18, 48, 7]
[5, 22, 33, 44, 14, 54, 0, 27, 38, 8, 19, 48]
[5, 22, 33, 44, 14, 54, 0, 27, 38, 8, 48, 19]
[5, 33, 22, 44, 12, 54, 0, 28, 38, 18, 7, 48]
[5, 33, 22, 44, 12, 54, 0, 28, 38, 18, 48, 7]
[5, 33, 45, 23, 12, 54, 0, 39, 29, 18, 7, 48]
[5, 33, 45, 23, 12, 54, 0, 39, 29, 18, 48, 7]
[5, 34, 22, 44, 12, 54, 0, 28, 38, 18, 7, 48]
[5, 34, 22, 44, 12, 54, 0, 28, 38, 18, 48, 7]
```

[5, 34, 45, 13, 26, 54, 0, 39, 19, 8, 31, 48]
[5, 34, 45, 13, 26, 54, 0, 39, 19, 8, 48, 31]
[5, 34, 45, 15, 26, 54, 0, 39, 9, 20, 31, 48]
[5, 34, 45, 15, 26, 54, 0, 39, 9, 20, 48, 31]
[5, 34, 45, 23, 12, 54, 0, 39, 29, 18, 7, 48]
[5, 34, 45, 23, 12, 54, 0, 39, 29, 18, 48, 7]
[5, 34, 45, 25, 12, 54, 0, 39, 19, 30, 7, 48]
[5, 34, 45, 25, 12, 54, 0, 39, 19, 30, 48, 7]
[5, 35, 24, 13, 43, 54, 0, 30, 19, 8, 38, 48]
[5, 35, 24, 13, 43, 54, 0, 30, 19, 8, 48, 38]
[5, 45, 34, 23, 12, 54, 0, 40, 29, 18, 7, 48]
[5, 45, 34, 23, 12, 54, 0, 40, 29, 18, 48, 7]
[5, 46, 34, 23, 12, 54, 0, 40, 29, 18, 7, 48]
[5, 46, 34, 23, 12, 54, 0, 40, 29, 18, 48, 7]
[6, 22, 33, 44, 14, 54, 0, 27, 38, 8, 19, 48]
[6, 22, 33, 44, 14, 54, 0, 27, 38, 8, 48, 19]
[6, 34, 45, 15, 26, 54, 0, 39, 9, 20, 31, 48]
[6, 34, 45, 15, 26, 54, 0, 39, 9, 20, 48, 31]
[19, 8, 48, 37, 26, 0, 14, 54, 43, 32, 6, 21]
[19, 8, 48, 37, 26, 0, 14, 54, 43, 32, 21, 6]
[23, 34, 6, 44, 16, 54, 28, 0, 38, 10, 21, 48]
[23, 34, 6, 44, 16, 54, 28, 0, 38, 10, 48, 21]
[31, 20, 48, 10, 38, 0, 26, 54, 16, 44, 6, 33]
[31, 20, 48, 10, 38, 0, 26, 54, 16, 44, 33, 6]
[35, 46, 6, 17, 28, 54, 40, 0, 11, 22, 33, 48]
[35, 46, 6, 17, 28, 54, 40, 0, 11, 22, 48, 33]
[48, 20, 9, 39, 28, 0, 54, 15, 45, 34, 6, 23]
[48, 20, 9, 39, 28, 0, 54, 15, 45, 34, 23, 6]
[48, 32, 21, 10, 40, 0, 54, 27, 16, 46, 6, 35]
[48, 32, 21, 10, 40, 0, 54, 27, 16, 46, 35, 6]
[49, 8, 20, 31, 42, 0, 54, 14, 25, 36, 6, 47]
[49, 8, 20, 31, 42, 0, 54, 14, 25, 36, 47, 6]
[49, 9, 20, 31, 42, 0, 54, 14, 25, 36, 6, 47]
[49, 9, 20, 31, 42, 0, 54, 14, 25, 36, 47, 6]
[49, 19, 30, 41, 11, 0, 54, 24, 35, 46, 6, 16]
[49, 19, 30, 41, 11, 0, 54, 24, 35, 46, 16, 6]
[49, 20, 9, 29, 42, 0, 54, 15, 35, 24, 6, 47]
[49, 20, 9, 29, 42, 0, 54, 15, 35, 24, 47, 6]
[49, 20, 9, 31, 42, 0, 54, 15, 25, 36, 6, 47]
[49, 20, 9, 31, 42, 0, 54, 15, 25, 36, 47, 6]
[49, 20, 9, 39, 28, 0, 54, 15, 45, 34, 6, 23]
[49, 20, 9, 39, 28, 0, 54, 15, 45, 34, 23, 6]
[49, 20, 9, 41, 28, 0, 54, 15, 35, 46, 6, 23]
[49, 20, 9, 41, 28, 0, 54, 15, 35, 46, 23, 6]
[49, 20, 32, 10, 42, 0, 54, 26, 16, 36, 6, 47]
[49, 20, 32, 10, 42, 0, 54, 26, 16, 36, 47, 6]
[49, 21, 9, 31, 42, 0, 54, 15, 25, 36, 6, 47]
[49, 21, 9, 31, 42, 0, 54, 15, 25, 36, 47, 6]

```
[49, 21, 32, 10, 42, 0, 54, 26, 16, 36, 6, 47]
[49, 21, 32, 10, 42, 0, 54, 26, 16, 36, 47, 6]
[49, 32, 21, 10, 40, 0, 54, 27, 16, 46, 6, 35]
[49, 32, 21, 10, 40, 0, 54, 27, 16, 46, 35, 6]
[49, 32, 21, 10, 42, 0, 54, 27, 16, 36, 6, 47]
[49, 32, 21, 10, 42, 0, 54, 27, 16, 36, 47, 6]
[49, 33, 21, 10, 42, 0, 54, 27, 16, 36, 6, 47]
[49, 33, 21, 10, 42, 0, 54, 27, 16, 36, 47, 6]
[49, 34, 23, 12, 42, 0, 54, 29, 18, 7, 37, 47]
[49, 34, 23, 12, 42, 0, 54, 29, 18, 7, 47, 37]
```

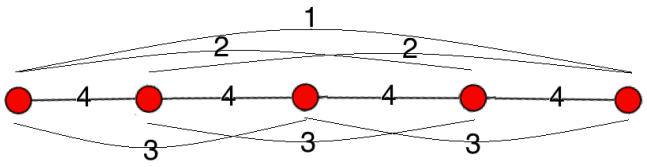
Appendix C: Another Method

There are two ways we can look at maximum hopping distance and related ideas: the first is to look for a way to maximize the distance. Another is to find a way to ensure that $F(x_i)$ increases by the smallest value possible.

We have mainly been concerned with the first approach throughout this paper. However, I have toyed quite a bit with the second approach, and have an alternate explanation for the solutions to $rn(P_{2k+1})$. Instead of trying to maximize the hopping distance, I used the hopping distance as a way to account for increases in the $F(x_{2k+1})$. Thus, I was considering a tight hopping scenario where each increase of $F(x_i)$ contributed to an increase in $F(x_{2k+1})$ (ie. $F(x_{2k+1})$ is a partition into the minimum increases.)

Hop the Graph!

It remains to be proved whether this experimental method works. The idea is to start on a vertex, and to hop through all the vertices, increasing the value of the label with each hop. The last vertex hopped to will the maximum vertex. The logic works as follows: we assign a price to every hop. (See below).



This prices assigned above represent the smallest amount that $F(x_i)$ must increase when traveling from one vertex to another. We see that traveling the longest distance has the lowest price etc. Since we have n vertices, we know that we start on a vertex and must hop $n-1$ times. Our goal is to minimize the "hopping price" when hopping. We also know that there are exactly m hoppings with price m, for $1 \leq m \leq D$. It turns out that for odd paths P_{2k+1} with $k \geq 3$, choosing $k+1$ segments of value $k+1$ and $k-1$ segments that have an average of $k-1$ indeed minimizes $F(x_{2k+1})$. Similar observations hold for even paths. This is why, in Section 4 I decided to write $rn(G)$ as $(k+1)^2 + (k-1)^2$ and $k^2 + (k-1)^2$, since it made sense with this explanation. Obviously, there are more simple and compact ways to write the two solutions, but this way makes sense due to this method.