HEURISTIC ALGORITHMS FOR BROADCASTING IN CACTUS GRAPHS

Neil Conlan

A thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

April 2017

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Neil Conlan**

Entitled: **Heuristic Algorithms for Broadcasting in Cactus Graphs**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. O Ormandjieva	Chair
Dr. T.Fevens	Examiner
Dr. D Goswami	Examiner
Dr. H. Harutyunyan	Supervisor
Dr. E. Maraachlian	Co-supervisor

Approved ______________________________

Chair of Department or Graduate Program Director

______________ 20 ______ ______________________________

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

Abstract

Heuristic Algorithms for Broadcasting in Cactus Graphs

Neil Conlan

Broadcasting is an information dissemination problem in a connected network, in which one node, called the originator, disseminates a message to all other nodes by placing a series of calls along the communication lines of the network. Once informed, the nodes aid the originator in distributing the message. Finding the broadcast time of a vertex in an arbitrary graph is NP-complete. The problem is solved polynomially only for a few classes of graphs. In this thesis, we study the broadcast problem in a class of graph called a Cactus Graph. A cactus graph is a connected graph in which any two simple cycles have at most one vertex in common. Equivalently, it is a connected graph in which every edge belongs to at most one simple cycle. We review broadcasting on subclasses of cactus graphs such as, the unicyclic graphs, necklace graphs, k-cycle graphs, 2-restricted cactus graphs and k-restricted cactus graphs. We then provide four heuristic algorithms that solves broadcasting on a k-cycle graph. A k-cycle graph is a collection of k cycles of arbitrary lengths all connected to a central vertex. Finally, we run simulations of these heuristic algorithms on different sized k-cycle graphs to compare and discuss the results.

Acknowledgments

I would like that thank my supervisor Hovhanness Harutyunyan who helped, guided, and motivated me since I took one of his classes as an undergraduate student and for accepting me as one of his graduate students to research broadcasting. I would also like to thank Edward Marashlian who took time away from his life to come and help with my research.

Contents

List of Figures **viii**

List of Tables **x**

List of Algorithms **xi**

1 Introduction **1**

1.1 Broadcasting 2

1.2 Review of Commonly Used Topologies 4

1.2.1 Path P_n 5

1.2.2 Cycle C_n 5

1.2.3 Tree T 5

1.2.4 Complete Graph K_n 6

1.2.5 Hypercube H_n 7

1.2.6 2d Grid Network $G_{m,n}$ 7

1.2.7 d-Torus Graph 8

1.2.8 DeBruijn Graph DB_m 8

1.2.9 The Shuffle-Exchange SE_m 8

1.2.10 The Butterfly BF_m 9

1.2.11 The Cube-Connected Cycles CCC_m 9

1.2.12 Other Topologies 10

1.3 Thesis Contribution 10

2 Cactus Graphs **12**

2.1 Model Definition 12

2.2 Broadcasting in Cactus Graphs 14

2.2.1 Unicyclic Graphs 14

2.2.2 Necklace Graphs 14

2.2.3 k-Cycle Graphs 15

2.2.4 2-Restricted Cactus Graphs 16

2.2.5 k-Restricted Cactus Graphs 16

3 Heuristic Algorithms for Cactus Graph Broadcasting **18**

3.1 Lower Bounds on Broadcast Time for k-cycle Graph 20

3.1.1 Lower Bounds When Originator is the Central Vertex 21

3.1.2 Lower Bounds when Originator is Not the Central Vertex 23

3.2 Algorithm BroadcastBucket 24

3.3 Algorithm CycleBroadcastTimeHelper 31

3.4 Algorithm BroadcastGreedyMinMax 33

3.5 Algorithm BroadcastGreedyEven 34

3.6 Algorithm BroadcastRandom 35

3.7 Algorithm SCycle 36

4 Heuristic Algorithm Implementation Details and Results **39**

4.1 Implementation Details 39

4.2 Simulation Results 40

4.2.1 BroadcastBucket Results 43

4.2.2 BroadcastGreedyMinMax Results 45

4.2.3 BroadcastGreedyEven Results 46

4.2.4 BroadcastRandom Results 48

5 Conclusions and Future Work **50**

List of Figures

1 Broadcast Tree 4
2 Path 5
3 Cycle 5
4 Tree 6
5 Complete Graph 6
6 Hypercube 7
7 Grid 7
8 2-Torus graph with 12 vertices 8
9 DeBruijn graph DB3 9
10 Shuffle-Exchange graph SE_3 10
11 Butterfly graph BF_3 11
12 Cube-Connected Cycle CCC_3 11
13 Cactus Graph 13
14 Unicyclic Graph 14
15 Necklace Graph 15
16 k-cycle Graph (where $k = 4$) 15
17 2-Restricted Cactus Graph 16
18 k-restricted Cactus Graph (where $k = 3$) 17
19 k-cycle graph to use for Algorithms 20

20 Cycle Broadcast 25

21 Example bucket edge assignments 26

22 Showing that $BroadcastGuess/BroadcastBucket$ is not optimal 30

23 Bucket assignments from example 8 7 3 30

24 Optimal bucket assignment when $l_i = l_{i+1}$ for all $1 \leq i \leq k$ 31

25 Cycle Broadcast Figure 32

List of Tables

1 Heuristic Algorithm Simulation Results 40

List of Algorithms

1 iskcyclegraph: Procedure to determine if a given graph G is a k-cycle graph. Returns a list with the size of the cycles of the k-cycle graph or **false** if the given graph is not a k-cycle graph. 19

2 BroadcastBucket: This procedure will determine if broadcasting on a given k-cycle graph G_k from a central vertex u is possible in a given time $broadTime$. 28

3 BroadcastGuess: This procedure will generate possible broadcast times of a k-cycle graph G_k by performing a binary search between the theoretical minimum and maximum possible broadcast times. It will call *BroadcastBucket* to determine if broadcasting is possible in the given time. 29

4 CycleBroadcastTimeHelper: This procedure will determine the end broadcast time of a given cycle depending on which rounds we visit the first and second edges of the cycle. 32

5 BroadcastGreedyMinMax: This procedure determines the broadcast time of a given k-cycle graph G_k. It uses a greedy approach when assigning the first and second edges to each cycle C_i of the graph G_k. 33

6 BroadcastGreedyEven: This procedure determines the broadcast time of a given k-cycle graph G_k. It uses a greedy approach when assigning the first and second edges to each cycle C_i of the graph G_k. 34

7 BroadcastRandom: This procedure will determine the broadcast time of a k-cycle graph by randomly assigning the rounds to visit each edge of each cycle. 35

8 SCycle: This procedure will determine the broadcast time of a k-cycle graph by adding cycles into sets and moving them to different sets when the cycle is broadcasted on. 37

Chapter 1

Introduction

Since the early days of computing, computers have been used to solve certain problems in speeds that are unattainable by humans. Initially digital computers where designed using a single processor. While software executing using a single processor was great at solving problems in a sequential way, the need for higher speeds was needed.

Multi-computer and multi-processor systems (often called distributed computing) was a solution to this speed problem. While designing software to run on a distributed system introduced more complexity to the software, thus making it more difficult to design software, the speedup of solving problems this way was substantial enough to accept the increase in complexity.

Distributed computing works by breaking down a large problem that's needs to be solved into smaller independent problems which can be solved in parallel and then merging the results to obtain the final solution to the large problem. In some cases, the processors of the distributed system need to share information with each other. This can be accomplished using shared memory that all processors have access to or each processor can have its own local memory (distributed memory). Shared memory systems have limitations on the number of processors that can be connected, which makes it not practical for very large problems that require a large amount of processors.

Solving problems using distributed memory systems have some advantages over the shared memory model. One advantage is that there is theoretically no limitation on the number of processors that can be used to solve a problem. Another advantage is that each processor has its own memory pool which can be used to fit the data of the smaller problem it is tasked to solve fully inside memory. Since each processor has its own memory the total amount of memory can be larger than the shared memory model.

Processors of the interconnected network often have to share information with each other. This is accomplished by sending data over the network. It turns out that not only the power of the individual processors is important to solve a given problem but also the speed at which processors can disseminate information over the network. In recent years a lot of work has gone into studying properties of interconnected networks in order to find the best network structures for communication between processors of a network.

1.1 Broadcasting

There are different types of communication primitives a network can use when data needs disseminated to other processors. These communication primitives are:

- **Routing** or one-to-one communication.
- **Broadcasting** or one-to-all communication.
- **Multicasting** or one-to-many communication.
- **Gossiping** or all-to-all communication.

One of the most fundamental and interesting dissemination problems is *broadcasting*. The study of *broadcasting* was introduced by Slater, Cockayne and Hedetniemi in 1977 [24]. This problem has also been studied a lot in survey articles [14], [26] and a relatively recent book dedicated to information dissemination in networks [27].

Broadcasting is when one node of a network, called the *originator*, has data that it wants to share with all other nodes of that network. This is accomplished by placing a series of calls over the communication lines of the given network. Once a node is informed, the informed node can help the originator in distributing the data. These calls are assumed be be performed in discrete time units. The broadcasting of this data should be finished as quickly as possible, subject to the following constraints:

- Each call involves one of the informed nodes with one of its uninformed neighboring nodes.
- Each call requires one unit of time.
- A node can only participate in one call per unit of time.
- In one unit of time each informed node can work in parallel.

Formally, any network can be modelled as a connected graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges (or communication lines) between the vertices of the graph G. Two vertices $u, v \in V$ are said to be *adjacent* (or neighbors) if there is an edge $e \in E$, such that $e = (u, v)$. The *degree* of a vertex u, $d(u)$ or $deg(u)$, is defined as the number of incident vertices of the vertex u. The maximum degree of a graph G, denoted by $\Delta(G)$, and the minimum degree of a graph, denoted by $\delta(G)$, are the maximum and minimum degree of its vertices. The shortest path between a vertex u and a vertex v is called the *distance* between u and v, and is denoted by $dist(u, v)$. The *diameter* of a graph G is the maximum distance between two vertices of the graph, $max\{dist(u, v) \mid u \in V, v \in V)\}$.

A *broadcast scheme* of a graph with originating vertex u is defined as the set of calls performed to complete the broadcasting in the network. The broadcast time from originating vertex u, $b(u, G)$ or just $b(u)$, is the minimum number of time units (or rounds) required to complete the broadcast from vertex u. From any originating vertex u it is clear that the minimum number of rounds required to complete the broadcast is $b(u) \geq \lceil \log n \rceil$ since at best the number of informed nodes each round can double. The maximum number of rounds required to broadcast on a network is $b(u) = n - 1$ since in the worst case there will be only one newly informed vertex. The broadcast time of a graph G, is defined as $max\{b(u) \mid u \in V\}$. For any connected graph G we can represent a broadcast scheme from a vertex u as a spanning tree. Figure 1 shows a broadcast schemes spanning tree that ends in 5 rounds.

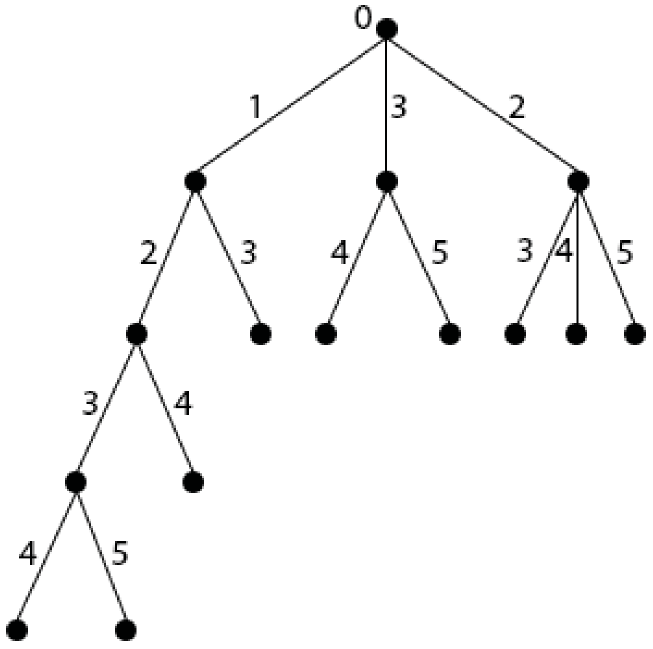

Figure 1: Broadcast Tree

Determining $b(u, G)$ for a vertex u of an arbitrary graph G is *NP*-Complete [28]. The proof of this is presented in [40]. Therefore there have been a lot of research into finding approximation algorithms or heuristic algorithms to determine the broadcast time of a vertex u in G, $b(u, G)$. ([1], [8], [7], [9], [12], [13], [33], [39]).

Since *broadcasting* in an arbitrary graph is *NP*-Complete a lot of research has gone into studying certain classes of graphs to design polynomial time algorithms that solves $b(u, G)$. One of the first graphs to be shown to have a linear time solution $O(|V|)$ was the tree [40].

1.2 Review of Commonly Used Topologies

In this section, we will review some commonly used topologies and give their broadcast times.

1.2.1 Path P_n

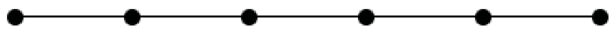

Figure 2: Path

A path P_n with length n is a sequence of vertices such that each vertex is connected to the next by an edge. For n vertices numbered v_1 to v_n there exists a total of $n-1$ edges in P_n. The broadcast time of $b(P_n)$ is $n-1$ because the maximum broadcast time for P_n is when the originating vertex u is one of the end vertices, v_1 or v_n. In Figure 2, $b(P_6) = 5$.

1.2.2 Cycle C_n

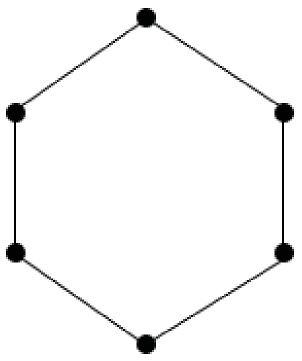

Figure 3: Cycle

A cycle C_n with n vertices is a path P_n where the first vertex v_1 and the end vertex v_n are connected by an edge $(v_1, v_n) \in E$. The broadcast time of a cycle is $b(C_n) = \lceil \frac{n}{2} \rceil$. In Figure 3, $b(C_6) = 3$.

1.2.3 Tree T

The tree T_n is a connected graph with n vertices and $n-1$ edges. Trees have a good property such that there is exactly one path between any two vertices. The broadcast time of a tree $b(T_n)$ has been shown to have a linear time $O(|V|)$ [40]. In Figure 4, $b(T_{13}) = 5$.

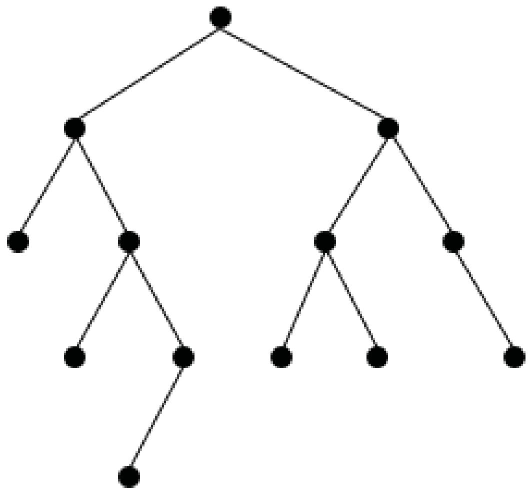

Figure 4: Tree

1.2.4 Complete Graph K_n

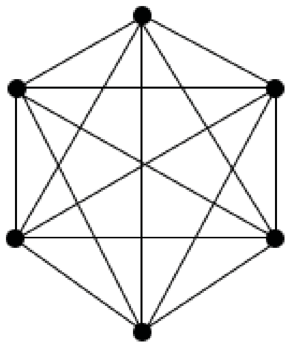

Figure 5: Complete Graph

The complete graph K_n with n vertices is a connected graph such that each vertex has an edge to each of the other vertices of the graph. This means that the number of edges in k_n is $\frac{n(n-1)}{2}$. The broadcast time $b(K_n) = \lceil \log n \rceil$ because at every round, except the last round, the number of informed vertices can double. In Figure 5, $b(K_6) = 3$.

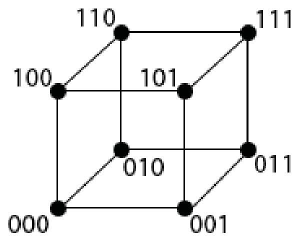

Figure 6: Hypercube

1.2.5 Hypercube H_n

The n-dimensional hypercube, H_n, is defined to be a graph on 2^n vertices. Each vertex is represented with a n-bit binary string, and two vertices are linked with an edge if and only if their binary strings differ in precisely one bit. For example, the vertices v_1 and v_5 in H_3 are neighbors because their binary representations 001 and 101 differ only in the third position. The hypercube is one of the few infinite family of graphs where the broadcast time is equal to *log n*, i.e. $b(H_n) = n$. Figure 6, $b(H_3) = 3$.

1.2.6 2d Grid Network $G_{m,n}$

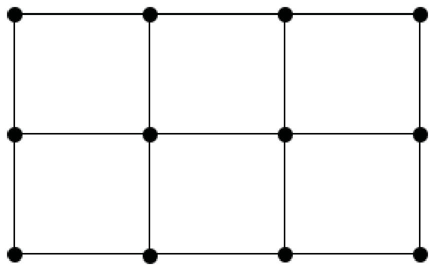

Figure 7: Grid

The 2 dimensional grid network $G_{m,n}$ (or mesh) is a graph with mn vertices. Each vertex is represented as a tuple (i, j) and can be connected to a maximum of 4 vertices denoted by $(i-1, j)$, $(i, j-1)$, $(i+1, j)$, $(i+1, j+1)$ for $1 < i < m$ and $1 < j < n$. The corner vertices are connected to 2 neighboring vertices, for example $(0, 0)$ is connected to $(0, 1)$, $(1, 0)$. The side vertices which are not corner vertices are connected to 3 neighboring vertices, for example $(0, j)$ is connected to $(0, j-1)$,

$(0, j+1)$, $(1, j)$. The broadcast time has been shown to be $b(G_{m,n}) = m + n - 2$ [24]. Figure 7, $b(G_{4,3}) = 5$.

1.2.7 d-Torus Graph

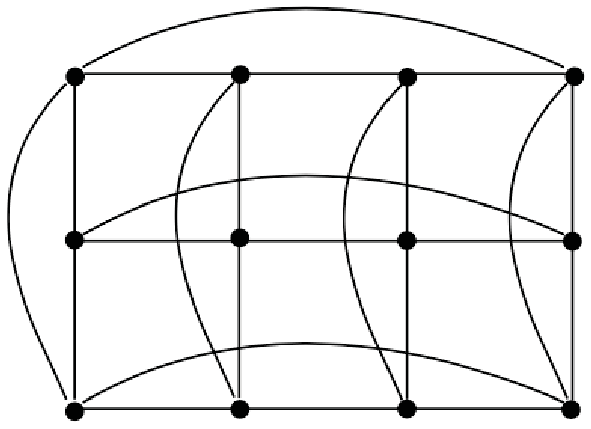

Figure 8: 2-Torus graph with 12 vertices

A d-Torus graph is a d-grid graph with both ends of rows and columns connected. Figure 8 shows a 2-Torus graph.

1.2.8 DeBruijn Graph DB_m

The DB_m is a graph where the vertices are represented by binary strings of length m and whose edges connect each string αa, where α is a binary string of length $m-1$ and a is in $\{0,1\}$, with the string αb, where b is a symbol in $\{0,1\}$. Figure 9 shows a 3-dimensional DeBruijn graph.

1.2.9 The Shuffle-Exchange SE_m

The SE_m is a graph where the vertices are represented by binary strings of length m and whose edges connect each string αa, where α is a binary string of length $m-1$ and a is in $\{0,1\}$, with the string αc and with the string αa, where c is the binary complement of a. Figure 10 shows a 3-dimensional shuffle-exchange graph.

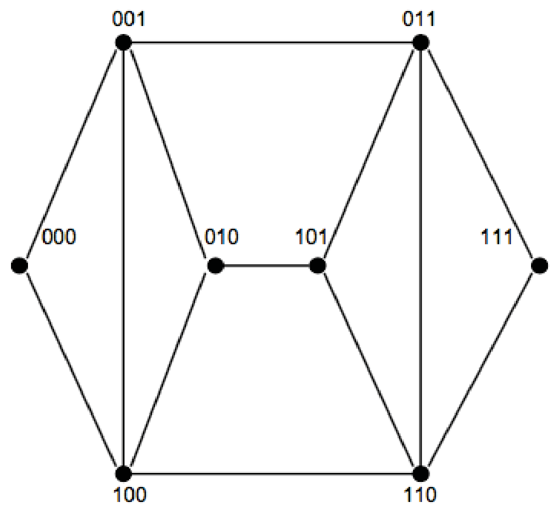

Figure 9: DeBruijn graph DB3

1.2.10 The Butterfly BF_m

The m-dimensional butterfly network, BF_m is a graph with vertex-set $V_m = \{0, 1, ..., m-1\}$ x $\{0,1\}^m$, where $\{0,1\}^m$ denotes a set of binary strings of length m. For each vertex $v = (i, j) \in V_m$, $i \in \{0, 1, ..., m-1\}$, $j \in \{0,1\}^m$, i is the level and j the position within level of v. There are two types of edges in BF_m : for each $i \in \{0, 1, ..., m-1\}$ and each $j = a_0a_1...a_{m-1} \in \{0,1\}^m$, the vertex (i, j) on level i is connected by a straight-edge with vertex $((i+1) \bmod m, j)$ and by a cross-edge with vertex $((i+1) \bmod m, j(i))$ on level $(i+1) \bmod m$. Here $j(i) = a_0a_1...a_{i-1}c_ia_{i+1}...a_{m-1}$, where c_i denotes the binary complement of a_i. Figure 11 shows a 3-dimensional butterfly network.

1.2.11 The Cube-Connected Cycles CCC_m

The CCC_m is similar to the hypercube except that each vertex is replaced by a cycle of m nodes. The i^{th} dimension edge incident to a node of the hypercube is then connected to the i^{th} node of the corresponding cycle of the CCC_m. This CCC_m has $m2^m$ vertices, diameter $\lfloor \frac{5m}{2} \rfloor -1$ and maximum degree 3. It has been shown that $b(CCC_m) = \lceil \frac{5m}{2} \rceil -1$ (see [34]). Figure 12 shows a 3-dimensional cube-connected cycle.

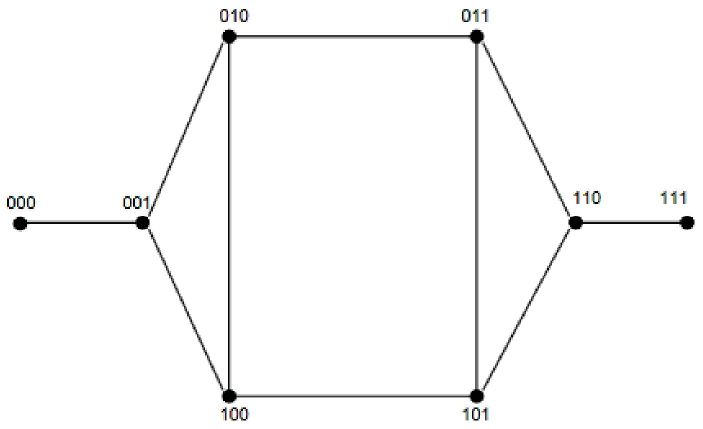

Figure 10: Shuffle-Exchange graph SE_3

1.2.12 Other Topologies

There has been a lot of research into other types of graph topologies besides the ones mentioned above. The Kn*ö*del graphs have been studied extensively in these papers [3], [10], [11], [16], [17], [31]. A unicyclic graph is a connected graph containing exactly one cycle [18]. The broadcast time of a bipartite double loop graph is $d + 2$ where d is the diameter of the graph [19]. Polynomial time broadcasting solutions has been researched in necklace graphs [21] and fully connected trees [20]. Optimal broadcasting in a 2-dimensional Manhattan graph is shown in [6].

1.3 Thesis Contribution

The rest of this thesis is organized as follows. In chapter 2 we will define the cactus graph and talk about broadcasting on some subclasses of the cactus graph. In chapter 3 we will present four heuristic algorithms that perform broadcasting on a k-cycle graph. In chapter 4 we will talk about the implementation details of the algorithms and show results of running these algorithms on many different sized k-cycle graphs. In chapter 5 we will conclude the thesis and talk about some future work that can be done.

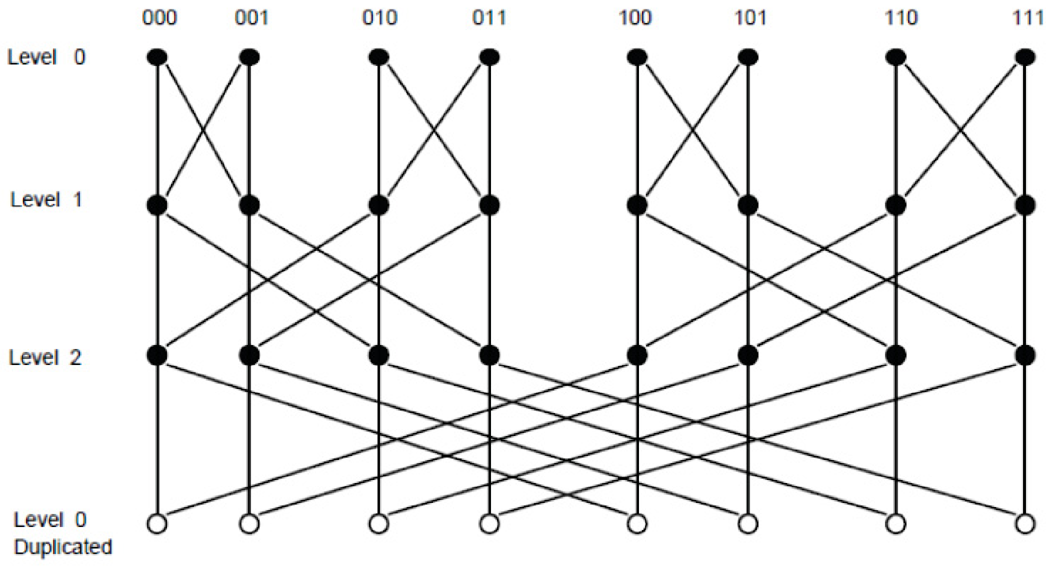

Figure 11: Butterfly graph BF_3

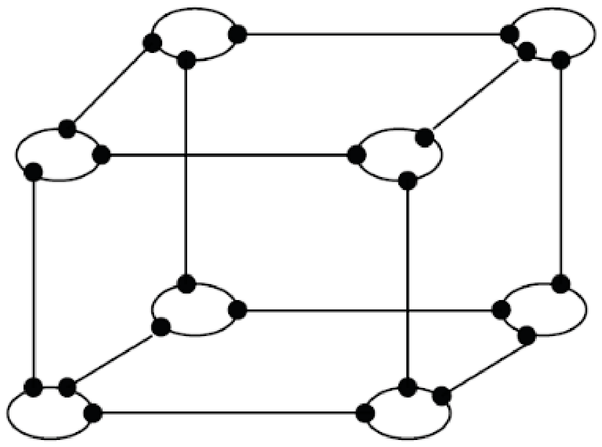

Figure 12: Cube-Connected Cycle CCC_3

Chapter 2

Cactus Graphs

In this section, we introduce the Cactus graph. First, we will define the cactus graph topology followed by a brief discussion on broadcasting on subclasses of the cactus graph.

2.1 Model Definition

A cactus graph is a connected graph in which any two simple cycles have at most one vertex in common. Equivalently, it is a connected graph in which every edge belongs to at most one simple cycle. Historically cacti where studied under the name of Husimi trees by Frank Harary and George Eugene Uhlenbeck in honor of previous work on these graphs by Kôdi Husimi [15], [23]. A cactus graph can be constructed from a tree by replacing some set of edges with cycles of arbitrary size. Note that every pseudo-tree (i.e., a graph containing exactly one cycle C_n for some $n \geq 3$) is a cactus graph.

There has been a lot of interesting research that use cacti. For example, the facility location problem which is a branch of operations research and computational geometry concerned with the optimal placement of facilities to minimize transportation costs while considering factors like avoiding placing hazardous materials near housing, and competitors' facilities. This problem is NP-Hard for general graphs but can be solved in polynomial time for cacti [2], [42]. Cacti are special cases

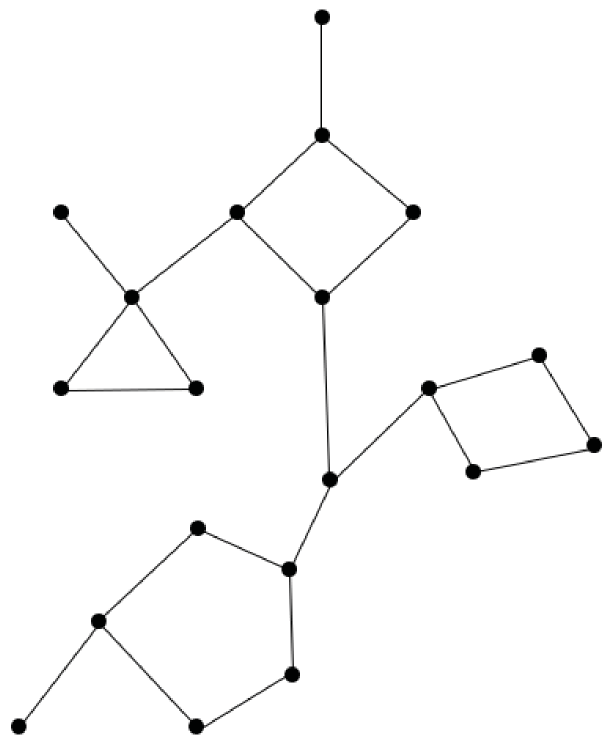

Figure 13: Cactus Graph

of outerplanar graphs, several combinatorial optimization problems on graphs may be solved for them in polynomial time [32]. Another application of Cacti is in comparative genomics as a way of representing the relationship between different genomes or parts of genomes [38].

There are also a lot of well-known problems in graph theory that have linear or polynomial time solutions on cacti. For example, all-pair shortest path problem [35], domination problem [25] coloring problem [30] and labeling problem [29].

Cacti are also used in the study of combinatorial optimization because a lot of efficient solutions to many problems can be generalized to cactus graphs, often within the same time complexity [37], [42].

2.2 Broadcasting in Cactus Graphs

In this section, we will discuss broadcasting in cactus graphs. Broadcasting in an arbitrary cactus graph is not easy, therefore we will look at algorithms that solves broadcasting in different subclasses of cacti.

2.2.1 Unicyclic Graphs

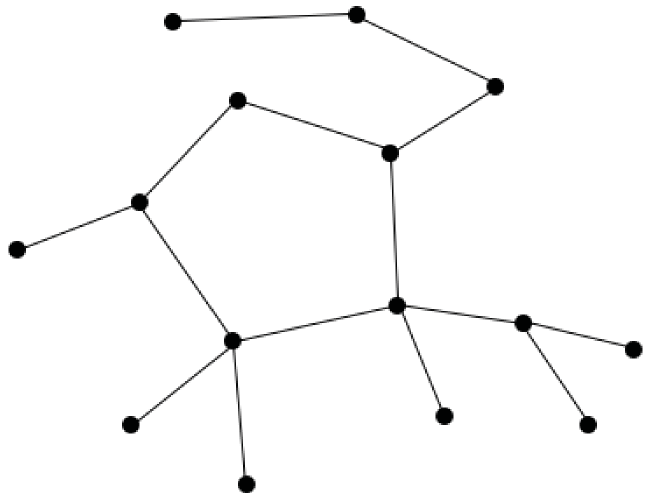

Figure 14: Unicyclic Graph

A unicyclic graph (Figure 14) is a connected graph with only one cycle. You can also say that it is a tree with only one extra edge. It can also be seen as a cycle where every vertex on the cycle is the root of a tree. In the paper [18] a linear $O(|V|)$ algorithm is presented that determines the broadcast time from any vertex u in an arbitrary unicyclic graph $G = (V, E)$.

2.2.2 Necklace Graphs

A necklace graph is a collection of cycles, where each consecutive pair of cycles is connected by one vertex. In other words, a necklace graph is a chain of cycles (see figure 15). In [21] a linear $O(|V|)$ algorithm is presented that determines the broadcast time of an arbitrary necklace graph $G = (V, E)$.

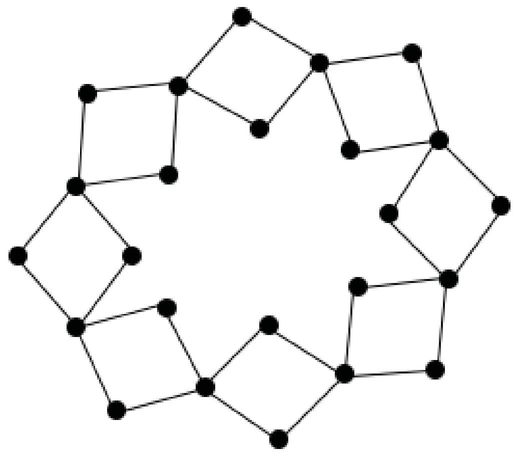

Figure 15: Necklace Graph

2.2.3 k-Cycle Graphs

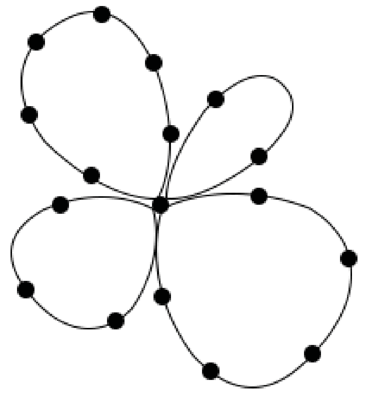

Figure 16: k-cycle Graph (where $k = 4$)

A k-cycle graph is a collection of k cycles of arbitrary lengths all connected to a central vertex (see figure 16). In [4] a constant approximation algorithm to find the broadcast time of an arbitrary k-cycle graph is given. They also show the optimality of the algorithm on some subclasses of the k-cycle graph.

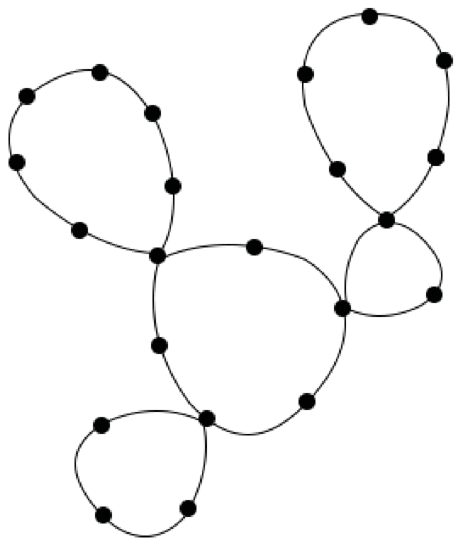

Figure 17: 2-Restricted Cactus Graph

2.2.4 2-Restricted Cactus Graphs

Cactus graphs are defined to be connected graphs where no two cycles have more than one vertex in common. In a cactus graph it is possible to have a vertex that belongs to more than two cycles. A 2-restricted cactus graph is a cactus graph such that a vertex can belong to at most 2 cycles (see figure 17). Broadcasting in a 2-restricted cactus graph is not as easy as it seems. In [36] a partial solution to broadcasting in a 2-restricted cactus graph is given along with an explanation of why it is difficult.

2.2.5 k-Restricted Cactus Graphs

A k-restricted cactus graph is a cactus graph where no more than k cycles can have more than one vertex in common, or equivalently, a cactus graph in which every vertex is on at most k cycles (see figure 18). In [5] a $O(n \log \Delta)$ algorithm for broadcasting on a k-restricted cactus graph from any originating vertex is given, where Δ is defined as the maximum degree of all vertices of the graph. Another algorithm that calculates the broadcast time for all vertices in a k-restricted cactus graph with the same time complexity is given. The algorithm also provides an optimal broadcast scheme

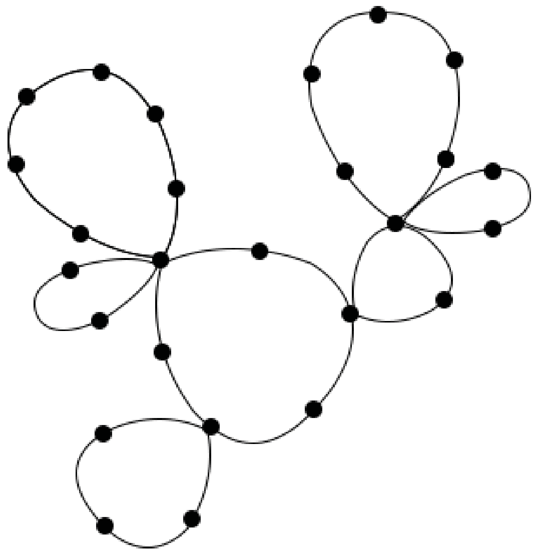

Figure 18: k-restricted Cactus Graph (where $k = 3$)

for every vertex. They also compute the broadcast center of a k-restricted cactus graph.

Chapter 3

Heuristic Algorithms for Cactus Graph Broadcasting

In this chapter, we will present some heuristic algorithms for broadcasting in a subclass of the cactus graph CG called a k-cycle graph G_k. A k-cycle graph is a collection of k cycles of arbitrary lengths all connected to a central vertex (see figure 19). Given a graph G there is a way to determine if the graph is a k-cycle graph. Algorithm 1 will return the sizes of the cycles of the k-cycle graph or **false** if the given graph G is not a k-cycle graph. It will perform a depth-first search starting from an arbitrary vertex v and check the degree of each vertex. The degree of each vertex must be 2 except for one vertex which is the common vertex to all cycles that must have an even degree. This central vertex will be saved and used in another depth-first search right after the first one. The second depth-first seach starts from the central vertex and counts the sizes of each cycle. The sizes are added to a list and will be returned by the algorithm. This return value is a list with the sizes of the cycles which is the input for other algorithms defined in this chapter.

Each cycle of the k-cycle graph will be assigned values to the edges incident to the originator vertex u. These values correspond to the broadcast round that we visit those edges. From originator vertex u we can visit two edges of each cycle. For example, the i^{th} cycle C_i will get values assigned

Algorithm 1 iskcyclegraph: Procedure to determine if a given graph G is a k-cycle graph. Returns a list with the size of the cycles of the k-cycle graph or **false** if the given graph is not a k-cycle graph.

```
procedure isKCycleGraph($G, v$)
    $cycleSizeList$                                   ▷ Will hold the sizes of the cycles
    $centralVertex \leftarrow$ false        ▷ used to track if the cental vertex has been found
    $S$ is a stack
    $S.push(v)$
    while $S$ is not empty do
        $u \leftarrow S.pop()$
        if $u$ is not labeled as discovered then
            $deg \leftarrow$ degree of vertex $u$
            if $deg > 2$ then
                if $deg \bmod 2 = 0$ and $centralVertex =$ false then
                    $centralVertex \leftarrow u$                ▷ we found the central vertex
                else
                    return false   ▷ degree is odd or the central vertex has already been found
                end if
            else if $deg < 2$ then
                return false                        ▷ cannot have vertex with degree $< 2$
            end if
            set $u$ as discovered
            for all edges from $u$ to $w$ in $G.adjacentEdges(u)$ do
                if $w$ is not labeled as discovered then
                    $S.push(w)$
                end if
            end for
        end if
    end while
    $S.push(centralVertex)$
    $cycleSize = 0$
    while $S$ is not empty do
        $u \leftarrow S.pop()$
        if $u$ is not labeled as discovered then
            set $u$ as discovered
            $foundUndiscovered \leftarrow$ false
            $cycleSize \leftarrow cycleSize + 1$
            for all edges from $u$ to $w$ in $G.adjacentEdges(u)$ do
                if $w$ is not labeled as discovered then
                    $S.push(w)$
                    $foundUndiscovered \leftarrow$ true
                end if
            end for
            if $foundUndiscovered =$ false then
                $cycleSizeList \leftarrow cycleSizeList + \{cycleSize\}$
                $cycleSize \leftarrow 1$            ▷ Reset cycle size to 1 (central vertex)
            end if
        end if
    end while
    return $cycleSizeList$                            ▷ Return the sizes of the cycles
end procedure
```

to each of it's edges, $e_{i,1}$ and $e_{i,2}$, that represent which broadcast round we will visit those edges.

The optimal broadcast time of G_k is when the assignment of rounds to each incident edge of u results in the minimal broadcast time. Determining these assignments is the problem that needs to be solved. The following algorithms will assign the rounds to these edges and calculate the total broadcast time $b(G_k)$. Depending on the sizes of the cycles in G_k certain algorithms will perform better than others.

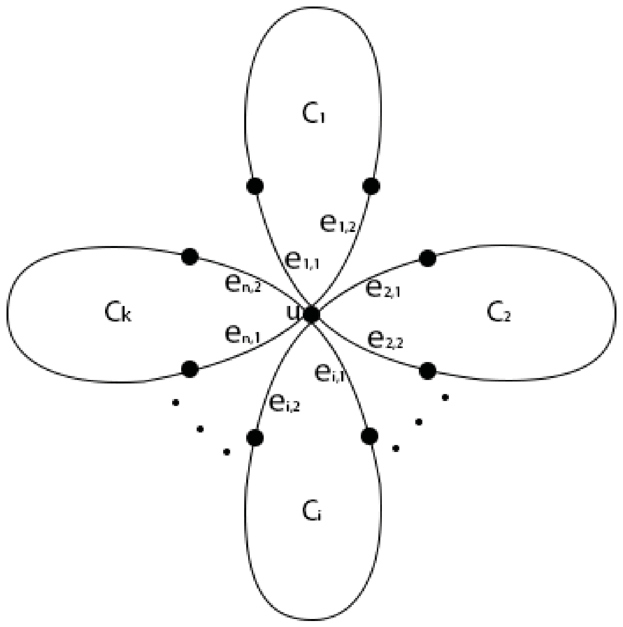

Figure 19: k-cycle graph to use for Algorithms

3.1 Lower Bounds on Broadcast Time for k-cycle Graph

In this section, we will give lower bounds on the broadcast time of a k-cycle graph G_k from an arbitrary vertex u. A lower bound on the broadcast time means that broadcasting will take at least that amount of time to complete. These lower bounds were presented in the paper [4] with proofs. The paper gives lower bounds on when the originating vertex u is the central vertex and when it is a vertex on a cycle.

Definition 1. *Let $l_1 \geq l_2 \geq ... \geq l_k \geq 2$, where l_i is the number of vertices in cycle C_i (excluding vertex u) for all $1 \leq i \leq k$.*

3.1.1 Lower Bounds When Originator is the Central Vertex

In this section, we will give lower bounds on the broadcast time of a k-cycle graph when the originator is the central vertex.

Lemma 1. *Let G_k be a k-cycle graph where the originator is the central vertex u. Then (i) $b(u) \geq k+1$. (ii) $b(u) \geq \left\lceil \frac{l_j+2j-1}{2} \right\rceil$ for j, $1 \leq j \leq k$. (iii) $b(u) \geq \left\lceil \frac{2k+l_j+2j+1}{4} \right\rceil$. for j, $1 \leq j \leq k$.*

Proof. (i): Under any minimum time broadcast scheme, k time units are necessary to inform at least one vertex in each of the k cycles from vertex u. Since $l_j \geq 2$ for any j, where $1 \leq j \leq k$, at least one more time unit is required to inform the second vertex on the cycle which initially receives the message from u at time unit k. So, $b(u) \geq k+1$.

(ii): We consider any cycle C_j where $1 \leq j \leq k$. Under any minimum time broadcast scheme all vertices in C_j must be informed. u informs the k cycles in some order and assume it initially informs C_j at time unit j or later. Then u informs its second neighboring vertex in C_j no sooner than time unit $j+1$. At time unit j there are at least $l_j - 1$ uniformed vertices in C_j. Starting at time $j+1$ onwards, C_j receives the message from both directions from u. At each time unit two new vertices on C_j will get informed. So, $b(u) \geq j + \left\lceil \frac{l_j}{2} \right\rceil = \left\lceil \frac{l_j+2j-1}{2} \right\rceil$. Suppose, by contradiction u initially calls path C_j before time j. Then by the pigeonhole principle these exists m, $1 \leq m \leq j-1$ such that u initially calls C_m at time j. Similarly at time unit j there are at least $l_m - 1$ uniformed vertices in C_m. If, starting at time $j+1$ onwanrds, C_m receives the message from both directions from u, then $b(u) \geq \lceil \frac{l_m-1}{2} \rceil = \left\lceil \frac{l_m+2j-1}{2} \right\rceil \geq \left\lceil \frac{l_j+2j-1}{2} \right\rceil$ as $l_m \geq l_j$. Hence, $b(u) \geq \left\lceil \frac{l_j+2j-1}{2} \right\rceil$.

For the proof of (iii), we combine the inequalities in (i) and (ii). We get $2b(u) \geq k+1+\left\lceil \frac{l_j+2j-1}{2} \right\rceil \geq \left\lceil \frac{l_j+2j+2k+1}{2} \right\rceil$. Hence, $b(u) \geq \left\lceil \frac{l_j+2j+2k+1}{2} \right\rceil$ for any j, $1 \leq j \leq k$. □

Lemma 2. *Let G_k be a k-cycle graph where the originator is the central vertex u and n is the total number of vertices in G_k. Then (i) $b(u) \geq \lceil \frac{n-1}{2k} + k - \frac{1}{2} \rceil$ if $b(u) \geq 2k$. (ii) $b(u) \geq \left\lceil \sqrt{(2n - \frac{7}{4})} - \frac{1}{2} \right\rceil$*

if $k+1 \leq b(u) \leq 2k-1$.

Proof. (i): Since $b(u) \geq 2k$, then u will be busy informing its adjacent vertices in k different cycles at time units $1, 2, ..., 2k$. By $b(u)$ time units, u can inform at most $b(u), b(u)-1, ..., b(u)-(2k-1)$ vertices in these k different cycles. So, $n \leq b(u)+b(u)-1+...+b(u)-(2k-1)+1 \Rightarrow n \leq 2kb(u)-k(2k-1)+1$. Hence, $b(u) \geq \lceil \frac{n-1}{2k} + k - \frac{1}{2} \rceil$.

(ii): Since $k+1 \leq b(u) \leq 2k-1$, then u can inform its adjacent vertices in k different cycles at time units $1, 2, ..., b(u)$, where $b(u) \leq 2k-1$. By $b(u)$ time units, u can inform at most $b(u), b(u)-1, ..., 1$ vertices in these k different cycles. So, $n \leq b(u)+b(u)-1+...+1+1 \Rightarrow n \leq \frac{b(u)(b(u)+1)}{2}+1 \Rightarrow b(u)^2+b(u)-(2n-2) \geq 0$. Roots of $b(u)$ are $\frac{-1 \pm \sqrt{8n-7}}{2}$. Considering the positive root of $b(u)$, we get $b(u) \geq \left\lceil \sqrt{(2n-\frac{7}{4})} - \frac{1}{2} \right\rceil$. □

Lemma 3. *There is a minimum time broadcast scheme from w in G_k in which w first sends the information along the shortest path towards vertex u.*

Proof. Let S_1 be a minimum broadcast scheme, $b_{S_1}(w) = b(w, G_k)$ under which w first informs its adjacent vertex along the longer path towards vertex u. We will construct a new broadcast scheme S_2 under which w first sends information towards the shorter path. We will show that $b_{S_2}(w) \leq b_{S_1}(w) = b(w, G_k)$.

According to scheme S_1, w informs its adjacent vertex along the shorter path at time two. Now we construct a new broadcast scheme S_2 where w informs its adjacent vertex along the shorter path at time one. The order in which u broadcasts along the remaining $k-1$ cycles is the same in both schemes. However, under S_2, every vertex along the longer path towards vertex u from w will receive the message exactly one time unit later compared to S_1. To prove that $b_{S_2}(w) = b(w, G_k)$ we consider two cases:

Case 1: under S_1, u is informed along the shorter path at time $b_1 \leq b(w, G_k)$: Under S_2 all the vertices along the shorter path will be informed exactly one time unit earlier. So, u is informed at time $b_1 1$. u has exactly one free time unit immediately after $b_1 - 1$ to inform its adjacent vertex along the longer path towards w. Since the broadcast time in the remaining $k-1$ paths remains the same, $b_{S_2}(w) \leq b_{S_1}(w)$.

Case 2: under S_1, u is informed along the longer path from w: Recall the length of the shorter path is d and the length of the longer path is $l_j + 1 - d$. Under S_1, u is informed along the longer path from w when either $d = l_j + 1 - d$ or $d + 1 = l_j + 1 - d$. When $d = l_j + 1 - d$, it is quite trivial that $b_{S_2}(w) \leq b_{S_1}(w)$ since the broadcast time in the remaining $k-1$ paths remains the same. When $d + 1 = l_j + 1 - d$: Recall that under S_2 all the vertices along the shorter path will be informed exactly one time unit earlier. So u is informed at time unit d instead of time unit $l_j + 1 - d = d + 1$ under scheme S_1. u has exactly one free time unit immediately after d to inform its adjacent vertex along the longer path towards w. Since the broadcast time in the remaining $k - 1$ paths remains the same, $b_{S_2}(w) \leq b_{S_1}(w)$. □

3.1.2 Lower Bounds when Originator is Not the Central Vertex

In this section, we will give lower bounds on the broadcast time of a k-cycle graph when the originator is not the central vertex.

Lemma 4. *Let G_k be a k-cycle graph where the originator is any vertex w on a cycle C_m and the length of the shortest path from w to vertex u is d. Then (i) $b(w) \geq d + k$. (ii) $b(w) \geq d + \lceil \frac{l_j+2j-2}{2} \rceil$ for any j, $1 \leq j \leq k$. (iii) $b(w) \geq d + \lceil \frac{2k+l_j+2j-2}{4} \rceil$ for any j, $1 \leq j \leq k$.*

Proof. (i): By Lemma 3 there is a minimum time broadcast scheme from originator w in G_k in which w first sends the information along the shorter path towards vertex u. Considering this minimum broadcast scheme, u is informed no earlier than d time units. It takes another $k - 1$ time units to inform at least one vertex in each of the remaining $k - 1$ cycles from u. Recall that $l_j \geq 2$ for any j, where $1 \leq j \leq k$. So, at least one more time unit is required to inform the second vertex on the cycle which initially receives the message from u at time unit $d + k - 1$. So, $b(w) \geq d + k$.

(ii): Similarly, at least d time units are necessary for u to receive the message from w .Now, we consider any cycle C_j where $1 \leq j \leq k$ and $j \neq m$. Under any minimum time broadcast scheme all vertices in C_j must be informed. u informs the remaining $k - 1$ cycles in some order and assume it initially informs C_j at time unit $d+j$ or later. Then u informs C_j along the second branch no sooner than time unit $d+j+1$. At time unit $d+j$ there are at least l_j-1 uninformed vertices in C_j. Similar to the argument given in Lemma 1(ii), we car write $b(w) \geq d+j+\left\lceil \frac{l_j-1}{2} \right\rceil = d+\left\lceil \frac{l_j+2j-1}{2} \right\rceil \geq d+\left\lceil \frac{l_j+2j-2}{2} \right\rceil$.

When $j=m$, the number of uninformed vertices in C_m at time d, denoted as $\Gamma(m) = l-(2d-1)$. Considering $j=1$ and $l=\Gamma(m)$ for the cycle C_m, we get $b(w) \geq d+\left\lceil \frac{l_j+2j-2}{2} \right\rceil$ for any j, $1 \leq j \leq k$ included m.

For the proof of (iii), we combine the inequalities in (i) and (ii). We get $2b(u) \geq d+k+d+\left\lceil \frac{l_j+2j-2}{2} \right\rceil \geq 2d+\left\lceil \frac{l_j+2j+2k-2}{2} \right\rceil$. Hence, $b(w) \geq d+\left\lceil \frac{l_j+2j+2k-2}{4} \right\rceil$ for any j, $1 \leq j \leq k$. $\square$

3.2 Algorithm BroadcastBucket

The *BroadcastBucket* algorithm (See Algorithm 2) will determine if it is possible to broadcast in a given k-cycle graph G_k within a given time t. To determine which edges to visit during each round we first sort the cycles from largest to smallest. We make an assumption that we want to visit the first edge of the largest cycle during round 1. We can now calculate which round we will visit the second edge of the largest cycle. This can be determined by performing a calculation based on what values are already given. For example, let's take a cycle C_i. We are given the time t, which is the target broadcast time for the k-cycle graph G_k. We know the size of the cycle C_i, denoted by $|C_i|$, and we are given the round in which to visit the first edge $e_{i,1}$ of cycle C_i. In target time t we know we can visit $t-(e_{i,1}-1)$ vertices through edge $e_{i,1}$ and $t-(e_{i,2}-1)$ vertices through edge $e_{i,2}$ (Figure 20), where $e_{i,1}$ and $e_{i,2}$ are the rounds in which those edges are visited. Therefore, if broadcasting is possible in time t, we know $t-(e_{i,1}-1)+t-(e_{i,1}-1)+1 \geq |C_i|$. After some simple arithmetic, we get $e_{i,1}+e_{i,2} \leq 2t+3-|C_i| = a_i$. This gives these inequalities, $e_{1,1}+e_{1,2} \leq a_1$, $e_{2,1}+e_{2,2} \leq a_2$, ... , $e_{k,1}+e_{k,2} \leq a_k$ where k is the number of cycles. We can now calculate the latest possible round we need to visit the second edge $e_{i,2}$ of cycle C_i so that broadcasting on the cycle C_i will finish in time t. We denote this by $b_i = a_i - e_{i,1} = 2t+3-|C_i|-e_{i,1}$. From the inequalities above we know

$e_{i,2} \leq b_i$. This means the latest round in which we can visit $e_{i,2}$ of cycle C_i and finish broadcasting in time t is b_i.

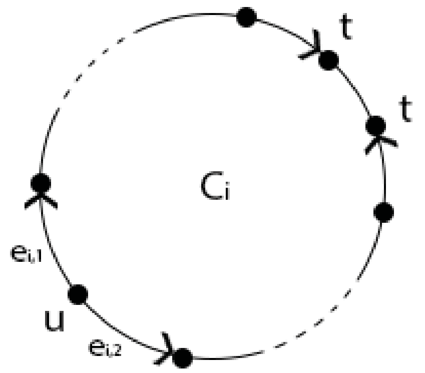

Figure 20: Cycle Broadcast

This algorithm uses a bucket B of size $2k$ where k is the number of cycles in G_k. This bucket is used to hold the edge assignments, for example, the edge $e_{1,1}$ is the first edge of the largest cycle and will be assigned to $B[1]$. This means that we will visit edge $e_{1,1}$ during the first round. As per our calculations above we assign the second edge $e_{1,2}$ of the largest cycle to bucket $B[b_i]$. Then we need to get the lowest empty bucket and assign that value to $e_{2,1}$ and use that when we calculate the bucket assignment of the second to largest cycle C_2. We continue to do this until all edges are assigned to a bucket. Bucket assignments are not necessarily unique. We can have more than one edge assigned to the same bucket. After assigning all edges to a bucket there may be some buckets that have more than one edge assigned to it. We can attempt to move these edges to a smaller empty bucket, this is essentially visiting the second edge $e_{i,2}$ of cycle C_i at an earlier round than what we calculated. Which is not a problem because the bucket assignment is the latest possible round for the second edges $e_{i,2}$. We also know that all first edges will be assigned to an empty bucket. On line 19 we perform a quick check to make sure that the first edge visited on the cycle C_i does not finish broadcasting on the cycle before the second edge starts broadcasting on this cycle. If this is the case we can assign the second edge $e_{i,2}$ to the last bucket.

After all edges have been assigned to a bucket, we need to make sure that we have room to uniquely assign values to all edges $(e_{1,1}, e_{1,2}), (e_{2,1}, e_{2,2}), ..., (e_{i,1}, e_{i,2}), ..., (e_{k,1}, e_{k,2})$. This can be calculated easily by keeping a count of the number of edges assigned to each bucket $B[i]$ starting from the first bucket. If at some point, we count that there are more edges assigned to the buckets we checked compared to the total number of buckets checked then we know we won't be able to broadcast in the given time t. For example, if we counted 5 edges that are assigned to the first 4 buckets then we know broadcasting of G_k in time t is not possible with this bucket assignment. If on the other hand the total number of edges counted does not exceed the count of the buckets checked then we can uniquely assign the edges to the buckets (if they are not already uniquely assigned) and finish broadcasting in time t. See Figure 21 for example bucket edge assignments. The first example is perfect, the second is not perfect but there is enough room to move edges in the last bucket to earlier empty buckets. The last bucket edge assignment tells us that it is not possible to broadcast in the time t. The *BroadcastBucket* algorithms will return a boolean value depending on it if is possible to broadcast in the given time t.

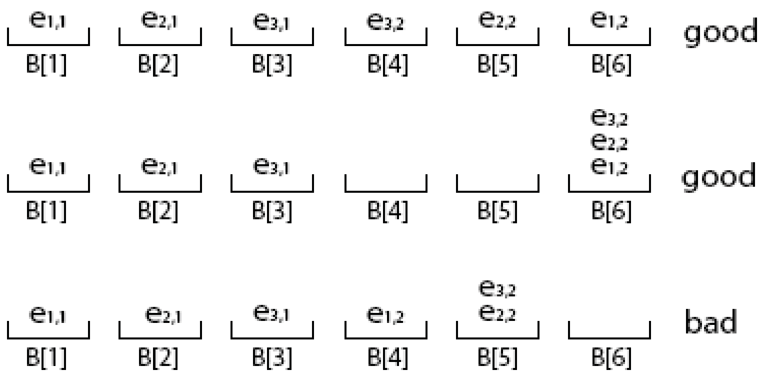

Figure 21: Example bucket edge assignments

To analyze the runtime of the *BroadcastBucket* algorithm we must look at different parts of the algorithm. The first is sorting the cycles from largest to smallest on line 10. Sorting takes $O(k \log k)$ where k is the number of cycles in G_k. We build a min heap from an array containing the values $1, 2, ..., 2k$. These values represent the indexes of the buckets and we want to keep track of the smallest empty bucket. Building a min heap takes $O(k)$. Next on line 13 we loop k times.

Inside this loop on line 14 we have to search for the smallest empty bucket. We extract the index of the smallest empty bucket from the *smallestEmptyBucketHeap* heap. Extracting a value from a min heap takes $O(\log k)$. In total this loop takes $O(k \log k)$. Finally, on line 13 we loop another k times which results in $O(k)$. The final runtime will be $O(k \log k) + O(k \log k) + O(k) = O(k \log k)$.

The *BroadcastBucket* algorithm expects a broadcast time t as a parameter. We know that the minimum number of rounds to finish broadcasting in any graph G is $b(G) \geq \lceil \log |V| \rceil$, where $|V|$ is the total number of vertices in the graph. We also know that the maximum number of rounds to broadcast in a graph G is $b(G) = |V| - 1$, since in the worst case there will be only one newly informed vertex each round. Therefore, we can perform a binary search between the minimum and maximum broadcast rounds and call *BroadcastBucket* until we find the lowest possible broadcast rounds for the given k-cycle graph G_k. This algorithm is called *BroadcastGuess* because it's basically taking a guess at the possible broadcast time (See Algorithm 3).

The *BroadcastGuess* algorithm performs a binary search of size n, where n is the number of vertices in the graph G_k. Inside the while loop on line 8 is calls the *BroadcastBucket* algorithm. Since the runtime of *BroadcastBucket* is $O(k \log k)$ the runtime of *BroadcastGuess* is $O(k \log k \log n)$, where k is the number of cycles in the k-cycle graph G_k and n is the number of vertices in the k-cycle graph G_k.

Let's take the simple example of a k-cycle graph with 3 cycles of sizes 8, 7, and 3 with $broadTime = 6$ and run it through the *BroadcastBucket* algorithm. The largest cycle is size 8, and the first edge of this cycle will be $e_{1,1} = x = 1$, this means it will get assigned to the first bucket. Then we calculate which bucket the second edge of the cycle of size 8 will be assigned to. From the calculations we get $a = 2 * 6 + 3 - 8 = 7$, then $e_{1,2} = b = a - x = 7 - 1 = 6$, this means we assign the second edge of the cycle of size 8 to bucket 6. The next largest cycle is size 7 and its first edge gets assigned the lowest empty bucket which is $e_{2,1} = x = 2$ and its second edge to $a = 2 * 6 + 3 - 7 = 8$, $e_{2,2} = b = a - x = 8 - 2 = 6$. The second edge also gets assigned to bucket 6, which is fine. The last cycle of size 3. The first edge gets assigned to the lowest empty bucket of $e_{3,1} = x = 3$. Then the

Algorithm 2 BroadcastBucket: This procedure will determine if broadcasting on a given k-cycle graph G_k from a central vertex u is possible in a given time $broadTime$.

```
1: procedure BroadcastBucket($G_k, u, broadTime$)
2:     $k \leftarrow$ The number of cycles in $G_k$
3:     $C \leftarrow$ The set of all cycles in $G_k$
4:     $B \leftarrow$ A set of size $2k$ (buckets)
5:     $smallestEmptyBucketHeap \leftarrow$ build a min heap from an array with values 1, 2, ..., 2k
6:     $E \leftarrow$ The set of all edges incident to originator vertex $u$
7:     $\{|E|$ is size $2k\}$
8:     $\{e_{i,1}$ is the first edge of the $i^{th}$ cycle$\}$
9:     $\{e_{i,2}$ is the second edge of the $i^{th}$ cycle$\}$
10:    {Sort cycles from largest to smallest size, $C_1 \geq C_2 \geq C_3 \geq C_{k-1} \geq C_k$}
11:    $i \leftarrow 1$
12:    $x \leftarrow 1$
13:    while $i \leq k$ do
14:        $x \leftarrow$ extract root of $smallestEmptyBucketHeap$      ▷ Index of smallest empty bucket
15:        $B[x] \leftarrow e_{(i,1)}$
16:        $a \leftarrow 2 \times broadTime + 3 - |C_i|$
17:        $b \leftarrow a - x$
18:        $y \leftarrow min(b, 2k)$
19:        if $y - x \geq |C_i| - 1$ then
20:            $y \leftarrow 2k$
21:        end if
22:        $B[y] \leftarrow e_{(i,2)}$
23:        $i \leftarrow i + 1$
24:    end while
25:    {Determine if there are enough room in $B$ for all edges}
26:    $count \leftarrow 0$
27:    $i \leftarrow 1$
28:    while $i \leq 2k$ do
29:        $num \leftarrow$ number of edges in $B[i]$
30:        if $num \geq 0$ then
31:            $count \leftarrow count + num$
32:            if $count > i + 1$ then
33:                return false      ▷ broadcasting in $broadTime$ is impossible
34:            end if
35:        end if
36:        $i \leftarrow i + 1$
37:    end while
38:    return true      ▷ broadcasting in $broadTime$ is possible
39: end procedure
```

Algorithm 3 BroadcastGuess: This procedure will generate possible broadcast times of a k-cycle graph G_k by performing a binary search between the theoretical minimum and maximum possible broadcast times. It will call *BroadcastBucket* to determine if broadcasting is possible in the given time.

```
1: procedure BroadcastGuess($G_k, u$)
2:     $numVert \leftarrow |V|$                      ▷ $|V|$ is the number of vertices in $G_k$
3:     $left \leftarrow \log \lceil numVert \rceil$          ▷ Minimum possible broadcast time
4:     $right \leftarrow numVert - 1$                ▷ Maximum possible broadcast time
5:     $broadTime \leftarrow numVert - 1$            ▷ The current broadcast time
6:     while $left \leq right$ do                   ▷ We have the answer if r is 0
7:         $middle \leftarrow \lceil \frac{(left+right)}{2} \rceil$
8:         $isBroadcastPossible \leftarrow$BroadcastBucket($G_k, u, middle$)
9:         if $isBroadcastPossible \neq false$ then
10:            $broadTime \leftarrow middle$
11:            $right \leftarrow middle - 1$
12:        else
13:            $left \leftarrow middle + 1$
14:        end if
15:    end while
16:    return $broadTime$                 ▷ The minimum broadcast time for $G_k$ from vertex $u$
17: end procedure
```

second edge $a = 2 * 6 + 3 - 3 = 12$, and $e_{3,2} = a - x = 12 - 3 = 9$. Since we got a value higher than the number of buckets we can just assign this to the last bucket, therefore $e_{3,2} = 6$. Now we have three values assigned to the last bucket we can move two down to the free buckets 4 and 5 and not affect the broadcast time of 6 (see figure 23 for the bucket assignments of this example).

In the *BroadcastBucket* algorithm we make the assumption that the largest cycle is visited first. This seems to be an obvious assumption to get the best broadcast time, but as it turns out this is not always the case. A simple counterexample is shown in figure 22, where the optiomal broadcast time is actually when you visit the second to largest cycle first.

Proposition 1. *If $l_i = l_{i+1}$ for all $1 \leq i \leq k$, where l_i is the length of cycle C_i, then the BroadcastBucket algorithm finds the optimal broadcast time, $b_B(G_k) = b(G_k) = \lceil \frac{l}{2} \rceil + k - 1$.*

Proof. Since all the cycles are the same size we will denote the length of each cycle by l. We must show that $b_B(G_k) = \lceil \frac{l}{2} \rceil + k - 1$, where k is the number of cycles in the k-cycle graph G_k. We must prove that the *BroadcastBucket* algorithm will generate the broadcast scheme in figure 24 when the given time is $broadTime = \lceil \frac{l}{2} \rceil + k - 1$.

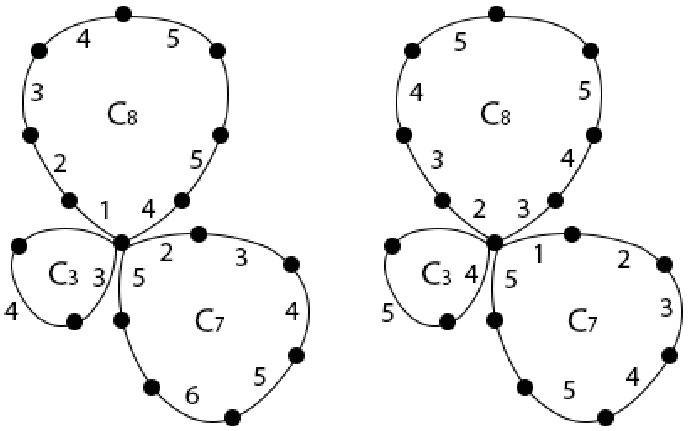

Figure 22: Showing that *BroadcastGuess/BroadcastBucket* is not optimal

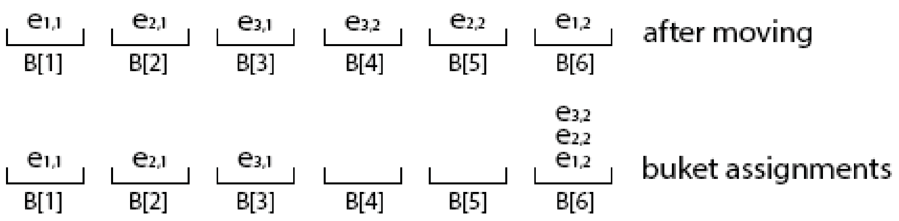

Figure 23: Bucket assignments from example 8 7 3

Starting at the first cycle C_1 we assign $e_{1,1} = x = 1$, which means we will visit the first edge of cycle C_1 at round 1. Then we calculate the latest possible round we can visit the second edge of cycle C_1 so that broadcasting on this cycle finishes at time $\lceil \frac{l}{2} \rceil + k - 1$. Following the algorithm we get $a = 2\left(\lceil \frac{l}{2} \rceil + k - 1\right) + 3 - l$. When l is even this results in $(l + 2k - 2) + 3 - l = 2k + 1$, and when l is odd $((l+1) + 2k - 2) + 3 - l = 2k + 2$. Now calculating $e_{1,2} = b = a - 1$, when l is even $2k$ and when l is odd $2k + 1$. When the algorithm calculates a bucket larger than the total number of buckets it will just place this edge is the last bucket $2k$. Now let's look at cycle C_2. Edge $e_{2,1} = x = 2$ because this is the lowest empty bucket. Now calculating the the second edge of this cycle we get the same value for a since the cycle size is the same, but for $e_{2,2} = b = a - x = 2k + 1 - 2 = 2k - 1$ when l is even, and $e_{2,2} = b = a - x = 2k + 2 - 2 = 2k$ when l is odd. From C_1 we know that bucket $2k$ is already assigned, then when l is odd we can just move the edge to bucket $2k - 1$ and the total

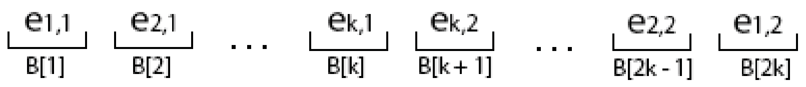

Figure 24: Optimal bucket assignment when $l_i = l_{i+1}$ for all $1 \leq i \leq k$

broadcasting time will not change.

We must now show that if for cycle C_j, where $1 \leq j \leq k$, that edge $e_{j,1} = j$ and $e_{j,2} = 2k - j + 1$, then the edges of cycle C_{j+1} will be $e_{j+1,1} = j + 1$ and $e_{j+1,2} = 2k - (j + 1) + 1$.

Since by our inductive hypothesis we know that C_j is true for all $1, 2, ..., j$, then buckets $1, 2, ..., j$ and $2k, 2k-1, ..., 2k-j+1$ are occupied. We must show that the *BroadcastBucket* algorithm assigns edge $e_{j+1,1} = j + 1$ and $e_{j+1,2} = 2k - (j + 1) + 1$. Since bucket $j + 1$ is empty, $e_{j+1,1} = x = j + 1$. Then calculating the second edge we get, $e_{j+1,2} = b = a - x = 2\left\lceil \frac{l}{2} \right\rceil + 2k + 1 - l - (j + 1)$. When l is even $e_{j+1,2} = 2k - (j + 1) + 1$, and when l is odd $e_{j+1,2} = 2k - (j + 1) + 2$. From the inductive hypothesis we know that bucket $2k - (j + 1) + 2$ is not free, but we know that bucket $2k - j + 1$ is free because the occupied buckets are $1, 2, ..., j$ and $2k, 2k - 1, ..., 2k - (j + 1) + 2$. Therefore the *BroadcastBucket* algorithm assigns $e_{j+1,2} = b = 2k - (j + 1) + 1$. In summary, at each iteration i the *BroadcastBucket* algorithm assigns the buckets i and $2k - i + 1$. The lower bound gives $b(G_k) \geq \left\lceil \frac{l}{2} \right\rceil + k - 1$, since $b_B(G_k) \geq b(G_k) \geq \left\lceil \frac{l}{2} \right\rceil + 1$. □

3.3 Algorithm CycleBroadcastTimeHelper

To calculate the broadcast time of a cycle we created a helper algorithm called $CycleBroadcastTimeHelper$ (See Algorithm 4). This algorithm will be used in the next three algorithms in this chapter.

This algorithm takes as parameters the size of a cycle C_i, the rounds the first edge of the cycle was visited $e_{i,1}$ from a start vertex, and the round the second edge of the cycle was visited $e_{i,2}$ from the start vertex. When the second edge is visited, we know that there has already been $e_{i,2} - e_{i,1}$ vertices visited through edge $e_{i,1}$. To calculate the total broadcast time of the cycle we can split the calculation into two separate calculations and add the results together for the total broadcast time.

Algorithm 4 CycleBroadcastTimeHelper: This procedure will determine the end broadcast time of a given cycle depending on which rounds we visit the first and second edges of the cycle.

```
1: procedure CycleBroadcastTimeHelper($CycleSize$, $Edge1Round$, $Edge2Round$)
2:     $t1 \leftarrow Edge2Round - Edge1Round$
3:     if $t1 < CycleSize - 1$ then
4:         $t2 \leftarrow \lceil \frac{CycleSize - t1 + 1}{2} \rceil - 1$
5:         return $Edge1Round + t1 + t2 - 1$
6:     else
7:         return $Edge1Round + (CycleSize - 1) - 1$
8:     end if
9: end procedure
```

The first part we will call $t_1 = e_{i,2} - e_{i,1}$. The time t_1 is the number of rounds that is visited from edge $e_{i,1}$ before edge $e_{i,2}$ starts helping with the broadcast. Once the second edge starts broadcasting both sided of the graph will help in broadcasting. Let's call this $t_2 = \lceil \frac{CycleSize - t1 + 1}{2} \rceil - 1$. The time t_2 is the same as broadcasting on a path where each end of the the path is informed at the beginning and can both broadcast to a neighbor vertex at the same time. The total number of rounds needed to broadcast in a cycle is $b(C_i) = t_1 + t_2$ (See figure 25).

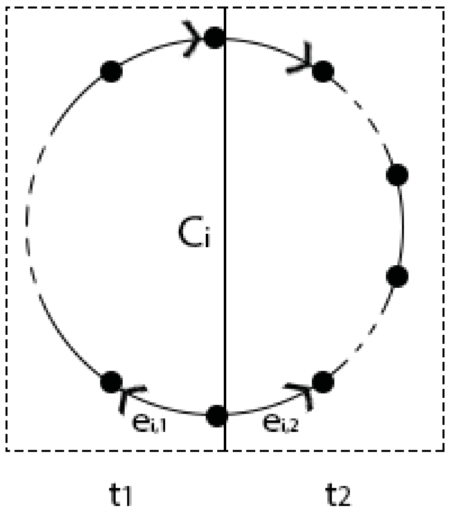

Figure 25: Cycle Broadcast Figure

This algorithm will return the round that is finishes broadcasting based on the value of $e_{i,1}$ which is the round when the first edge was visited. We just add the value of $e_{i,1}$ to the total number of rounds, $b(C_i) = e_{i,1} + t_1 + t_2 - 1$. If the broadcasting of the cycle is finished before we start broadcasting on the second edge $e_{i,2}$ then the end time will be $b(C_i) = e_{i,1} + (|C_i| - 1) - 1$. The

$CycleBroadcastTimeHelper$ algorithm runs in constant $O(1)$ time.

3.4 Algorithm BroadcastGreedyMinMax

The $BroadcastGreedyMinMax$ algorithm (See Algorithm 5) is a greedy algorithm for assigning the edges to each cycle of the k-cycle graph G_k and calculating the end broadcast round of each cycle. It depends on the cycles of G_k to be sorted from largest to smallest. The greedy choice of this algorithm is visiting cycle C_i in round i and in round $2k + 1 - i$ where k is the number of cycles in G_k. This algorithm gives unique edge assignments for all cycles $((1, 2k + 1 - 1), (2, 2k + 1 - 2), ..., (i, 2k + 1 - i), ..., (k, 2k + 1 - k))$.

Algorithm 5 BroadcastGreedyMinMax: This procedure determines the broadcast time of a given k-cycle graph G_k. It uses a greedy approach when assigning the first and second edges to each cycle C_i of the graph G_k.

```
procedure BroadcastGreedyMinMax($G_k, u$)
    {Sort cycles from largest to smallest size, $C_1 \geq C_2 \geq C_3 \geq C_{k-1} \geq C_k$}
    $broadTime \leftarrow 0$                              ▷ Will hold the worst broadcast time
    $k \leftarrow$ number of cycles in $G_k$
    $i \leftarrow 1$                                       ▷ Used in loop
    while $i \leq k$ do
        $cycleSize \leftarrow$ size of $i^{th}$ cycle of $G_k$
        $edge1Round \leftarrow i$
        $edge2Round \leftarrow (2 * k) + 1 - i$
        $cycleBroadTime \leftarrow$ CycleBroadcastTimeHelper($cycleSize, edge1Round, edge2Round$)
        if $cycleBroadTime > broadTime$ then
            $broadTime \leftarrow cycleBroadTime$
        end if
    end while
    return $broadTime$
end procedure
```

The cycles in the $BroadcastGreedyMinMax$ algorithm have to be sorted from largest to smallest. There are k cycles in G_k, therefore sorting takes $O(k \log k)$. This algorithm also loops k times on line 6 and calls $CycleBroadcastTimeHelper$ inside the loop. The runtime of $CycleBroadcastTimeHelper$ is constant. Therefore the total runtime of $BroadcastGreedyMinMax$ is $O(k \log k) + O(k) = O(k \log k)$.

3.5 Algorithm BroadcastGreedyEven

The *BroadcastGreedyEven* algorithm (See Algorithm 6) is a greedy algorithm for assigning the edges to each cycle of the k-cycle graph G_k and calculating the end broadcast round of each cycle. It depends on the cycles of G_k to be sorted from largest to smallest. The greedy choice of this algorithm is visiting cycle C_i in round i and in round $k+i$ where k is the number of cycles in G_k. This algorithm gives unique edge assignments for all cycles $((1, k+1), (2, k+2), ..., (i, k+i), ..., (k, k+k))$. This algorithm will also use the *CycleBroadcastTimeHelper* algorithm for determining the end broadcast round for each cycle. It will return the largest broadcast round calculated from each cycle which will be the broadcast time of the graph G_k.

Algorithm 6 BroadcastGreedyEven: This procedure determines the broadcast time of a given k-cycle graph G_k. It uses a greedy approach when assigning the first and second edges to each cycle C_i of the graph G_k.

```
 1: procedure BROADCASTGREEDYEVEN($G_k$)
 2:     {Sort cycles from largest to smallest size, $C_1 \geq C_2 \geq C_3 \geq C_{k-1} \geq C_k$}
 3:     $broadTime \leftarrow 0$                          ▷ Will hold the worst broadcast time
 4:     $k \leftarrow$ number of cycles in $G_k$
 5:     $i \leftarrow 1$                                  ▷ Used in loop
 6:     while $i \leq k$ do
 7:         $cycleSize \leftarrow$ size of $i^{th}$ cycle of $G_k$
 8:         $edge1Round \leftarrow i$
 9:         $edge2Round \leftarrow k + i$
10:         $cycleBroadTime \leftarrow$CYCLEBROADCASTTIMEHELPER($cycleSize, edge1Round, edge2Round$)
11:         if $cycleBroadTime > broadTime$ then
12:             $broadTime \leftarrow cycleBroadTime$
13:         end if
14:     end while
15:     return $broadTime$
16: end procedure
```

The cycles in the *BroadcastGreedyEven* algorithm have to be sorted from largest to smallest. There are k cycles in G_k, therefore sorting takes $O(k \log k)$. This algorithm also loops k times on line 6 and calls *CycleBroadcastTimeHelper* inside the loop. The runtime of *CycleBroadcastTimeHelper* is constant. Therefore the total runtime of *BroadcastGreedyMinMax* is $O(k \log k) + O(k) = O(k \log k)$.

3.6 Algorithm BroadcastRandom

The *BroadcastRandom* algorithm (See Algorithm 7) is an algorithm for assigning the edges to each cycle of the k-cycle graph G_k at random and calculating the end broadcast round of each cycle. This algorithm will create an array *edgeRounds* of size $2k$ containing the values 0 to $2k-1$. It will then randomize the array and use this when assigning values to the edges of each cycle. For example, the first entry of *edgeRounds* will be assigned to the first edge of the first cycle and the second value of *edgeRounds* will be the second edge of the first cycle. Since this algorithm also uses the *CycleBroadcastTimeHelper* algorithm to calculate the end broadcast time of each cycle it makes sure that the first edge is less than the second edge of each cycle. It will return the largest broadcast round calculated from each cycle which will be the broadcast time of the graph G_k.

Algorithm 7 BroadcastRandom: This procedure will determine the broadcast time of a k-cycle graph by randomly assigning the rounds to visit each edge of each cycle.

```
procedure BroadcastRandom($G_k$)
    $broadTime \leftarrow 0$                                   ▷ Will hold the worst broadcast time
    $k \leftarrow$ number of cycles in $G_k$
    $edgeRounds \leftarrow$ array of size $2k$
    $j \leftarrow 0$                                           ▷ Used in loop
    while $j < 2k$ do                                          ▷ Initialize edgeRounds array
        $edgeRounds[j] = j + 1$
    end while
    {Randomize edgeRounds array}
    $i \leftarrow 1$                                           ▷ Used in loop
    while $i \leq k$ do
        $cycleSize \leftarrow$ size of $i^{th}$ cycle of $G_k$
        $edge1Round \leftarrow edgeRounds[2 * (i - 1)]$
        $edge2Round \leftarrow edgeRounds[2 * (i - 1) + 1]$
        if $edge1Round > edge2Round$ then      ▷ We want edge1Round to less than edge2Round
            $tmp \leftarrow edge1Round$
            $edge1Index \leftarrow edge2Index$
            $edge2Index \leftarrow tmp$
        end if
        $cycleBroadTime \leftarrow$CycleBroadcastTimeHelper($cycleSize, edge1Round, edge2Round$)
        if $cycleBroadTime > broadTime$ then
            $broadTime \leftarrow cycleBroadTime$
        end if
    end while
    return $broadTime$
end procedure
```

The difference between the *BroadcastRandom* algorithm and the other algorithms in the chapter is that the cycles in $BroadcastRandom$ do not have to be sorted. Instead of sorting k cycles it has to randomize an array of size $2k$ on line 9. The best runtime for randomizing an array of size $2k$ is $O(k)$ by using the Fisher−Yates shuffle algorithm [41]. This algorithm also loops k times on line 11. Therefore, the total runtime of $BroadcastRandom$ is $O(k) + O(k) = O(k)$.

3.7 Algorithm SCycle

The algorithm *SCycle* (See Algorithm 8) is an algorithm that was defined in this paper [4]. It is an algorithm for broadcasting on a k-cycle graph. It starts by broadcasting on the largest cycle. This algorithm splits up the cycles into 3 sets. The set X_0 consists of the cycles where there are no informed vertices. Let there be r cycles such that $l_{10} \geq l_{20} \geq l_{30} \geq l_{r0}$, where $l_{j,0}$ is the length of the cycle C_{j0} in X_0 and $1 \leq j \leq r$. $C_{10}, C_{20}, C_{30}, ..., C_{r0}$ is a combination of r cycles from $C_1, C_2, C_3, ..., C_k$. The set X_1 consists of cycles where at least one vertex has been informed along one branch from the central vertex u. There are m cycles such that $l_{11} \geq l_{21} \geq l_{31}, ..., l_{m1}$, where l_{j1} is the number of uninformed vertices in the cycle C_{j1} in X_1 at time i and $1 \leq j \leq m$. The cycles $C_{11}, C_{21}, C_{31}, ..., C_{m1}$ is a combination of m cycles from $C_1, C_3, C_3, ..., C_k$ that are not in X_0. The set X_2 consists of the cycles which have been informed from u along both directions. Let there be p such cycles and $r + m + p = k$.

At the start of each round the algorithm checks the largest cycle in X_0 with a cycle containing the most uninformed vertices in X_1. If the number of uninformed vertices in X_1 is larger than the largest cycle in X_0 is will broadcast on X_1 and move the cycle to X_2. If the cycle in X_0 is larger it will broadcast on that cycle and move it to X_1. It does this until there are no more cycles in X_0. After each round, it also has to sort the cycles in X_1 by the number of uninformed vertices. It will continue to broadcast on X_1 until all cycles are in X_2 and the total broadcast time can be calculated. This algorithm will return the broadcast time of this k-cycle graph.

To order the cycles in sorted order this will take $O(k \log k)$. To move the cycles from one graph to another it will take $O(k)$ and the actual broadcasting will take $O(|V|)$. Therefore the complexity

Algorithm 8 SCycle: This procedure will determine the broadcast time of a k-cycle graph by adding cycles into sets and moving them to different sets when the cycle is broadcasted on.

```
procedure SCycle($G_k, x$)
    if x is not the central vertex of the $k$-cycle graph then
        $x$ broadcasts on the shorter path towards $u$        ▷ This takes $d$ time units
    end if
    $X_0 \leftarrow$ cycles $C_1, C_2, C_3, ..., C_k$
    $X_1 \leftarrow$ empty set
    $X_2 \leftarrow$ empty set
    while $X_0$ is not empty do
        if $X_1$ is not empty then
            if $l_{10} \geq l_{11} - 1$ then
                $u$ broadcasts along $C_{10}$
            else
                $u$ broadcasts along $C_{11}$
            end if
        else
            $u$ broadcasts along $C_{10}$
        end if
        if $u$ informed $C_{10}$ then
            $X_0 \leftarrow X_0 - C_{10}$
            $X_1 \leftarrow X_1 + C_{10}$
        else
            $X_1 \leftarrow X_1 - C_{11}$
            $X_2 \leftarrow X_2 + C_{11}$
        end if
        for every cycle in $X_1$ do
            $l_{j1} \leftarrow l_{j1} - 1$
        end for
        if $u$ informed along $C_{10}$ then
            Sort cycles in $X_1$ in decending order of the number of uninformed vertices.
        end if
    end while
    $X_1$ cycles will be in order of highest to lower uninformed vertices
    while $X_1$ is not empty do
        $u$ broadcasts along $C_{11}$
        $X_1 \leftarrow X_1 - C_{11}$
        $X_2 \leftarrow X_2 + C_{11}$
    end while
    return highest broadcast time out of cycles in $X_2$
end procedure
```

of this algorithm is $O(|V| + k \log k)$.

Chapter 4

Heuristic Algorithm Implementation Details and Results

In this section, we will briefly discuss the implementation details of the algorithms. We will also show a table of results on many different sized k-cycle graphs and discuss how certain sized k-cycle graphs perform better than others for each algorithm.

4.1 Implementation Details

The algorithms in the previous chapter have been implemented in the Java programming language to run simulations on different sized k-cycle graphs. The input to the algorithms are stored in a text file, one test case per line. Each test case is a list of numbers that represent the cycle size of the graph. For example, 13 42 5 3 show a cactus graph with 4 cycles of of sizes 13, 42, 5 and 3. A method called *solveCases* will loop through the text file containing all the test cases line per line and run the algorithms defined in chapter 3 on each test case. The *BroadcastRandom* algorithm is run 1000000 times per test case and the best broadcast time is used as output for this algorithm.

We want to see how good the results of the simulations are when compared to the theoretical lower bounds that we described in chapter 3 (See Lemma 1). By comparing the simulation results with these lower bounds we can determine if the simulation result for each test case is an optimal broadcast time. The lower bounds are indicated in bold text and any algorithm that resulted in a lower bound for that test case in also in bold. We also implemented the algorithm SCycle (See Algorithm 8) to compare the results of the four Heuristic algorithms with it.

4.2 Simulation Results

Table 1: Heuristic Algorithm Simulation Results

Heuristic Algorithms Simulation Results						
k-cycle Graph cycle sizes	Lower Bound	Bucket	GreedyMinMax	GreedyEven	Random	SCycle
1000 cycles of size 8	**1003**	**1003**	**1003**	1006	1884	**1003**
1000 cycles of size 1000	**1499**	**1499**	**1499**	1998	2405	**1499**
100 cycles of random size between 3 and 100	**101**	107	109	109	201	120
500 cycles of random size between 3 and 100	**501**	**501**	**501**	**501**	948	**501**
1000 cycles of random size between 3 and 100	**1001**	**1001**	**1001**	**1001**	1930	**1001**

100 cycles of random size between 3 and 10000	**4936**	**4936**	5035	4985	5001	**4936**
500 cycles of random size between 3 and 10000	**4976**	5188	5475	5225	5463	**4976**
1000 cycles of random size between 3 and 10000	**4999**	5436	5998	5498	6450	**4999**
100 cycles of random size between 100 and 1000	**491**	496	590	541	567	**491**
500 cycles of random size between 100 and 1000	**551**	777	999	777	1319	994
1000 cycles of random size between 100 and 1000	**1049**	1091	1094	1102	2286	1199
100 cycles of random size between 10000 and 100000	**49958**	**49958**	50057	50008	49971	**49958**
500 cycles of random size between 10000 and 100000	**49904**	**49904**	50403	50153	50260	**49904**

1000 cycles of random size between 10000 and 100000	**49987**	**49987**	50986	50486	50921	**49987**
1000 cycles starting at 3 increments of 1	**1001**	**1001**	**1001**	**1001**	2244	**1001**
1000 cycles starting at 3 increments of 2	**1001**	1500	2000	1500	2691	1957
1000 cycles starting at 3 increments of 3	**1500**	1750	2499	2000	3105	2000
1000 cycles starting at 3 increments of 4	**2000**	**2000**	2999	2499	3613	**2000**
1000 cycles starting at 3 increments of 1000	**499502**	**499502**	500501	500001	499860	**499502**
1000 cycles starting at 1000 increments of 1	**1499**	1998	1998	1998	2784	1998
1000 cycles starting at 1000 increments of 2	**1499**	1999	2498	1999	3206	2476
1000 cycles starting at 1000 increments of 3	**1999**	2249	2998	2498	3664	2498

1000 cycles starting at 1000 increments of 4	**2498**	**2498**	3497	2998	4136	**2498**
1000 cycles starting at 1000 increments of 1000	**500000**	**500000**	500999	500500	500254	**500000**
500 cycles of size 100000 and 500 cycles of size 10	**50499**	**50499**	50999	50999	51854	**50499**
7 8 3	**5**	6	6	6	**5**	6
10 20 30	**15**	**15**	17	16	**15**	**15**
100 91 89 87 85 83 81	**50**	**50**	56	53	53	52
93 91 89 87 85 83 81	**47**	50	53	50	51	51

4.2.1 BroadcastBucket Results

The *BroadcastBucket* algorithm is by far the best performing algorithm out of the four described in chapter 3. The only test case where it did not have the best result was the test case 7 8 3, where the best result came from the *BroadcastRandom* algorithm. The reason why *BroadcastRandom* gave a better broadcast time than the *BroadcastBucket* algorithm is because in this test case the optimal broadcast time comes from visiting the second to largest cycle of the k-cycle graph first. The *BroadcastBucket* algorithm always visits the largest uninformed cycle first, which would be the best choice most of the time.

The *BroadcastBucket* algorithm performs better than any other algorithm because of the way it decides which round to visit each edge of each cycle of the k-cycle graph. It makes that greedy choice of visiting the largest uninformed cycle first, but also is decides the latest possible round it will visit the second edge of the cycle so that the cycle will finish broadcasting in the given time. That is why the algorithm gives very good broadcast times but as seen from the test case 7 8 3 it does not necessarily give the optimal bradcast time.

Comparing the results from the *BroadcastBucket* algorithm with the theoretical lower bounds described in chapter 3 we actually see that *BroadcastBucket* generates optimal broadcast times for some test cases. For example, if $l_j \geq l_{j+1} + 4$ for all $1 \leq j \leq k-1$, then the *BroadcastBucket* algorithm generates optimal results when we compare the results with the lower bounds. If $l_j = l_{j+1} + 2$ or $l_j = l_{j+1} + 3$ for all $1 \leq j \leq k-1$, the *BroadcastBucket* algorithm does not generate the optimal broadcast time. When all cycles of the k-cycle graph are the same length, $l_j = l_{j+1}$ for all $1 \leq j \leq k-1$ the *BroadcastBucket* algorithm generates the optimal broadcast time.

An interesting result we observed is when $l_j = l_{j+1} + 1$ for all $1 \leq j \leq k-1$. For the test case, *1000 cycles starting at 3 increments of 1*, we see that the *BroadcastBucket* algorithm generates an optimal broadcast time. But for this test case, *1000 cycles starting at 1000 increments of 1*, it does not generate the optimal broadcast time. The reason for this is because of the sizes of the cycles. When we have 1000 cycles starting at 3 and increment each cycle by one then most of the cycles will finish broadcasting before it's second edge starts broadcasting. When we increase the minimum sized cycle to 1000 then it obviously takes longer to broadcast because most cycles will have both of it's edges incident to the originator help in broadcasting for that cycle.

Another interesting result is when one of the cycles is larger than all the others by a certain amount. For example, this test case 100 91 89 87 85 83 81, gives an optimal broadcast time when comparing the broadcast time of the *BroadcastBucket* algorithm with the lower bound even though we know this algorithm does not perform optimally when cycles are separated by 2. The reason this test case returned an optimal result is because the first cycle is large enough to dominate the broadcast time for the graph, which makes all other cycles irrelevant for the total broadcast time. If we take the same test case but modify the 100 for a 93 we see that all cycles are separated by two and it does not produce an optimal broadcast time.

4.2.2 BroadcastGreedyMinMax Results

The *BroadcastGreedyMinMax* algorithm performs well on some test cases and poorly on others. When all cycles of the k-cycle graph are the same size then this algorithm gives optimal broadcast times. The reason for this is in the way it visits the edges of the cycles that are incident to the central originator vertex. Cycle C_i is visited on round i and on round $2k+1-i$. Take two consecutive cycles of the same size, the first cycle will start broadcasting on the first edge at time i and at time $2k+1-i$, and the second cycle will start broadcasting on its first edge at time $i+1$ and on the second edge at time $2k+1-(i+1)$. This means that cycle C_{i+1} will be visited on it's first edge one round after cycle C_i, but its second edge will be visited one round before the second edge of cycle C_i. This means that when all the cycles of the k-cycle graph are the same size they will all finish broadcasting at the same time.

This algorithm performs poorly when the cycles differ in size. The larger the cycles differ in size the worse the algorithm performs. Let's take for example the test case, *1000 cycles starting at 3 increments of 4*. This test case resulted in a poor total broadcast time when compared to the lower bound. Take the cycle C_i and C_{i+1}, instead of visiting the cycle C_i at round i and round $2k+1-i$ then cycle C_{i+1} at round $i+1$ and round $2k+1-(i+1)$, it would give a better broadcast time when visiting the cycle C_i at round i and $2k+1-i$ then cycle C_{i+1} at round $i+1$ and round $2k+1-i$. This way the larger cycle will finish it's broadcasting at a closer time to the smaller cycle.

We observed some interesting results from the simulations for this algorithm. From the three test cases, *100 cycles of random size between 3 and 100*, *500 cycles of random size between 3 and 100*, and *1000 cycles of random size between 3 and 100* we see optimal results when compared with the lower bound for when the number of cycles is 500 or 1000. The reason for this is because due to the pigeonhole principle. The difference between the sizes of the cycles are very small and a lot of them in fact are the same size. As the number of cycles increases there will be more cycles of the same size due to the bounds on the possible sizes of the cycles. In fact, when there are 500 or 1000 cycles it resulted in optimal broadcast times when we compare to the lower bound. This is because a lot of the cycles are the same size which is the strength of this algorithm.

As the cycle sizes increases the possible difference between consecutive cycles increases and the total broadcast time of the algorithm gets worse compared to the lower bound. Let's take for example, *100 cycles of random size between 10000 and 100000.* The difference between the possible sizes of the cycles are much higher than the total number of cycles for this k-cycle graph. This spaces out the cycle sizes more which is the weakness of this algorithm.

Another interesting result comes from the test cases, *1000 cycles starting at 3 increments of 1* and *1000 cycles starting at 1000 increments of 1.* When the cycle size starts at 3 and then increments by 1 for each consecutive cycle this algorithms actually results in an optimal broadcast time when we compare to the lower bound. When the cycle size starts at 1000 and increments by one for each consecutive cycle this algorithm does not result in optimal broadcast times. The reason for this is because of the sizes of the cycles. When starting at 3 and incrementing by 1 for 1000 cycles the largest cycle will be size 1002. As the first and second edge are assigned to each cycle when the cycle size gets small enough the round that we will start broadcasting on the second edge of a small cycle does not matter because the broadcasting on that cycle will finish from the first edge before we start broadcasting on the second edge. But when the cycle sizes start at 1000 and increment by 1 for each consecutive cycles the second edge does matter for the broadcast times of the smaller cycles.

4.2.3 BroadcastGreedyEven Results

Just like the $BroadcastGreedyMinMax$, the $BroadcastGreedyEven$ algorithm performs well on some test cases and poorly on others. This is because of the sizes of the cycles of the k-cycle graph and the rounds it assigns to visit the first and second edges of each cycle that are incident to the central originating vertex. Cycle C_i is visited on round i and on rounds $k+i$. Take two consecutive cycles of the same size, C_i and C_{i+1}. Cycle C_i will start broadcasting on the first edge at time i and at time $k+i$, and cycle C_{i+1} will start broadcasting on it's first edge at time $i+1$ and on the second edge at time $k+(i+1)$. The first edge of cycle C_i will be visited before the first edge of cycle C_{i+1}, and the second edge of cycle C_i will be visited before the second edge of C_{i+1}. This results in cycle C_i finish its broadcasting before cycle C_{i+1}. The more cycles of the same size there are the higher the broadcast time of the k-cycle graph will be.

This algorithm gives better broadcast times than the $BroadcastGreedyMinMax$ algorithm when the cycle sizes are different because of the way it visits the edges of the cycles. Because the first edge of cycle C_i will be visited before the first edge of cycle C_{i+1}, and the second edge of cycle C_i will be visited before the second edge of C_{i+1}, this gives more time for the larger cycles to finish broadcasting when compared to the smaller cycles.

We observed some interesting results from the simulation of this algorithm. Just like the $BroadcastGreedyMinMax$ algorithm, these two test cases, *500 cycles of random size between 3 and 100*, and *1000 cycles of random size between 3 and 100*, gave optimal results when compared to the lower bound. Since the cycles sizes are relatively small compared to the number of cycles, the broadcast time is dominated by the larger cycles and eventually when it gets to the smaller sized cycles the broadcasting is finished before the second edge of these cycles is visited. In the test case *100 cycles of random size between 3 and 100* the broadcast time is not optimal for this algorithm because the number of cycles is not large enough for the same results as when there are 500 or 1000 cycles. As we increase the size of the cycles the results are not optimal but are better than the $BroadcastGreedyMinMax$ algorithm which is what we expected.

This test case, *500 cycles of size 100000 and 500 cycles of size 10* resulted in the same broadcast time as the $BroadcastGreedyMinMax$ algorithm. The reason for this is because half of the cycles are a very large size compared to the other half of the cycles which are relatively small. The total broadcast time is dominated by the 500 cycles of size 100000. The *BroadcastGreedyEven* algorithm does not give good broadcast times when all cycles are the same size. But in this test case half the cycles are size 100000 and the other half are size 10. So if we compare the edge assignments for $BroadcastGreedyMinMax$ and $BroadcastGreedyEven$ on cycle $\frac{k}{2}$ which would be the highest broadcast time for the $BroadcastGreedyEven$ algorithm we see why they result in the same broadcast time. For cycle $\frac{k}{2}$, $BroadcastGreedyMinMax$ will have edge assignments $(\frac{k}{2}, 2k+1-\frac{k}{2})$, and $BroadcastGreedyEven$ will have edge assignments $(\frac{k}{2}, k+\frac{k}{2})$. Pluging in the value of $k = 1000$ we get for $BroadcastGreedyMinMax$ $(500, 1501)$ and for $BroadcastGreedyEven$ we get $(500, 1500)$. These edge assignments when used with $CycleBroadcastTimeHelper$ give the same result. If instead of 500 cycles of size 100000, we set 502 cycles of sie 100000 and 498 cycles of size 10, the $BroadcastGreedyMinMax$ algorithm gets a better broadcast time that the $BroadcastGreedyEven$ algorithm with times of 50999 and 51000 respectively.

4.2.4 BroadcastRandom Results

The *BroadcastRandom* algorithm resulted in very poor broadcast times compared with the lower bound and all other algorithms for most of the test cases. This algorithm works by randomly selecting the rounds that we visit each edge incident to the central vertex for each cycle of the k-cycle graph. This means if we are lucky we may get a good or even optimal broadcast time when we compare with the lower bound. We ran this algorithm 100000 times for each test case and keep the best broadcast time.

This algorithm actually gave an optimal broadcast time for this test case 7 8 3, when all other algorithms did not produce the optimal broadcast time. The reason is that all the algorithms make a greedy choice of visiting the largest cycle first, which seems to be the logical choice. But in this test case we actually get an optimal time when we visit the second to larges cycle first. This is a surprising result.

The problem with this random approach is that as the number of cycles increases the number of possible edge round assignments grows very fast. In fact there are $2k!$ possible edge assignments, where k is the number of cycles. Even when $k = 5$ this gives a total of 3628800 possible edge round assignments for the cycles. So even running this algorithm 100000 there is a small chance of finding the optimal broadcast time. This is the reason for the poor performance of the algorithm.

Chapter 5

Conclusions and Future Work

In this thesis, we reviewed broadcasting in cactus graphs and subclasses of cactus graphs such as the unicyclic graphs, necklace graphs, k-cycle graphs, 2-restricted cactus graphs and k-restricted cactus graphs. We then defined four heuristic algorithms that solve broadcasting on a subclass of the cactus graph called the k-cycle graph. After running simulations on the four algorithms, *BroadcastBucket*, *BroadcastGreedyMinMax*, *BroadcastGreedyEven* and *BroadcastRandom* we concluded that the *BroadcastBucket* algorithm gives the best broadcast times out of all the algorithms on nearly all sized k-cycle graphs. It actually gives optimal broadcast times for them the cycles are the same size or when consecutive cycles differ by 4 or more. The *BroadcastGreedyMinMax* algorithm gives optimal broadcast times when the cycles of the k-cycle graph are the same size and good broadcast times when the cycles are very close to the same size. When the cycles are different sizes on the k-cycle graph the *BroadcastGreedyEven* algorithm gives better broadcast times that the *BroadcastGreedyMinMax* algorithm but still not as good as the *BroadcastBucket* algorithm. We also showed from the simulations that the *BroadcastRandom* algorithm will give bad broadcast times when the number of cycles of the k-cycle graph is large. In the simulation results we compared the resulting broadcast times of each algorithm with the theoretical lower bound for broadcasting on a k-cycle graph and saw that for some test cases on certain algorithms we did get an optimal broadcast time.

In the future, we can extend the *BroadcastBucket* algorithm to create a new polynomial algorithm that can not only give good broadcast times for the k-cycle graph but can give good broadcast times for a general cactus graph. We can also refactor the *BroadcastBucket* algorithm to give better broadcast times when consecutive cycle sizes differ by 1, 2 or 3. An approximation algorithm can be created to figure out how well the *BroadcastBucket* algorithm is performing compared to the optimal broadcast time for a general k-cycle graph.

Bibliography

[1] A. Bar-Noy, S. Guha, J. Naor, B. Schieber. *Multicasting in Heterogeneous Networks.* Proc. of ACM Symp. on Theory of Computing, STOC98, 1998.

[2] B. Ben-Moshe, B. Bhattacharya; Q. Shi, *Efficient algorithms for the weighted 2-center problem in a cactus graph*, Algorithms and Computation, 16th Int. Symp., ISAAC 2005, Lecture Notes in Computer Science, 3827, Springer-Verlag, (2005), pp. 693703

[3] J.C. Bermond, H.A. Harutyunyan, A.L. Liestman, S. Perennes. *A note on the dimensionality of modified knödel graphs.* In IJFCS: International Journal of Foundations of Computer Science, volume 8, pages 109117, 1997.

[4] P. Bhabak, H.A. Harutyunyan. *Constant Approximation for Broadcasting in k-cycle Graph.* CALDAM 2015: 21-32

[5] M. Čevnik, J. Žerovnik. *Broadcasting on Cactus Graphs*, Journal of Combinatorial Optimization, January 2017, Volume 33, Issue 1, pp 292316

[6] F. Comellas, C. Dalfo. *Optimal broadcasting in 2-dimensional manhattan street networks.* J. Parallel and Distributed Computing and Networks, 4:3751, 2003.

[7] M. Elkin, G. Kortsarz. *Sublogarithmic approximation for telephone multicast: path out of jungle.* Proc. of Symposium on Discrete Algorithms, SODA03, Baltimore, Maryland, pp. 7685, 2003.

[8] M. Elkin, G. Kortsarz. *A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem.* Proc. of ACM Symp. on Theory of Computing, STOC02, pp. 438447, 2002

[9] U. Feige, D. Peleg, P. Raghavan, E. Upfal. *Randomized broadcast in networks.* Proc. of International Symposium on Algorithms, SIGAL90, pp. 128137, 1990.

[10] G. Fertin, A. Raspaud. *A survey on knödel graphs.* Discrete Appl. Math., 137(2):173195, 2004.

[11] G. Fertin, A. Raspaud, H. Schroder, O. Sykora, I. Vrto. *Diameter of the knödel graph.* In Proceedings of the 26th International Workshop on Graph- Theoretic Concepts in Computer Science, WG 00, pages 149160, 2000.

[12] P. Fraigniaud, S. Vial. *Approximation algorithms for broadcasting and gossiping.* J.Parallel and Distrib. Comput. 43(1):4755, 1997.

[13] P. Fraigniaud, S. Vial. *Comparison of Heuristics for One-to-All and All-to-All Com- munication in Partial Meshes.* Parallel Processing Letters, 9(1), pp. 920, 1999.

[14] P. Fraigniaud, E. Lazard. *Methods and Problems of Communication in Usual Networks.* Discrete Appl. Math, 53:79133, 1994.

[15] Harary, F., Uhlenbeck, G. E., *On the number of Husimi trees I*, Proc. Nat. Acad. Sci.U. S. A. 39 (1953), 315322.

[16] H.A. Harutyunyan. *Multiple message broadcasting in modified knödel graph.* In SIROCCO 00, pages 157165, 2000.

[17] H.A. Harutyunyan, C.D. Morosan. *On the minimum path problem in knödel graphs.* Networks, 50(1):8691, 2007.

[18] H.A. Harutyunyan, E. Maraachlian. *Linear algorithm for broadcasting in unicyclic graphs.* In Guohui Lin, editor, COCOON, pages 372382. Springer, 2007.

[19] H.A. Harutyunyan and E. Maraachlian. *Broadcasting in optimal bipartite double loop graphs.* In Proceedings of the conference on Information Visualization, IV 06, pages 521528. IEEE Computer Society, 2006.

[20] H.A. Harutyunyan, E. Maraachlian. *Broadcasting in fully connected trees.* In Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems, ICPADS 09, pages 740745, Washington, DC, USA, 2009. IEEE Computer Society.

[21] H.A. Harutyunyan, G. Laza, E. Maraachlian. *Broadcasting in necklace graphs.* In Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, C3S2E 09, pages 253256. ACM, 2009.

[22] H. A. Harutyunyan, E. Maraachlian, *Linear algorithm for broadcasting in networks with no intersecting cycles.*, International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA09, 296301, (2009).

[23] K. Husimi, *Note on Mayers' theory of cluster integrals* Journal of Chemical Physics, 18 (5) (1950): 682684.

[24] S.T. Hedetniemi, S.M. Hedetniemi, A.L. Liestman. *A Survey of Gossiping and Broadcasting in Communication Networks.* Networks, 18(4):319349, 1988.

[25] S.T. Hedetniemi, R. Laskar, J. Pfaff, *A linear algorithm for finding a minimum dominating set in a cactus*, Discrete Appl. Math., 13 (1986), pp. 287292

[26] J. Hromkovic, R. Klasing, B. Monien, R. Peine. *Dissemination of informa- tion in interconnection networks.* Combinatorial Network Theory, pages 125 212, 1996.

[27] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzicka, W. Unger. *Dissemination of information in communication networks: broadcasting, gossiping, leader election, and fault-tolerance.* Texts in Theoretical Computer Science, An EATCS Series. Springer, 2005.

[28] D. Johnson, M. Garey. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979

[29] N.Khan, M.Pal, A.Pal, *L(0,1)-labelling of cactus graphs*, Communications and Network, 4 (2012) 18-29.

[30] N. Khan, M. Pal, A. Pal, *Edge colouring of cactus graphs.* Adv. Model. Optim 11(4), 407421 (2009)

[31] W. Knödel. *New gossips and telephones.* Discrete Mathematics, 13:95, 1975.

[32] N. M. Korneyenko, *Combinatorial algorithms on a class of graphs*, Discrete Applied Mathematics, 54 (23): (1994), 215217

[33] G. Kortsarz, D. Peleg. *Approximation algorithms for minimum time broadcast.* SIAM J. Discrete Math. 8:401427, 1995.

[34] A. Liestman, J. Peters. *Broadcast networks of bounded degree.* SIAM J. Discrete Math, 1998.

[35] K. Maity, M. Pal, T.K. Pal, *An optimal algorithm to find all-pairs shortest paths problem on weighted cactus graphs*, V.U.J. Physical Sciences, 6 (2000) 45-57

[36] E. Maraachlian. *Optimal Broadcasting in Treelike Graphs*, PHD Thesis, Computer Science Department, Concordia University, pp. 77-81, 2010.

[37] M. Markov, M. Ionut Andreica, K. Manev, N. Tapus, *A linear time algorithm for computing longest paths in cactus graphs.* Serdica Journal of Computing, 6(3), 287298, (2012).

[38] B. Paten, M. Diekhans, D. Earl, J. St. John, J. Ma, B. Suh, D. Haussler, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, Lecture Notes in Computer Science, (2010) 6044: 410425

[39] R. Ravi, *Rapid Rumor Ramification: Approximating the minimum broadcast time.* Proc. of 35th Symposium on Foundation of Computer Science, FOCS94, pp. 202 213, 1994.

[40] P. J. Slater, E. J. Cockayne, S. T. Hedetniemi. *Information dissemination in trees.* SIAM J.Comput. 10(4), pp. 692701, 1981.

[41] Wikipedia: the free encyclopedia. *Fisher-Yates Shuffle.* available at https://en.wikipedia.org/wiki/Fisher-Yates_shuffle.

[42] B. Zmazek, J. Zerovnik, *Estimating the traffic on weighted cactus networks in linear time.* International Conference on Information Visualisation, 536541, (2005).