Approximation Algorithms for Broadcasting in Flower Graphs

Anne-Laure Ehresmann

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

September 2021

Abstract

Approximation Algorithms for Broadcasting in Flower Graphs

Anne-Laure Ehresmann

Over the last century, telecommunication networks have become the nervous system of our society. As data is generated and stored on varied nodes, effective communication is imperative to ensure efficient use of the network. Our ever-growing reliance on these increasingly large and complex networks make ineffective communication strategies evermore apparent.

Broadcasting is a fundamental information-dissemination problem which models communication across a connected graph in the following manner: a single vertex, the *originator*, seeks to pass some message along to all other vertices in the graph. In general, research on broadcasting can be grouped in roughly two categories: Firstly, given some particular graph and some particular vertex chosen to be originator, what is a broadcast scheme that informs the entire graph in the minimum time possible? Secondly, given some number of nodes, how can we arrange them in a particular network topology such that we can achieve minimal broadcast time from any vertex? This thesis focuses on problems of the first category. Finding the minimum broadcast time of any vertex in an arbitrary graph is NP-Complete, but efficient algorithms have been found for particular graph families. In particular, polynomial time algorithms have been found for trees and some tree-like graphs: unicyclic graphs, tree of cycles. Such algorithms have also been found for some graphs with no intersecting cliques, such as fully connected trees and trees of cliques. Finally, graphs containing cycles with particular restrictions were also studied, and efficient algorithms for necklace graphs and k-restricted cactus graphs were also found. The question still stands however, of whether these restrictions may be too conservative, and that efficient algorithms exist on broader classes of graphs. In particular, significant research has been made towards finding an efficient broadcasting algorithm on cactus graphs, which has not been found so far.

This thesis studies the broadcasting problem on *Flower* graphs, which capture the difficulty of cactus graphs in a simple graph family. Flower graphs, or k-cycle graphs, are graphs composed of k cycles all joined on a single central vertex v_c. The contributions of this thesis for broadcasting on flower graphs is two-fold: it first improves the approximation ratio for broadcasting on flower graphs. It then provides a heuristic which performs significantly better in practice than the current best heuristic. We also demonstrate that our heuristic finds the optimal broadcast time for particular subcases of flower graphs.

Acknowledgments

There are several people without whom this thesis would not have been made possible, and to whom I would like to express my sincere gratitude.

I would like to first deeply thank my professor and supervisor, Dr. Harutyunyan, for the guidance and encouragement he provided during my completion of this research, and during my time at university as a whole. It is his enthusiasm and genuine interest in his courses that led me to pursue academic research in the first place. His continued efforts to support and advise me throughout these past two years have been invaluable in completing this thesis and in my formation in academic research.

I would also like to thank Dr. Narayanan, who gave me my first research project and supervised my work on it during my third summer at university. This led to a small contribution to a simple problem with a reachable solution, yet with enough freedom to let me explore and challenge myself. She provided just the right amount of direction, personal enthusiasm, and collaboration to captivate and develop my interest in research.

It would be amiss not to thank those involved in open science initiatives, who seek to make access to scientific knowledge available to all without restrictions. Without their incessant efforts, I could not have completed this thesis.

Last but not least, I deeply thank my family, whose unconditional love, support, and patience have accompanied me throughout my entire life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation behind the problem

As our society increasingly turns to computers to solve its problems, the demands for additional computing power grow more and more ravenous. A single computer, even with our most technologically-advanced hardware, can do little but chip away at some of the colossal tasks we throw at it. Instead, a common solution is to break the task into smaller but more manageable constituent parts, and solve each of them simultaneously on extensive computing networks. Once each individual part is completed, their results are merged into a solution to the original problem. But many crucial decisions need to be taken before one can even attempt this solution, all of which may tremendously affect the performance of the network when solving some particular computing problem. Because of this, computing networks have appeared in very diverse forms: numerous fields with radically different needs and constraints make use of them, and the networks they employ thus appear in equally diverse forms that seek to address each of these needs and constraints. Just a few examples of computing networks [28] [8] can be found in global telecommunication networks, cloud-computing networks on the internet, parallel computing networks for distributed information-processing and scientific computing, and industrial control systems for automation. When confronted with a large problem to be solved by a computing network, some questions to consider are the following[35]:

- Given a particular problem, how do we decompose it?
- How do we split our computing resources across the network?
- What sort of memory model will we use?
- What sort of topology will we use?

- How do each node in the network communicate with each other?

- What algorithm can we use, such that we take full advantage of our computing network?

These are not easy decisions to make, especially if our problems come in varied forms. One can additionally quickly see that answering one question will additionally affect our answer for other questions: for instance, the topology of the network directly affects the management of computing tasks and communication of data across the nodes of the network, as data movements are entirely dependent on the topology [13]. Similarly, the communication strategy we choose will drastically affect which algorithms we can execute, and their performance. Overall, a single poor decision can result in bottlenecking the entire process, and thus fail to take full advantage of the resources provided by the network.

Our ability to make these decisions soundly is becoming even more crucial today: as the scale and quantity of our computational problems grow, so does the complexity of the networks that we use to solve them, as well as the emergent difficulties of parallel computing itself. More recently, the rapid explosion of big data [8] has drastically increased the demand for efficient, powerful, and fast computing power, and widespread computing networks are the only ones capable of providing the huge storage and computing resources required in an acceptable timeframe.

One of the above-cited concerns was that of communication across the network. Such communication tasks can be naively thought to require a negligible amount of work when compared to the actual computing task itself, but a number of factors mean that a poor communication strategy will lead to a poor performance overall [1]:

- Due to the high computing performance of current processors, communication is often more time-consuming than the computing task itself [24], and this bottleneck is worsening; Technologically-speaking, our capacity to telecommunicate is growing about half as fast as our capacity to complete computing tasks [23].
- A large number of problems on the internet are purely communication tasks: that is, we merely seek to disseminate some information as quickly and cheaply as possible [24].
- Our technologies are creating datasets of ever-increasing sizes, with created and copied data volume in the world predicted to double at least every two years [8].

The speed at which nodes can communicate some data to other nodes thus has a huge effect on the performance of the entire network for a given problem. We can separate the approaches to solve this bottleneck into roughly two categories [36]: The first category seeks to reduce the quantity of data that needs to be transferred through various data-reduction techniques. The second seeks to reduce the delay involved in sending a message. This thesis is focused on the second approach.

1.2 Statement of the problem

Communication strategies have been studied under different models, the majority of which are surveyed in [22]. Many of these models seek to accurately represent real-world conditions by simulating the parameters and difficulties that come with simply seeking to pass a message across the network, such as: How many nodes in the network start with the message, and how many (and which) nodes need to receive this message? How large is the message? How long does it take to traverse the connection from one node to other? How reliable are the channels between the nodes, and what can we do in case of faulty channels? How do we handle an incomplete knowledge of the topology of the network? How can we take advantage of the fact that we may be able to send a message to multiple other nodes at the same time? etc. [22]. However, numerous communication problems remain exceedingly difficult even on the simplest models, and these hence remain the most studied. In particular, the following two models are heavily studied:

- *Gossiping*: all-to-all information dissemination with each call involving exactly two nodes and no node participating in more than one call at a time.
- *Broadcasting*: one-to-all information dissemination with each call involving exactly two nodes and no node participating in more than one call at a time.

This thesis focuses on broadcasting. Broadcasting has a wide number of practical applications, but is also an interesting problem purely theoretically as it remains one of the simplest communication problems, yet still remains exceedingly difficult on some networks. In practice, in addition to applications in distributed networks mentioned earlier, broadcasting is often particularly relevant on LANs, P2P networks, satellite networks, and remains a crucial operation for the general maintenance of the internet[41].

Broadcasting [40] seeks to represent one-to-all information dissemination in a connected graph $G(V,E)$ where each vertex represents a particular computer or processor in the network, and each edge represents a communication link between two such computers. In the typical graph model used for broadcasting, only one edge between two vertices is allowed. In this problem, an *originator* vertex u is informed with some particular message, and it needs to disseminate that message to all other vertices in the graph. This process is achieved by making a series of calls between vertices subject to the following constraints:

- Each call takes exactly one unit of time.

- Each call must be between two neighbouring vertices.
- Each call must involve at least one informed vertex.
- Each vertex can participate in only one call per unit of time.
- In one unit of time, multiple calls can be performed in parallel.

Once a vertex has been informed with the message at some particular unit of time, it may, after this unit, disseminate the message to any uninformed neighbours it may have. These themselves may disseminate it to their own neighbours, and so on until all vertices in the graph have been informed with the message. Such a series of calls, provided the above restrictions are respected and all vertices are informed by the end of this series, is called a *broadcast scheme* of u.

The minimum number of time units required to completely inform all vertices of the graph G when starting from some originator u is referred to as the *broadcast time* $b(u,G)$ of u, often shortened merely to $b(u)$. The maximum broadcast time of any vertex $u \in V(G)$ is defined as the *broadcast time* $b(G)$ of G.

Finding the broadcast time [40] of any vertex u in an arbitrary graph has been shown to be NP-Complete by Slater et al., who then also show that we can determine the broadcast time of any vertex in a tree, as well as the entire tree itself, in linear time.

Scheuermann and Wu [36] give a nonpolynomial-time dynamic programming formulation for determining $b(u)$ for any u in an arbitrary graph, but observing the obvious computational inefficiency of their solution, additionally gave heuristics for obtaining efficient near-optimal broadcast schemes. A backtracking algorithm based on this formulation was later found [37].

With respect to approximation, Schindelhauer showed in [38] that it is NP-hard to approximate $b(u)$ within a factor of $57/56 - \varepsilon$ for arbitrary $\varepsilon > 0$. Elkin and Kortsarz improved this to a factor of $(3-\varepsilon)$. They then give the current best approximation algorithm for finding $b(u)$ with $O\big(\frac{\log(|V|)}{\log\log(|V|)}\big)$ approximation, and which runs in $\tilde{O}(|E||V|)$.

Since we are limited in how well we can approximate the broadcast time of an arbitrary graph, but know that there exists a polynomial-time algorithm for solving the broadcasting problem for a particular graph family (trees), this spurs a number of questions, which are studied by a significant part of the literature on broadcasting; That is, given a particular graph family,

- What is the broadcast time of any graph in this family?
- Can we efficiently find the broadcast time of any graph in this family, or is this problem NP-Complete as well?

- If the former, how efficiently can an algorithm return an optimal broadcast scheme of any graph in this family?
- If we fail to answer the above questions or if finding the broadcast time of any graph in this family is NP-Complete, how well can we approximate the broadcast time of any graph in this family?

These questions have been answered for a number of graph families, and some of the major ones are discussed in Section 2.

There exists another main category of research in the broadcasting literature based on the following observation: on a graph G of n vertices, $b(G) \geq \lceil \log(n) \rceil$, since when starting with a single informed vertex, we can have at most 2^t informed nodes after t rounds. This happens if every vertex that was informed during some previous round is able to make a call to an uninformed vertex every round hereafter until the entire network is informed. If a graph G achieves $b(G) = \lceil \log(n) \rceil$, we refer to it as a *broadcast graph*. A simple way to design a broadcast graph is of course to use a complete graph K_n, but observe that we can carefully remove some edges from K_n and still maintain this optimal broadcast time. This spurs the following criteria [12] to determine how effective G is with respect to the problem of broadcasting:

- Is G a broadcast graph?
- Is G using the minimum number of edges required to maintain its $b(G)$ value?

If we manage to find a broadcast graph which optimally satisfies these two criteria, that is, for n vertices, it has the fewest quantity of edges required to broadcast in $\lceil \log(n) \rceil$ units of time, we instead call this graph a *minimum broadcast graph* (*mbg*). This quantity of edges for a particular number of vertices n is known the broadcast function $B(n)$. A significant part of the literature focuses on finding these values of $B(n)$ and designing such mbgs. See [22] for a survey on this. Such graphs have been found for all values of $n < 33$ except $n = 23$, 24, and 25 (see for instance [11][29][26][15] and their respective bibliographies) for several other values of $n < 128$, for $n = 58$, 59, 60, 61, 63, 127[15], 1023, 4095[39]. Additionally, two graph families were identified as minimum broadcast graphs: the hypercube with $n = 2^k$ for $k \geq 1$ [11], and the Knödel graphs with $n = 2^k - 2$ for $k \geq 2$ [3]. In general, finding these graphs has proven to be very difficult. As a result, part of the research also has sought to construct broadcast graphs which still succeed in using a relatively small number of edges, and also obtaining upper bounds on values of $B(n)$ in the process. To achieve this, most techniques combine known smaller *mbg*'s to generate a larger *bg*. See for instance [20] which relies on the Knödel graphs for their graph construction.

Since this thesis does not focus on minimum broadcast graphs and their related problems, this area of research, while directly related to ours, will not be covered further.

The high-level organisation of the thesis is summarised as follows:

First, Chapter 2 reviews the current literature on broadcasting in various relevant graph families. Special attention is given to cactus graphs and flower graphs, two families with as of yet no existing polynomial algorithm for obtaining the broadcast time of any graph in these families. Then, Chapter 3 provides a simple approximation algorithm for achieving a 1.5-approximation ratio on flower graphs. It is then shown that a small intuitive change can be made on this approximation algorithm to create a new algorithm which maintains the previous approximation ratio, but achieves better results in practice. Chapter 4 gives a heuristic using similar intuitions as the aforementioned small change. Chapter 5 provides simulations to compare the performances of the new algorithms along with the ones in current literature, and a discussion on their differences and weaknesses follows. Chapter 6 discusses future work.

Chapter 2

Literature review

2.1 Broadcasting in various graph families

Polynomial-time algorithms for finding the exact values of $b(u)$ were found for a number of useful graph families [14][4]. In this section, we start from the most basic graph structure studied under the broadcasting model, the tree, and from there cover more complex structures until reaching the particular graph family with which this thesis is concerned.

2.1.1 The Tree T

While a simple structure, it is useful to consider the broadcasting problem on trees, as the strategy used there can also be applied on other graph families.

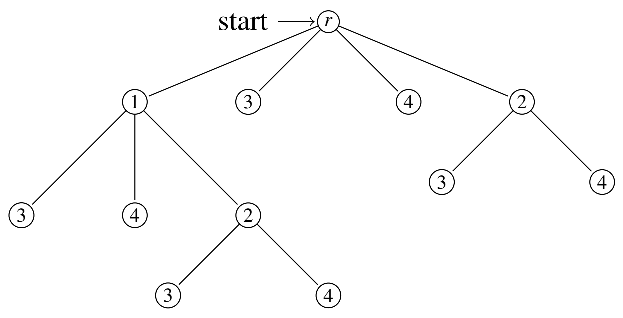


Figure 2.1: Broadcasting in a tree.

A tree T of n nodes and $n-1$ edges is a connected acyclic graph. Given any tree T, a linear algorithm provided in [40] returns the *broadcast center* of the tree. The *broadcast center* $BC(G)$ of a graph G is the set of all vertices with minimum broadcast time in G. In [40], Slater et al. proved the following two lemmas:

Lemma 1: *For any tree T with $|V(T)| \geq 2$, $BC(T)$ consists of a star with at least two vertices.*

Lemma 2: *For any vertex v in a tree T but not in $BC(T)$, let k be the minimum distance from v to any vertex in $BC(T)$. Then, $b(v) = k + b(BC(T))$*

They combined these lemmas in an algorithm which determines the broadcast time of a single vertex in T's broadcast center: starting from its leaves, it labels each vertex with what can be considered to be a "local" broadcasting time: the time it takes to broadcast in that vertex and all its children, that is, vertices considered in previous iterations. This local broadcasting time can be obtained by sorting; Then, it removes all leaves of the graph, and moves on to the next iteration, where again it considers the leaves of the newly modified graph, and so on. Special cases are to be considered when reaching the last iteration to ensure exactly one vertex remains, but broadly speaking, the algorithm behaves in the above manner during all other iterations.
From the broadcast time of that vertex, the broadcast time of all other vertices can easily be obtained using the above lemmas.

2.1.2 The Cycle C_n

A cycle on n vertices is a trail $v_1, v_2, \dots v_n$ where v_i has an edge to v_{i+1} for every $1 \leq i < n$, and v_n has an edge to v_1. We can observe that the broadcast time of C_n is directly related to its diameter $D(G)$, that is, the maximum distance between any two nodes of G. In cycles, $D(G) = \lfloor \frac{n}{2} \rfloor$, and $b(G) = \lceil \frac{n}{2} \rceil$.

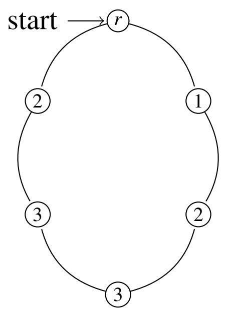


Figure 2.2: Broadcasting in a cycle.

2.1.3 The Unicyclic graph

A unicyclic graph G of n nodes and n edges is a connected graph. Its cycle of length k is labelled C_k and composed of vertices $r_1, r_2, \dots r_k$, each a root of a subtree of G.

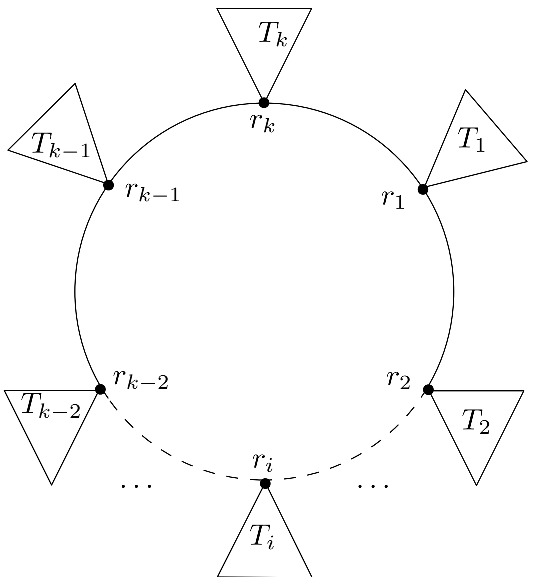


Figure 2.3: A Unicyclic graph.

Unicyclic graphs were studied under the context of broadcasting in [18], where a linear algorithm was given for finding their broadcast time. Harutyunyan et al. show that for any unicyclic graph, this broadcast time can be obtained by calculating the broadcast time of each substree, and finding the vertex called $SBC(T)$ in each tree's broadcast center that is closest to the root of the tree.

They then demonstrate that they can use the broadcast center of each subtree to calculate the broadcast time and broadcast center of any two trees T_1 and T_2, whose roots are joined by a single edge into a new tree $T_1 \oplus T_2$. They do so using the following result:

Lemma 3: *There exists a vertex u such that $u \in BC(T_1 \oplus T_2)$and u is on the path joining $u_1 = SBC(T_1)$ and $u_2 = SBC(T_2)$*

The above lemma as well as summations based on the difference between $B(T_1)$ and $B(T_2)$, and the distance between each original tree's respective distance between their root their *SBC* can be used to determine $b(T_1 \oplus T_2)$ and $BC(T_1 \oplus T_2)$ in constant time.

With this, [18] can slowly join the trees into all possible spannings trees possible, determine their broadcast time and broadcast center, find the distance from the originator *u* to the broadcast center of each spanning tree, and use that to find the minimum broadcast time of the entire unicyclic graph in $O(n)$ time.

2.1.4 The Tree of Cycles

Pushing further into this category of graphs, we can add additional cycles but ensure they never share a vertex. The *Tree of Cycles* is a graph with no intersecting cycles [19]. That is, there is only one unique path between any two cycles. They can thus be visualised as a sort of tree of cycles.

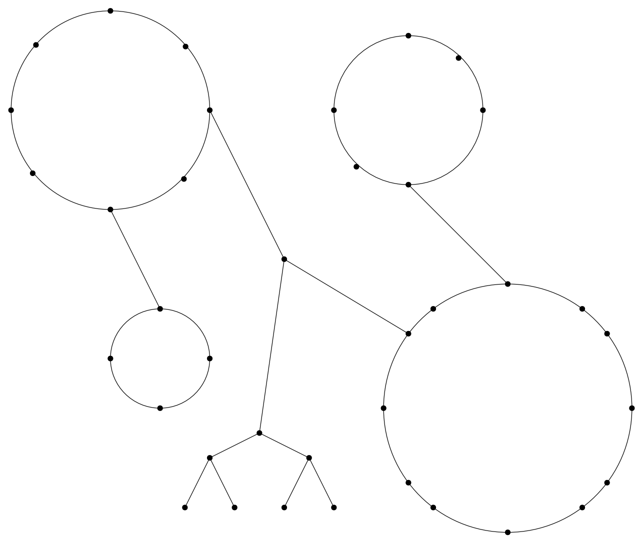


Figure 2.4: A tree of cycles graph.

The linear algorithm for unicyclic graphs implies that a polynomial algorithm exists for this graph family as well. Indeed, an optimal broadcast scheme in $O(n)$ was given in [19] with an algorithm that, in short, solves the problem recursively by considering each cycle individually when one is encountered, and then using the obtained broadcast tree for that broadcast cycle as a part of the overarching broadcast tree.

2.1.5 The Cactus graph

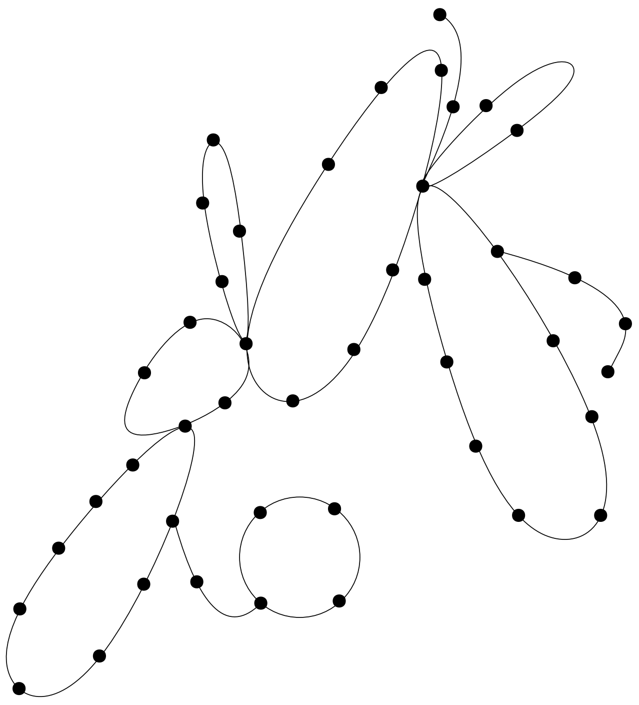


Figure 2.5: A Cactus graph.

As shown above, the broadcasting problem can be solved in graphs with no intersecting cycles in linear time. What about when cycles do intersect? It appears that it is difficult to find optimal broadcast schemes in polynomial time for such general graph families. Instead, when such algorithms are found, it is often in families with numerous restrictions on them, a few of which we present below. In particular, one family of graphs of interest is that of *cactus graphs*: graphs where cycles share at most one vertex in common [25]. Cactus graphs have numerous applications: initially known as Husimi trees [25], they first showed their use in the theory of condensation in statistical mechanics [42] [34] alongside Cayley trees. More recently, cactus graphs have been used in genome comparisons [32] and genomic alignments [33] in computational biology. Particular subcases of cactus graphs are also used: For instance, Christmas cactus graphs (connected cactus graphs for which the removal of any vertex disconnects the graph into at most 2 components) are used in greedy embeddings [27]. It is thus not only useful to investigate this family of graphs in the context of broadcasting for theoretical results, but also for practical applications.

Intuitively, one may think that the restriction of cactus graphs, that of cycles sharing at most one vertex, is easy to work with. This intuition comes from the fact that numerous other problems have been shown to have polynomial-time algorithms on cactus graphs, and that numerous algorithms that function on trees can be generalised to also work on cactus graphs. For instance, the following can be solved in polynomial time:

- Recognising a cactus graph [6].
- Finding a minimum dominating set in linear time [21].
- Computing the longest paths in linear time, in [31].
- Numerous facility-location problems: The *obnoxious center problem* on a weighted cactus graph in $O(cn)$ time on a cactus graph of n vertices and with c different vertex weights [43]. Improved to $O(n\log^3(n))$ in [2]. The *weighted 1-center* and *weighted 2-center problem* in $O(n\log(n))$ and $O(n\log^2(n))$, respectively, in [2].
- Computing the sum of all distances, using the generalisation of an algorithm originally designed for trees[44].

However, no polynomial-time algorithm for finding the broadcast time of general cactus graphs has been found, nor is it known if this is an NP-Hard problem. Instead optimal algorithms were found for subclasses of cactus graphs (observe additionally that the above *tree of cycles* is a subcase of cactus graphs).

2.1.6 The Necklace graph

Necklace graphs [17] are a specific subcase of cactus graphs, composed of a series of cycles (of arbitrary length) where each consecutive pair of cycles is connected by exactly one vertex. We highlight that the first and last cycles are not connected: This case, what we may call a *closed* necklace graph, was discussed but not solved in [17] for the broadcast problem.

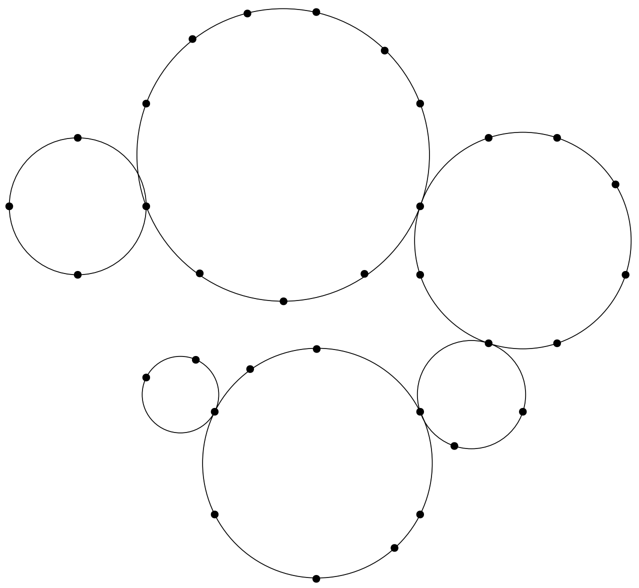


Figure 2.6: A necklace graph on 7 cycles.

In [17], Harutyunyan et al. give an $O(n)$ algorithm for finding a broadcast scheme in an arbitrary necklace graph, assuming the originator is on one of the end cycles and is not a vertex shared by two cycles. The cases when the originator is not one such vertex are built upon this case, and are given in [30].

To obtain their broadcast time, they first give a way of calculating the broadcast time of a graph composed of a cycle with a tree attached to one of its vertices.

They then show that they use the two key results:

Lemma 4: *For any originator u on a cycle C_N and some graph G' attached to a vertex v on C_N (where $u \neq v$), a broadcast scheme can be obtained by first sending the information along the shorter path towards v and then along the longer path.*

Lemma 5: *Let G' be the subgraph of G obtained by removing one of the end cycles of G (that does not contain the originator u). Broadcasting on G from u using, as part of our broadcasting routine,* *a broadcast scheme on G' that takes strictly greater than $b(G')$ rounds to complete, cannot result in a broadcast scheme that uses fewer rounds than a broadcasting scheme that does use an optimal broadcasting scheme on G'.*

In other words, one can use an optimal solution to the subproblem (broadcasting on a subgraph of the necklace graph) to build an optimal solution for the entire graph.

Using the first lemma, [17] give a linear algorithm for determining the broadcast time of a cycle with a tree attached, and then use it and the second lemma to construct optimal broadcast graphs to subgraphs of the necklace graph: They cut the furthest cycle (with respect to the originator's cycle) at the midpoint, and then consider the second-to-last cycle, now with a tree of two paths of same (or off by one) size attached to one of its vertices. They find the broadcast time of this cycle, and repeat the process on this cycle, then considering the broadcast time of the third-to-last cycle, and so on until a spanning broadcast tree for the entire graph is obtained.

2.1.7 The k-restricted Cactus graphs

k-restricted cactus graphs are cactus graphs where every vertex is on at most k cycles.

Some theoretical results were obtained for 2-restricted cactus graphs by the authors of [17] in [30], but no full algorithm was constructed. They demonstrated that the strategy used to obtain a broadcast scheme on necklace graphs can be generalised to 2-cactus graphs: That is, if one considers a cactus graph G to be a cycle C on i vertices with N_i 2-restricted cactus graphs as subgraphs rooted on each vertex on C, then provided that one can obtain a broadcast tree for each subgraph, one can construct a broadcast tree for the entire graph G. However, they were unable to obtain these trees in polynomial time.

In [7], a $O(kk!n)$ algorithm is given by Čevnik et al. for finding the broadcast time of *k-restricted cactus graphs* using the above logic. When k is assumed to be constant, this thus reduces down to a $O(n)$ algorithm. Observe additionally that the case for *closed* necklace graphs mentioned above can thus be solved in $O(n)$ running time using this algorithm, since necklace graphs have $k = 2$.

In [7], the authors first give an algorithm which works backward on a DFS-ordered cactus graph (rooted at the originator u), slowly determining what they define to be the *temporary broadcast time* of a vertex, knowing the temporary broadcast time of that vertex's children, if any. The final temporary broadcast time calculated, that of u, the first vertex in the DFS order, coincides with the broadcast time of u in the entirety of G. They then give their algorithm to determine the broadcast time of the entire graph while retaining linear running-time, instead of the naive $O(n^2)$. I refrain from summarizing their algorithm here, as it is too long to accurately represent.

A key point to retain from the algorithm provided in [7] is where it gets the $kk!$ factor from: when dealing with finding a vertex shared by multiple *cycle-like components* (a cycle C and all subcacti rooted at any vertex in C), they provide no strategy for deciding the order of which component to broadcast to, and instead merely check every possible order of components (at most, $k!$ possible orders). Since for each order, linear subalgorithms need to be called for each component (at most k components), then they obtain their $kk!$ running-time.
We can thus see that the area of interest for solving the broadcast problem on cactus graph is: how do we pick which cycle to broadcast to, when broadcasting from a vertex that is shared to multiple cycles?

2.1.8 The Flower graph

Flower graphs, also called k-cycle graphs, seem to capture the difficulty found in cactus graphs: They are graphs of k cycles of arbitrary length, all connected to one central vertex v_c. One may initially assume flower graphs to be simple to broadcast in, especially from the central vertex. And yet, no polynomial-time algorithm has been found for broadcasting on flower graphs, nor is it known if it is an NP-Hard problem. To solve this problem, attempts have been made at first finding the broadcast time of the central vertex v_c in polynomial time.

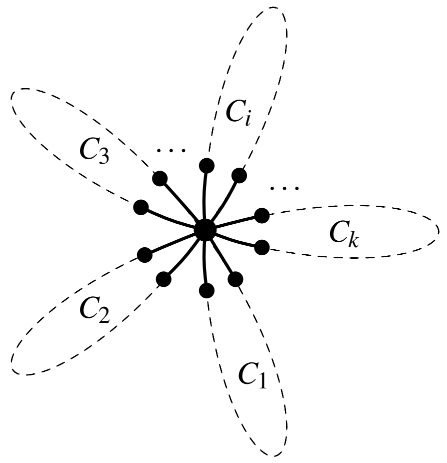


Figure 2.7: A general k-cycle graph. Each cycle may be of any length.

In [5], lower bounds on the broadcast time were found both when the originator vertex is the central vertex as well as when it is some arbitrary vertex on any cycle. In particular, the lower bounds for the central vertex are:

Lemma 6: *Let G_k be a flower graph where the originator u is the central vertex v_c. Let $l_1 \geq l_2 \ldots \geq l_k$ be the number of vertices in the cycles $C_1, C_2, \ldots C_k$, all excluding v_c. Then,*

i. $b(u) \geq k+1$

ii. $b(u) \geq \lceil \frac{l_j+2j-1}{2} \rceil$ *for any* $j, 1 \leq j \leq k$.

iii. $b(u) \geq \lceil \frac{2k+l_j+2j+1}{4} \rceil$ *for any* $j, 1 \leq j \leq k$.

Additionally, let n be the total number of vertices in G_k*. Then,*

iv. if $b(u) \geq 2k$*, then* $b(u) \geq \lceil \frac{n-1}{2k} + k - \frac{1}{2} \rceil$

v. if $k+1 \leq b(u) < 2k$*, then* $b(u) \geq \lceil \sqrt{(2n - \frac{7}{4})} - \frac{1}{2} \rceil$

Applying their proofs for Lemma 6 in a very similar logic for the case when the originator is not the central vertex, Bhabak et al. additionally gave these bounds:

Lemma 7: *Let* G_k *be a flower graph where the originator* $u \neq v_c$ *is on some cycle* C_w*. Let* $l_1 \geq l_2 \ldots \geq l_k$ *be the number of vertices in the cycles* $C_1, C_2, \ldots C_k$*, all excluding* v_c*, and let d be the length of the shortest path from u to* v_c*. Then,*

i. $b(u) \geq d + k$

ii. $b(u) \geq d + \lceil \frac{l_j+2j-2}{2} \rceil$ *for any* $j, 1 \leq j \leq k$.

iii. $b(u) \geq d + \lceil \frac{2k+l_j+2j-2}{4} \rceil$ *for any* $j, 1 \leq j \leq k$.

They additionally gave a $(2-\varepsilon)$ approximation algorithm for any vertex as originator with $O(|V| + k \log k)$ running time, named S_{cycle}. temp. They then showed that it found an optimal broadcast time for particular subfamilies of flower graphs, specifically, when:

- $l_j \geq l_{j+1} + 4$ for all $1 \leq j < k$
- $l_j = l_{j+1}$ for all $1 \leq j < k$

They additionally provide an erroneous proof for the subcase $l_j = l_{j+2}$, and we warn that in fact, S_{cycle} in practice does *not* find optimal broadcast schemes for this particular subfamily.

In [10], a $O(k \log k \log |V|)$ heuristic called *BroadcastGuess* was given for broadcasting with $u = v_c$ as originator. It tended to find either optimal or nearly-optimal broadcast times on randomly generated flower graphs, and in practice was shown to find a broadcast scheme that was as good as or better than the one found by S_{cycle}, but no theoretical results were found with respect to their heuristic's broadcast scheme.

When broadcasting with $u = v_c$, both algorithms from [5] and [10] make an initial decision of immediately using the first round to inform the largest cycle, a decision which is known to be suboptimal for particular flower graphs: that is, some cycle graphs have no optimal broadcast schemes that inform the largest cycle first. However, both algorithms use significantly different strategies:

S_{cycle} chooses which cycle to call depending on the number of uninformed vertices left in each cycle while prioritising cycles which have never been called before.

Instead, *BroadcastGuess* uses a subroutine, *BroadcastBucket*, that takes in a number of rounds t as additional parameter. *BroadcastGuess* uses binary search to find the smallest t value for which *BroadcastBucket* returns a valid broadcast scheme. *BroadcastBucket* loops through all cycles in order of descending size, and assigns them the earliest free round available. For each cycle, after assigning it one round, the subroutine verifies if the cycle needs an additional round to be completely informed. If it does, the algorithm assigns the latest free round it can, while still guaranteeing that the cycle is fully informed by the end of round t.

The rest of this thesis focuses on this graph family.

2.1.9 The k-Path graph

Before moving on to our contributions, there is one last family of graphs worth mentioning due to their clear connection to flower graphs: *k-path graphs* are graphs composed of k paths of arbitrary lengths connected by a vertex on both ends. Just like flower graphs, no polynomial-time algorithm has been found for determining the broadcast time of k-path graphs, and these graphs seem to exhibit the same difficult as in flower graphs, despite being a fairly simple topology.

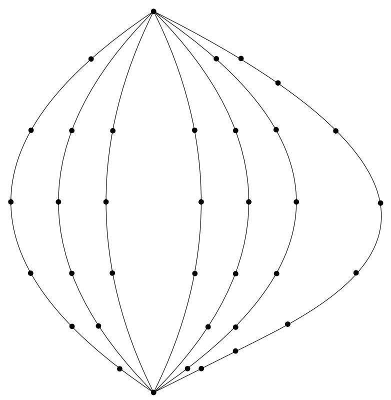


Figure 2.8: A k-path graph

In [5], Bhabak provides a $(4-\varepsilon)$-approximation algorithm for finding an optimal broadcast scheme on any originator in a k-path graph.

Chapter 3

Theoretical results

3.1 Approximation in the general case

In the following chapters, we use the following terminology: In a flower graph G of k cycles, we call l_i the *length* of cycle C_i, which we use to refer to the number of vertices in C_i excluding the central vertex v_c. Thus, supposing that our originator $u = v_c$, a cycle C_i has l_i vertices to be informed. We enumerate the cycles of G by descending order of their lengths: Hence, for the cycles $C_1, C_2, \dots C_k$, we have $l_1 \geq l_2 \geq \dots l_k$. When dealing with the case where $u = v_c$, we say that a cycle C is *called* or *informed* by the central vertex v_c when v_c broadcasts to one of the vertices of C. When every vertex in a cycle is informed, we say that this cycle is *fully informed*. At the beginning of the construction of some broadcast scheme on some t rounds $R_1, R_2, \dots R_t$, we initially consider our rounds to be *unassigned*. As we determine which cycle C_i should be called by v_c during some particular round R_j, we say that R_j is *assigned* to C_i. Likewise, if C_i has no rounds assigned to it, we say that C_i is *unassigned*. Observe also that when v_c calls C_i at some round R_j, that call results in having at most $t - j + 1$ vertices in C_i informed by the time R_t finishes (excluding v_c from our count, as it is the originator). Obviously, when v_c is the originator we can call each cycle twice, so if we call some C_i during some rounds R_p and R_q, we can inform at most a total of $2t - p - q + 2$ vertices on C_i (again excluding v_c).

We also note the following: First, $l_i \geq 2$ for all C_i: $l_i = 1$ would imply a single cycle of two vertices (including the originator) so a graph with multi-edges. This contradicts the graph model we are using, so such a cycle cannot exist.

We also hereby assume that we are broadcasting in non-trivial flower graphs: that is, we have more vertices than merely the originator. Note that a simple algorithm is given in [9] for recognising flower graphs, if needed.

Since we assume that we are working with non-trivial flower graphs, then for any flower graph G and any originator $u \in V(G)$, $b(u,G) \geq 2$: $b(u,G) = 1$ would imply a single cycle C_1 with $l_1 = 1$.

Using the same observation, we can also obtain an upper bound [4] for k: For any graph G with v_c as originator, v_c must call each cycle at least once: $1 \leq k \leq b(v_c,G)$. However, suppose we have $k = b(v_c,G)$ cycles. Consider the cycle called at the very last round of an optimal broadcast scheme: v_c cannot have called it in another earlier round, otherwise it would be left with a cycle it never called. But if it this cycle is only called by v_c the last round yet is successfully fully informed, then it must have at most 2 vertices including v_c: thus, it must be using multi-edges. Hence, we have $1 \leq k \leq b(v_c,G) - 1$.

3.1.1 Approximation when t is a parameter

In this section we describe Algorithm 1, a simple algorithm for broadcasting in a flower graph. It takes as input a flower graph G, an originator u, and some broadcast time t. It either returns a broadcast scheme which successfully informs the entire graph, or it returns $False$, indicating that it failed to generate a valid broadcast scheme. We will first show that for any G and originator $u \in V(G)$, Algorithm 1 can always generate a broadcast scheme using at most $\lfloor 1.5b(u,G) \rfloor$ rounds when given the correct[1] t value as input. We will then wrap Algorithm 1 with a binary search on t to identify the smallest t value for which this algorithm succeeds, thereby creating a (1.5)-approximation algorithm for finding the broadcast time of a flower graph.

Algorithm 1 separates the broadcasting problem on flower graphs in two cases: the first is when the originator $u = v_c$, the central vertex. In this case, it will attempt to generate a broadcast scheme using $\lfloor 1.5t \rfloor$ rounds. The second is when the originator is some vertex $u \neq v_c$, a vertex that is on some cycle C_w. In this case, it first determines which rounds are needed to inform C_w (and whether v_c will need to broadcast to C_w or not). Then, it will attempt to generate a broadcast scheme using v_c as originator, for broadcasting in the subgraph $G' = (V(G) - V(C_w), E(G) - E(C_w))$ (doing this while taking into account any already assigned round that is used by v_c for C_w, if such a round was used). Assuming that the distance from u to v_c is d, the broadcast scheme for G' (if successfully generated) will thus have to use $(d + \lfloor 1.5t \rfloor)$ rounds. The combination of these two schemes (the first to inform C_w, the second to inform G' from v_c) can be used to inform G completely, if the algorithm was successful in generating both broadcast schemes. If successful, the number of rounds used in total is dependent on the length of C_w relative to the number of rounds needed for G'. If unsuccessful, $False$ is returned.

[1] see Theorem 1

Algorithm 1 Approx$\lfloor 1.5 \rfloor$
Input: a flower graph G with central vertex v_c and k cycles sorted from largest to smallest, an originator u, and some value t
Output: A broadcast scheme on $\lfloor 1.5t \rfloor$ rounds or more, or the boolean value *False*

```
1: if $u \neq v_c$ then
2:     let $d = distance(u, v_c)$
3:     $u$ broadcasts along the shorter path to $v_c$ at $R_1$, and then broadcast along the longer path at $R_2$
4:     if $l_w > 2d + \lfloor 1.5t \rfloor - 1$ then          ▷ $C_w$ needs to be called by $v_c$ from the other side
5:         if $l_w \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$ then
6:             assign round $R_{t+d}$ to $C_w$
7:         else
8:             assign round $R_{d+1}$ to $C_w$
9:             Set $d = d + 1$
10: else let $d = 0$
11: for every cycle $C_i$ (if the originator is $u \neq v_c$, every cycle $C_i$ except $C_w$) do
12:     if $R_{d+i}$ is unassigned, assign $R_{d+i}$ to $C_i$
13:     else return False
14:     if $l_i > \lfloor 1.5t \rfloor - i + 1$ then          ▷ $C_i$ needs a second call from $v_c$
15:         if $l_i \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor - 2i + 2$ then          ▷ this $t$ value is too small, exit
16:             if $R_{t+d+i}$ is unassigned, assign $R_{t+d+i}$ to $C_i$
17:             else return False
18:         else return False
```

To prove that Algorithm 1 can generate valid broadcast schemes using at most $\lfloor 1.5b(u,G) \rfloor$ rounds, we will need the following two lemmas:

Lemma 8: *Given a flower graph G, some originator $u \neq v_c$ on some cycle C_w, let $G' = (V(G) - V(C_w), E(G) - E(C_w))$ be the subgraph of G induced by removing C_w. Suppose $d = distance(u, v_c)$. Then, $b(u,G) \geq d + b(v_c, G')$*

Proof. By construction of G', to inform G', u must inform v_c first and must then use a broadcast scheme that informs G' with v_c as originator. Since $d = distance(u, v_c)$, then $b(u,G) \geq d + b(v_c, G')$. □

Lemma 9: *Let G be a flower graph of k cycles. Let $t_{opt} = b(v_c, G)$, and let $l_1, l_2 \dots l_k \geq 2$ be the lengths (excluding the central vertex) of each cycle $C_1, C_2, \dots C_k$ in G. Then, for $i = 1, 2, \dots k$, $l_i \leq 2t_{opt} - 2i + 1$*

Proof. We know by assumption that there exists a broadcast scheme with v_c as originator which can fully informs G using t_{opt} vertices. Consider any broadcast scheme for G on t_{opt} rounds.

Observe that when calling cycles of size l_i or larger, v_c may only use rounds in the set $\{R_1, \dots, R_{\min(t_{opt}, 2t_{opt} - l_i + 1)}\}$:

When $l_i \leq t_{opt} + 1$ then $t_{opt} \leq 2t_{opt} - l_i + 1$, so the above statement claims that v_c may "only" use rounds in the set $\{R_1, \dots, R_{t_{opt}}\}$, that is, all rounds. This case is thus irrelevant, since no restrictions are given by the above statement.

Instead, when $l_i > t_{opt} + 1$, consider attempting to use $R_{2t_{opt} - l_i + 2}$ to inform a cycle of size l_i or larger. This round must exist since $l_i > t_{opt} + 1$. If we use this round to inform C_i, we will have at least $l_i - (t_{opt} - (2t_{opt} - l_i + 2) + 1) = t_{opt} + 1$ vertices left to inform in this cycle. We thus cannot completely inform the cycle by round t_{opt} using any other round for this cycle. So, for any broadcast scheme on t_{opt} rounds, $R_{2t_{opt} - l_i + 2}$ is not used to inform any cycle of size l_i or larger. The same proof can be applied on any round after $R_{2t_{opt} - l_i + 2}$. Thus, when informing cycles of size l_i or larger, we may only use rounds in the set $\{R_1, \dots, \min(t_{opt}, 2t_{opt} - l_i + 1)\}$.

We can now use this to our advantage: Consider again the two cases when $l_i \leq t_{opt} + 1$ and when $l_i > t_{opt} + 1$: We will show that in both cases, if we assume by contradiction that $l_i > 2t_{opt} - 2i + 1$, then we cannot fully inform all cycles in the set $\{C_1, \dots, C_i\}$ by round t_{opt}.

Since $l_i > 2t_{opt} - 2i + 1$ and cycles are ordered by their descending lengths, then we can conclude that we have at least i cycles of length of at least $2t_{opt} - 2i + 2$. So, we have at least $i(2t_{opt} - 2i + 2)$ vertices to inform in the set $\{C_1, \dots, C_i\}$.

First, consider the case when $l_i \leq t_{opt} + 1$. In this case, we have no restrictions on the rounds that we may use to inform any cycle in the set $\{C_1, \dots, C_i\}$. Observe that by having v_c make calls on all rounds $\{R_1, \dots R_{t_{opt}}\}$, we may thus at most inform $t_{opt} + (t_{opt} - 1) + (t_{opt} - 2) \dots + 1$ vertices in G. This sum, the total number of vertices that may possibly be informed by our calls, is $\sum_{j=1}^{t_{opt}} j = \frac{{t_{opt}}^2 + t_{opt}}{2}$. Then, observe that since $t_{opt} + 1 \geq l_i > 2t_{opt} - 2i + 1$, then $i > \frac{t_{opt}}{2}$. Inserting $i > \frac{t_{opt}}{2}$ in $i(2t_{opt} - 2i + 2)$, we get that we must inform strictly more than $\frac{t_{opt}}{2}(2t_{opt} - 2(\frac{t_{opt}}{2}) + 2) = \frac{t_{opt}^2 + 2t_{opt}}{2}$ vertices. Thus, we simply cannot fully inform our cycles with the number of vertices that we would be informing with rounds R_1 to $R_{t_{opt}}$.

Consider now the other case, $l_i > t_{opt} + 1$. In this case, it means that we cannot inform any cycle in the

set $\{C_1,\ldots,C_i\}$ using a single round: all of these cycles must be called twice to be fully informed by $R_{t_{opt}}$. Additionally, by the argument above, any cycle in the set $\{C_1,\ldots,C_i\}$ must use rounds in the set $\{R_1,\ldots,R_{2t_{opt}-l_i+1}\}$. Thus, we can conclude that we have $i \leq \frac{2t_{opt}-l_i+1}{2}$, otherwise we do not have enough rounds to call each cycle in the set $\{C_1,\ldots,C_i\}$ twice, meaning there must be a cycle in this set that is not fully informed by $R_{t_{opt}}$.
But, by assumption $l_i \geq t_{opt}+2$, so $i \geq \frac{2t_{opt}-l_i+2}{2} > \frac{2t_{opt}-l_i+1}{2}$. Contradiction.
So, in all cases, there does not exist a broadcast scheme for G on t_{opt} rounds. Contradiction, since it was assumed that $b(v_c,G) = t_{opt}$.

□

With the above two lemmas, we can now make the following claims:

Theorem 1: Subroutine (1.5)**-approximation:** *Given a flower graph G, an originator u, and some value t:*

- *if $u = v_c$ and $t = b(v_c,G)$, Algorithm 1 generates a $\lfloor 1.5b(u,G)\rfloor$ broadcast scheme.*
- *if $u \neq v_c$ on some C_w, then let $G' = (V(G)-V(C_w),E(G)-E(C_w))$. If $t = b(v_c,G')$, Algorithm 1 generates a $\lfloor 1.5b(u,G)\rfloor$ broadcast scheme.*

Proof.

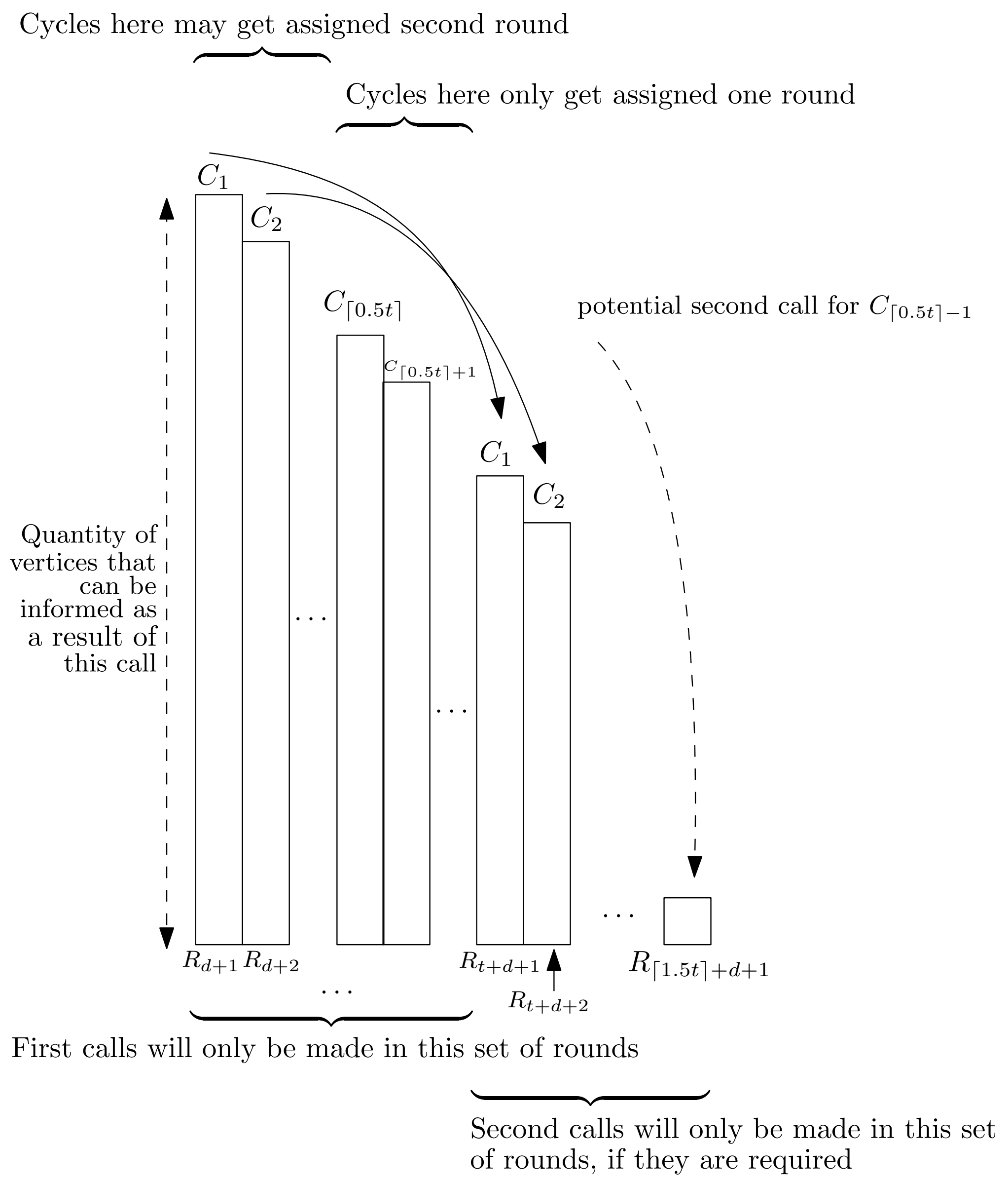
Case 1: *$u = v_c$ and $t \geq b(v_c,G)$, total quantity of rounds: $\lfloor 1.5t\rfloor$*
Note that even if in this case, $d = 0$ by line 10, we will still refer to d in our calculations as this case will be used by the case when $u \neq v_c$, where $d > 0$.
We will first describe the broadcast scheme generated by Algorithm 1, and then explain how it is a valid broadcast scheme that successfully informs G by round $R_{\lfloor 1.5t\rfloor}$.
Algorithm 1 loops on the set of cycles C_i from the largest cycle to the smallest one.
For every cycle C_i, it assigns them R_{d+i}. Thus, in total, it will assigns as first calls all rounds in the set $\{R_{d+1},\ldots R_{d+k}\}$, with $d+k \leq d+t-2$ as explained in Section 3.1.
For every cycle C_i, after having assigned it a first round, Algorithm 1 will decide if a second call from v_c will be granted. To do this, it verifies if $l_i > \lfloor 1.5t\rfloor - i + 1$ on line 14. When is this condition guaranteed to be false? In other words, for which cycles can we claim that Algorithm 1 will never grant them a second call?
Recall that by Lemma 9, $l_i \leq 2t - 2i + 1$.

$$2t - 2i + 1 \leq \lfloor 1.5t \rfloor - i + 1$$
$$\lceil 0.5t \rceil \leq i$$

Thus, Algorithm 1 will never grant a second call to any cycle C_i when $i \geq \lceil 0.5t \rceil$.

Consider instead the cycles C_i for $i < \lceil 0.5t \rceil$. For any such C_i, if $l_i \leq \lfloor 1.5t \rfloor - i + 1$, then Algorithm 1 merely moves on to the next cycle (or returns the entire broadcast scheme if that was the last cycle) after having assigned them R_{d+i}. Suppose instead that $l_i > \lfloor 1.5t \rfloor - i + 1$. In this case, Algorithm 1 verifies on line 15 whether or not the t value it was given is too small with respect to the size of the cycle. Observe that when $t \geq b(v_c, G)$ as assumed in this case, then $l_i \leq 2b(v_c, G) - 2i + 1 \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor - 2i + 2$ once again by Lemma 9. Thus when $t \geq b(v_c, G)$, line 15 will always resolve to $True$, and a second call on round R_{t+d+i} will be granted. Since in this case, $i < \lceil 0.5t \rceil$, all possible rounds assigned as second calls are rounds in the set $\{R_{t+d+1}, \ldots, R_{t+d+\lceil 0.5t \rceil - 1}\}$, with $t + d + \lceil 0.5t \rceil - 1 \leq \lfloor 1.5t \rfloor + d$.

Thus, if $t \geq b(v_c, G)$, Algorithm 1 returns a broadcast scheme that resembles the one in Figure 3.1.



Cycles here may get assigned second round
Cycles here only get assigned one round
C_1
C_2
$C_{\lceil 0.5t \rceil}$
$C_{\lceil 0.5t \rceil + 1}$
potential second call for $C_{\lceil 0.5t \rceil - 1}$
C_1
C_2
Quantity of vertices that can be informed as a result of this call
R_{d+1} R_{d+2}
R_{t+d+1}
R_{t+d+2}
$R_{\lceil 1.5t \rceil + d + 1}$
First calls will only be made in this set of rounds
Second calls will only be made in this set of rounds, if they are required

Figure 3.1: A broadcast scheme generated by Algorithm 1 with $u = v_c$ and $t = b(v_c, G)$.

Having explained the broadcast scheme generated, we will now demonstrate that it successfully fully informs G in at most $\lfloor 1.5t \rfloor + d$ rounds.

Assume thus that we are using $\lfloor 1.5t \rfloor + d$ rounds. As just shown, Algorithm 1 uses R_{d+i} to have v_c inform C_i. This call results in having at most $\lfloor 1.5t \rfloor + d - (d+i) + 1 = \lfloor 1.5t \rfloor - i + 1$ vertices informed in C_i (excluding v_c).

As shown above, if $i \geq \lceil 0.5t \rceil$, then $l_i \leq 2b(v_c, G) - 2i + 1 \leq \lfloor 1.5t \rfloor - i + 1$, so C_i is fully informed with this single call by round R_{d+i}.

If instead $l_i > \lfloor 1.5t \rfloor - i + 1$, then $i < \lceil 0.5t \rceil$, and Algorithm 1 additionally assign R_{t+d+i} to C_i.

This informs at most an additional $\lfloor 1.5t \rfloor + d - (t+d+i) + 1 = \lfloor 1.5t \rfloor - t - i + 1$ vertices, for a total of

$\lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor - 2i + 2 \geq 2t - 2i + 1$ vertices informed by both calls. By Lemma 9 and since we're using $t \geq b(v_c, G)$, Algorithm 1 always fully informs C_i using these two rounds, if needed.

Hence, in the case where when $u = v_c$ and $t \geq b(v_c)$, the scheme generated by Algorithm 1 fully informs all cycles by round $R_{\lceil 1.5t \rceil - 1 + d}$, and $\lceil 1.5t \rceil - 1 + d \leq \lfloor 1.5t \rfloor + d$. When $t = b(v_c, G)$ and $u = v_c$, this means that we inform all cycles by round $R_{\lfloor 1.5b(v_c,G) \rfloor + d}$ with $d = 0$, and thus achieve a (1.5)-approximation in this case. Note that because cycles are not guaranteed to need second calls, it is possible that v_c ends up not making a call on the last round (or some contiguous sequence of rounds that includes the last round). However, we still need to use $R_{\lfloor 1.5t \rfloor}$ rounds to guarantee that all cycles are fully informed by the time the broadcast scheme terminates.

Case 2: *$u \neq v_c$ and $t \geq b(v_c, G')$*

In the following proof, despite the fact that Algorithm 1 has the possibility of setting $d = d + 1$, we will always use $d = dist(u, v_c)$, and merely account for this additional 1 explicitly in our sums if we need it.

u first broadcasts along its shorter path towards v_c, and then to its longer path. Since $d = distance(u, v_c)$ initially, v_c will be informed at round R_d and start making calls at round R_{d+1}. Algorithm 1 needs to figure out whether or not v_c will broadcast to C_w, and when it does so.

After deciding if and when to broadcast to C_w, Algorithm 1 can fall back to Case 1, acting as if v_c is the originator in the subgraph G', provided all the rounds needed by Case 1 are unassigned.

According to Algorithm 1, we have three possibilities for calling C_w from v_c:

- $l_w < 2d + \lfloor 1.5t \rfloor - 1$: v_c does not broadcast to C_w. Thus, all rounds $\{R_{d+1}, R_{d+\lfloor 1.5t \rfloor}\}$ remain unassigned and v_c can start to broadcast to G' at round R_{d+1}, and finish informing all vertices in $V(G')$ at $R_{d+\lfloor 1.5t \rfloor}$, as per Case 1.

- $2d + \lfloor 1.5t \rfloor - 1 \leq l_w \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$: v_c broadcasts to C_w at round R_{d+t}, which is left unassigned by Case 1, and like the above case, finishes informing all vertices in $V(G')$ at $R_{d+\lfloor 1.5t \rfloor}$.

- $\lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d < l_w$: assuming $d = dist(u, v_c)$, v_c broadcasts to C_w at round $R_{d'} = R_{d+1}$ and thus broadcasts to G' starting at R_{d+2} and finishes informing all vertices in $V(G')$ at $R_{d+\lfloor 1.5t \rfloor + 1}$.

We need to show that in all cases, we remain below or equal to $\lfloor 1.5b(u, G) \rfloor$ rounds by the time we fully inform C_w and G'.

For now consider only the vertices in C_w that are informed thanks to the calls that u makes, regardless of the call that v_c may or may not make on C_w. By round $R_{d+\lfloor 1.5t \rfloor}$, we have a total of $2d + \lfloor 1.5t \rfloor - 1$ vertices informed in C_w (excluding v_c), from the following sum:

- d vertices, consisting of u and the $d-1$ vertices on u's shortest path to v_c, in addition to:
- $d+\lfloor 1.5t \rfloor - 1$ vertices, consisting of vertices on the longer path from u to v_c, which begins to be informed at round R_2 by u's second call and can continue to be informed until the very last round, which is at least $R_{d+\lfloor 1.5t \rfloor}$.

Consider now the following subcases:

Subcase 2.1: *$l_w \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$, total quantity of rounds: $\lfloor 1.5t \rfloor + d$*

If $l_w \leq 2d + \lfloor 1.5t \rfloor - 1$, Algorithm 1 just has v_c ignore C_w as it will be fully informed by $R_{\lfloor 1.5t \rfloor + d}$, as per the sum above.

If $l_w > 2d + \lfloor 1.5t \rfloor - 1$, we can have v_c broadcast to C_w using R_{d+t}, since, as seen in the proof of Case 1, R_{d+t} will never used by v_c to inform any cycle in G' if $t \geq b(v_c, G')$. If we use it to have v_c inform C_w, that brings our total of vertices informed in C_w to $2d + \lfloor 1.5t \rfloor - 1 + \Big(\lfloor 1.5t \rfloor + d - (t+d) + 1\Big) = 2d + \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor$. Since G' is also completely informed by $R_{\lfloor 1.5t \rfloor + d}$ and $\lfloor 1.5t \rfloor + d < \lfloor 1.5b(u,G) \rfloor$, then G is fully informed before $\lfloor 1.5b(u,G) \rfloor$.

Subcase 2.2: *$l_w > \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$, total quantity of rounds:* $\max(d + \lfloor 1.5t \rfloor + 1, \lfloor \frac{l_w}{2} \rfloor + 1)$

First observe that in this case, C_w must be the largest cycle in G: by Lemma 9, the largest cycle in G' has its length $\leq 2t - 1 < \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$.

Then, since $l_w = \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d + x$ for some $x \geq 1$, then by Lemma 6(ii.), $b(u,G) \geq \lceil \frac{\lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d + x + 1}{2} \rceil \geq t + d + \lceil \frac{x}{2} \rceil$.

Here, we merely have v_c's first call be to C_w's other vertex. Then, v_c can start informing G' at R_{d+2}, and thus G' is completely informed by $R_{d+\lfloor 1.5t \rfloor + 1}$, and C_w is completely informed by $R_{\lfloor \frac{l_w}{2} \rfloor + 1}$.

If $d + \lfloor 1.5t \rfloor + 1 \geq \lfloor \frac{l_w}{2} \rfloor + 1$, this means that our broadcast scheme takes $d + \lfloor 1.5t \rfloor + 1$ rounds total to complete.

Then, observe:
$$\begin{aligned} d + \lfloor 1.5t \rfloor + 1 &< 2d + \lfloor 1.5t \rfloor + \lceil \tfrac{x}{2} \rceil \text{ since } d, x \geq 1 \\ &< \lfloor 1.5(t + 2d + \lceil \tfrac{x}{2} \rceil) \rfloor \\ &\leq \lfloor 1.5(d + b(u,G)) \rfloor \text{ by Lemma 6(ii.)} \\ &\leq \lfloor 1.5(b(u,G)) \rfloor \text{ by Lemma 8} \end{aligned}$$

If instead $d + \lfloor 1.5t \rfloor + 1 < \lfloor \frac{l_w}{2} \rfloor + 1$, then our broadcast scheme instead takes $\lfloor \frac{l_w}{2} \rfloor + 1$ rounds to complete. Since an optimal broadcast scheme for the entire G with v_c as originator would need to inform the entirety of C_w, then $b(u,G) \geq \lfloor \frac{l_w}{2} \rfloor + 1$:

$$\begin{aligned}
\lfloor \tfrac{l_w}{2} \rfloor + 1 &\leq b(u,G) \\
&< \lfloor 1.5b(u,G) \rfloor \\
&< \lfloor 1.5(d + b(u,G)) \rfloor \\
&\leq \lfloor 1.5(b(u,G)) \rfloor \text{ by Lemma 8}
\end{aligned}$$

So, in all cases, when $u \neq v_c$ and $t = b(v_c, G')$, we obtain a broadcast scheme with $1.5(b(u,G))$ rounds or fewer.

□

Algorithm 1 has an $O(k)$ running time, since it performs constant-time steps for every cycle, of which we have k.

We can now wrap Algorithm 1 in a binary search that looks for the smallest t value for which Algorithm 1 sucessfully returns a broadcast scheme.

3.1.2 Subroutine for tightening the range of t values

Before giving the pseudocode for the binary search wrapper, we take the opportunity to comment on the range of t values we will need to search. Instead of searching for t on a naively wide initial range of values, we can use the lower bounds provided in [5] which are given in the previous section (see Lemma 6), as well as the following simple upper bounds to tighten the range. We do note that this still however results in an $O(|V|)$ range of values for t.

Lemma 10: *Let G be a flower graph with k cycles. Let $l_1, l_2 \dots l_k \geq 2$ be the lengths (excluding the originator) of each cycle $C_1, C_2, \dots C_k$ in G.*

Then, if its central vertex v_c is originator:

i. $b(v_c) \leq \max\limits_{1 \leq i \leq k} (l_i + i - 1)$

ii. $b(v_c) \leq \max\limits_{1 \leq i \leq k} \left(2i + (\lfloor \frac{l_i}{2} \rfloor + 1)\right)$

Else some other vertex u on a cycle C_w is originator:

iii. $b(u) \leq d + \max\limits_{1 \leq i \leq k} (l_i + i - 1)$

iv. $b(u) \leq d + \max_{1 \leq i \leq k} \left(2i + (\lfloor \frac{l_i}{2} \rfloor + 1)\right)$

Proof. First consider when v_c is the originator: Then, for case i., we can imagine a very simple broadcast scheme which simply calls each cycle once by descending order of their lengths. Thus, C_1 will be fully informed at round l_1, C_2 will be fully informed by round $l_2 + 1$, and in general C_i will be fully informed by round $l_i + i - 1$. This achieves a broadcast scheme that uses at most $\max(l_i + i - 1)$ rounds. For case ii., we instead use a different very simple broadcast scheme, which uses two rounds for each cycle: rounds R_1 and R_2 to inform C_1, rounds R_3 and R_4 to inform C_2, and in general rounds R_{2i-1} and R_{2i} to inform round C_i. Thus, C_1 will be fully informed at round $\lfloor \frac{l_2}{2} \rfloor + 1$, C_2 will be fully informed by round $\lfloor \frac{l_2}{2} \rfloor + 5$, and C_i will be in general fully informed by round $2i + (\lfloor \frac{l_i}{2} \rfloor + 1)$. This achieves a broadcast scheme that uses at most $\max(2i + (\lfloor \frac{l_i}{2} \rfloor + 1))$ rounds.

Then, consider when $u \neq v_c$ is the originator. Then, by Lemma 8, $b(u) \geq d(u, v_c) + b(v_c)$. Suppose u makes a call on its shorter path to v_c, and thus informs v_c at round d. For case iii., for any cycle C_j, v_c can begin making calls on the remaining $k-1$ cycles starting at round $d+1$, thus we simply have v_c make a call to C_j on round $l_j + d$. As we make no prescription in our algorithm for a particular behaviour of v_c when it reaches the cycle C_w, we can make a loose upper bound of $\max(d + l_i + i - 1)$.

In a very similar manner, case iv. can reach an upper bound of $\max(d + 2i + (\lfloor \frac{l_i}{2} \rfloor + 1))$ rounds. □

Hence we can make the following subroutine for the range of our t values:

Algorithm 2 tRange
Input: a flower graph G with k cycles sorted by descending sizes and an originator u
Output: A lower and an upper bound on $b(u)$

```
1: if $u$ is the central vertex $v_c$ then
2:     $left \leftarrow \max\left(k+1, \max_{1 \leq j \leq k}(\lceil \frac{l_j+2j-1}{2} \rceil), \max_{1 \leq j \leq k}(\lceil \frac{2k+l_j+2j+1}{4} \rceil)\right)$
3:     $right \leftarrow \min\left(\max_{1 \leq i \leq k}(l_i + i - 1), \max_{1 \leq i \leq k}\left(2i + (\lfloor \frac{l_i}{2} \rfloor + 1)\right)\right)$
4: else $u$ is some other vertex $u$ on some cycle $C_w$
5:     $left \leftarrow \max\left(k+d, \max_{1 \leq j \leq k}(\lceil \frac{l_j+2j-2}{2} \rceil), \max_{1 \leq j \leq k}(\lceil \frac{2k+l_j+2j-2}{4} \rceil)\right)$
6:     $right \leftarrow \min\left(\max_{1 \leq i \leq k}(d + l_i + i - 1), \max_{1 \leq i \leq k}\left(d + 2i + (\lfloor \frac{l_i}{2} \rfloor + 1)\right)\right)$
7: return $left, right$
```

We can now combine the lower and upper bounds into a range within which the optimal t value is guaranteed to be, a small result we need for an upcoming theorem.

Corollary 1: *For any flower graph G with k cycles and an originator u, consider the values **left**, **right** returned by Algorithm 2. Then,* $\textbf{left} \leq b(u,G) \leq \textbf{right}$.

Proof. Follows from Lemma 6 and Lemma 10. □

We can use the above subroutine in our binary search wrapper:

Algorithm 3 BroadcastWrapper
Input: a flower graph G with k cycles, an originator x, and an algorithm that returns a valid broadcast scheme or the boolean value **False**
Output: the smallest t value for which $A(G,x,t)$ found a broadcast scheme, or the upper bound on $b(x,G)$ from $tRange(G,x)$ if $A(G,x,t)$ failed to find any broadcast schemes.

```
1:  sort G's cycles from largest to smallest lengths.
2:  $left, right \leftarrow tRange(G,x)$
3:  $BroadTime \leftarrow right$
4:  while $left \leq right$ do
5:      $middle = \lceil \frac{left+right}{2} \rceil$
6:      $PossibleBroadcastScheme \leftarrow A(G,x,middle)$
7:      if $PossibleBroadcastScheme$ is not False then
8:          $BroadTime \leftarrow middle$
9:          $right \leftarrow middle - 1$
10:     else
11:         $left \leftarrow middle + 1$
12: return $BroadTime$
```

Theorem 2: (1.5)**-approximation:** *Algorithm 3 with* $A = Algorithm\ 1$ *is a* (1.5)*-approximation algorithm for any originator in the flower graph G.*

Proof. Follows from Theorem 1 and Corollary1. □

When using Algorithm 1, Algorithm 3 runs in $O(k\log(k)) + O(k) + O(k\log(|V|)) = O(k\log(|V|))$: it first sorts the cycles from largest to smallest at line 1, then fetches the upper and lower bound by performing constant-time operations on each cycle length at line 2, and finally performs a binary search on $\log(|V|)$ values starting at line 4 and at each step, executes our $O(k)$ Algorithm 1 at line 6.

3.2 Improvement on Algorithm 1

One can naturally see that a few minor modifications to Algorithm 1 can result in an improvement:

Recall that for the first $\lceil 0.5t \rceil - 1$ cycles, Algorithm 1 may assign a second round to these cycles if they are not fully informed using only R_{d+i}. If it does, it always assigns R_{t+d+i} to them, permitting us to make this assignment with a constant-time operation and maintain our running time to $O(k)$ for the entire algorithm. Instead, we can choose to search (naively, in $O(t)$ time) for the *latest unassigned round* that can be used to still completely inform C_i by round t. This minor change often results in successfully finding a broadcast scheme using fewer rounds than Algorithm 1 needs to find a valid broadcast scheme. However, since we use fewer rounds, it may be that the second round we assign to some C_i ends up being a round that is intended to be used as a first round or second round by a cycle that appears after C_i in the order of cycles. Thus, we need to additionally keep a pointer to the earliest unassigned round to ensure no accidental 'double booking' of the rounds is made.

We note that the *BroadcastBucket* heuristic given in [10] operates in nearly exactly the manner just described above, with some caveats: It uses buckets to represent the rounds that v_c will call vertices on, allowing for multiple calls to be assigned to the same bucket. When assigning the second call to some cycle during some particular round, the heuristic makes no attempt to strictly reserve that round to the cycle that claims it, and instead stores the call in the bucket used to represent that round. If another other cycle later claims that same round, it will insert the second call in that same bucket (thus now containing two calls). After having assigned call(s) to buckets for every cycle, it merely counts whether it has enough unassigned rounds to satisfy each cycle: A call in an already-occupied bucket can be moved to an earlier, but unassigned, bucket. If enough unassigned buckets (that appear early enough) exist, then a valid broadcast scheme exists. This results in nearly identical, but occasionally slightly different performances between the two algorithms (see Chapter 5).

Despite very similar behaviour to *BroadcastBucket* and thus very similar performances in practice, we still judge it pertinent to describe Algorithm 4, as it can be easily shown to also achieve (1.5)-approximation ratio.

Algorithm 4 ImprovedApprox$\lfloor 1.5 \rfloor$
Input: a flower graph G with k cycles sorted from largest to smallest, an originator u, and some value t
Output: A broadcast scheme on $\lfloor 1.5t \rfloor$ rounds or more, or the boolean value $False$

```
1: if $u \neq v_c$ then
2:     let $d = distance(u, v_c)$ and $t' = \lfloor 1.5t \rfloor + d$
3:     $u$ broadcasts along the shorter path to $v_c$ at $R_1$, and then broadcast along the longer path at $R_2$
4:     if $l_w > 2d + \lfloor 1.5t \rfloor - 1$ then          ▷ $C_w$ needs to be called by $v_c$ from the other side
5:         if $l_w \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$ then
6:             assign round $R_{t+d}$ to $C_w$
7:         else
8:             assign round $R_{d+1}$ to $C_w$
9:             Set $d = d + 1$ and $t' = \max(\lfloor 1.5t \rfloor + d + 1, \lfloor \frac{l_w}{2} \rfloor + 1)$
10: else let $d = 0$ and $t' = \lfloor 1.5t \rfloor$
11: let $eur = d + 1$          ▷ $eur$ is used to indicate the index of the earliest unassigned round
12: for every cycle $C_i$ (if $u$ is the originator, except $C_w$) do
13:     assign $R_{eur}$ to $C_i$
14:     if $l_i > t' - eur + 1$ then          ▷ $C_i$ needs to be assigned second round
15:         If there exists at least one unassigned round $R_j$ where $1 \leq j \leq 2t' - eur - l_i + 2$, assign the latest
        such round to $C_i$
16:         else return False
17:     if such a round exists, set $eur$ to the index of the earliest unassigned round, else return False
```

Theorem 3: $1.5 - \varepsilon$**-approximation:** *Algorithm 3 with* $A = Algorithm$ 4 *is a* $(1.5 - \varepsilon)$*-approximation algorithm for any originator in the flower graph G.*

Proof. Consider first how Algorithm 1 and Algorithm 4 differ: Algorithm 4 uses t' to represent what we will prove is an upper bound on the number of rounds used by the broadcast scheme. This value is obtained in line 2,line 9, or line 10. Algorithm 4 also uses *eur* to represent the index of the earliest round that v_c can use to inform any vertex, but that as of yet still stands unassigned. It is first set in line 13, and then updated for every cycle in line 17.

Algorithm uses these two values in line 14 to determine the number of rounds informed by the first call on C_i and thus decide if a second call to C_i is required or not. If so, it again uses these values in line 15 to find the latest round at which C_i can be called to be still completely informed by t'.

We will first prove the claims we just made on the purposes of t' and eur to demonstrate that when Algorithm 4 returns a broadcast scheme, this broadcast scheme is valid and fully informs G by round t'. We will then argue that if Algorithm 1 manages to create a broadcast scheme for some particular t value, so will Algorithm 4, thereby reaching a (1.5)-approximation.

First, we show that t' accurately represents the number of rounds used by the broadcast scheme. We ignore instances when Algorithm 4 returns False since no broadcast scheme is returned. Suppose thus that this is an instance where Algorithm 4 successfully returns a broadcast scheme. Recall the cases detailed in the proof for Theorem 2, as they come up again here:

Case 3: $u = v_c$ *and* $t = b(v_c, G)$

There, $d = 0$ and $t' = \lfloor 1.5t \rfloor$, as per line 10. Assume that the number of rounds used by our broadcast scheme is at least $t' = \lfloor 1.5t \rfloor$.

For every cycle C_i, consider the sums of vertices accounted for by the round(s) assigned to C_i:

By the first call using R_{eur}, at least $\lfloor 1.5t \rfloor - eur + 1$ vertices are informed. If $l_i \leq \lfloor 1.5t \rfloor - eur + 1$, no second round was assigned to C_i. This is fine, as we inform $\lfloor 1.5t \rfloor - eur + 1$ vertices by round $R_{\lfloor 1.5t \rfloor}$. If $l_i > \lfloor 1.5t \rfloor - eur + 1$, then a second round was assigned to C_i: R_j, with $1 \leq j \leq 2t' - eur - l_i + 2$. This second call informs $t' - (2t' - eur - l_i + 2) + 1 = -t' + eur + l_i - 1$ vertices by round t'. Thus we have at least a total of $\big(t' - eur + 1\big) + \big(-t' + eur + l_i - 1\big) = l_i$ vertices informed by round t'. Since we are working with an instance that successfully returned a broadcast scheme, than such a j must have existed: thus, we always fully inform C_i. In the case where l_i is too large to be fully informed by $\lfloor 1.5t \rfloor$, no such R_j would exist that would satisfy the condition in line 15.

Case 4: $u \neq v_c$ *and* $t = b(v_c, G')$

There, as in the proof for Theorem 2, we have two cases:

Subcase 4.1: $l_w \leq \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$

Then, $t' = \lfloor 1.5t \rfloor + d$ as per line 2.

By the same logic as in the proof for Theorem 2, C_w is fully informed by round $d + \lfloor 1.5t \rfloor$, possibly while using R_{d+t} if $l_w > 2d + \lfloor 1.5t \rfloor - 1$. G' must also be fully informed by round $R_{d+\lfloor 1.5t \rfloor}$ even in the case where C_w is granted a second call, as we assumed that Algorithm 4 returned a broadcast scheme, and as the lines 13 to 15 take into account the unavailability of rounds when selecting rounds to inform each C_i. Additionally, as shown for Case 3, these rounds are selected such that every C_i considered (that is, G') is fully informed by round t'.

Subcase 4.2: $l_w > \lfloor 1.5t \rfloor + \lfloor 0.5t \rfloor + 2d$

By the same logic as in the proof for Theorem 2 and the proof of the subcase above, C_w is again informed by round $R_{\max(d+\lfloor 1.5t \rfloor+1, \lfloor \frac{l_w}{2} \rfloor+1)}$ and G' by $R_{d+\lfloor 1.5t \rfloor+1}$.
Thus, t' accurately represents an upper bound on the number of rounds used by the broadcast scheme.
Suppose now that for some t value, Algorithm 1 returns a valid broadcast scheme. Observe by proof of Theorem 2 that the quantity of rounds that is used by the broadcast scheme of Algorithm 1 is equal to the value that t' is set to in Algorithm 4. This would thus mean that if Algorithm 4 returns a broadcast scheme, then that broadcast scheme uses as many rounds as Algorithm 1. But we still need to prove that Algorithm 4 successfully does return such a broadcast scheme.
We demonstrate this by showing Algorithm 4 always has rounds to assign to its cycles in G.
In the case where $u \neq v_c$, the rounds assigned to C_w are the same in both algorithms, so we can safely assume C_w will be fully informed by t' in Algorithm 4: the proof of Theorem 2 demonstrates this.
Then, for both cases $u \neq v_c$ and $u = v_c$, consider by strong induction on i the rounds chosen for $C_i \neq C_w$ in both algorithms: For our base case, both algorithms use as their first round for C_1, $R_{eur} = R_{d+1}$. Then, if C_1 needs a second call, then Algorithm 1 uses R_{d+t+1}, and Algorithm 4 uses R_{d+t+1} or a later round.
For our induction step, we assume that after having made the round assignments for C_i and all previous cycles, that the broadcast scheme slowly constructed by Algorihm 4 meets the following requirements:

- All rounds from $\{R_{d+1}, \dots, R_{d+i}\}$ are reserved, and all rounds $\{R_{d+i+1}, \dots, R_{d+t-1}\}$ are unassigned.
- There exists at least one unassigned round in the set $\{R_{d+t+1}, \dots, R_{d+t+i+1}\}$

Observe that after our base case, the above is true: First, R_{d+1} was used as the first round for C_1, so $\{R_{d+2}, \dots, R_{d+t-1}\}$ remain unassigned, making the first condition true. Secondly, if Algorithm 4 used R_{d+t+1} as a second round for C_1, then R_{d+t+2} is unassigned, making the second condition true. If Algorithm 4 instead used a later round as second round for C_1, then R_{d+t+1} remains unassigned.
Consider now the assignment for C_{i+1}: The earliest unassigned round useable by v_c must be R_{d+i+1}, since all rounds $\{R_{d+1}, \dots, R_{d+i}\}$ are reserved. So we grant R_{d+i+1} to C_{i+1}. This makes $\{R_{d+1}, \dots, R_{d+i+1}\}$ all reserved. It also maintains $\{R_{d+i+2}, \dots, R_{d+t-1}\}$ all unassigned.
Observe that Algorithm 1 also uses R_{d+i+1} for C_{i+1}. If C_{i+1} needs a second round, it assigns $R_{d+t+i+1}$ to it. Since at least one round in the set $\{R_{d+t+1}, \dots, R_{d+t+i+1}\}$ is unassigned for Algorithm 4, then Algorithm 4 can

also make a valid second round assignment. If it assignes a round in the set $\{R_{d+t+1},\dots,R_{d+t+i+1}\}$, then this means that there exists one unassigned round in the set $\{R_{d+t+1},\dots,R_{d+t+i+2}\}$: $R_{d+t+i+2}$ isn't used. If instead is uses a round that appears strictly after $R_{d+t+i+1}$, then that unassigned round in $\{R_{d+t+1},\dots,R_{d+t+i+1}\}$ still exists, so there exists a unassigned round in the set $\{R_{d+t+1},\dots,R_{d+t+i+2}\}$. We can continue this until all cycles are assigned rounds.

Thus, whenever Algorithm 1 finds a valid broadcast scheme, so does Algorithm 4, and these schemes use the same quantity of rounds, that is, t'.

□

3.3 Discussion on Algorithm 1, Algorithm 4, and *BroadcastBucket*

Before considering the running time of Algorithm 4, it is useful to observe the differences between *BroadcastBucket* and Algorithm 4:

First, remark that *BroadcastBucket* fails to get within (1.5)-approximation: *BroadcastBucket* requires two calls to be made to every cycle, regardless of whether a cycle was fully informed by its first call. Therefore, at best *BroadcastBucket* can only achieve a (2)-approximation ratio.

This can be very easily fixed by merely not giving a second round to cycles that don't need it: if the first call can fully inform a cycle by the given t value, then *BroadcastBucket* should immediately move on to the next cycle. Since this modification is very simple, we will now assume *BroadcastBucket* to have this modification applied, and we will identify this modified algorithm as *BroadcastBucket**.

When looking at the proof of Theorem 3, one can infer that a very similar proof may be applicable to *BroadcastBucket** and thus that it may achieve a (1.5)-approximation ratio, although we do not attempt this as Algorithms 1 and 4 already achieve this approximation ratio, and Algorithm 4 obtains nearly exactly the same broadcast times as *BroadcastBucket** in practice (see Chapter 5)

The second fundamental difference appears when having to look for a second round if a cycle requires it: For every C_i that needs it, Algorithm 4 will search for an unassigned round to use for second call (an $O(t)$ operation) immediately after assigning the first call to R_{eur}. As explained before, *BroadcastBucket** instead merely places the call temporarily in the latest bucket it can afford to, even if that bucket is already occupied. After completing this loop on every cycle, *BroadcastBucket** verifies it has enough unassigned buckets such that it can rearrange the calls, and thus evading this t factor. This crucial difference may mean that *BroadcastBucket** in fact fails to get a (1.5)-approximation ratio due to some particular nemesis instances that are particularly sensitive to this difference, but we have not verified this.

This difference comes into play when comparing their running times as well. First, consider the running time of Algorithm 4:

For every cycle, Algorithm 4 may need to search for R_j, the the latest round it can use while still being completely informed by t'. It also needs to update the value *eur*. These two actions naively take, in the worst case, $O(t) = O(|V|)$ time for each cycle. We can easily reduce the cost of each update operation on line 17 to a constant factor with amortized analysis (and thus have it be an $O(t)$ operation over the entire algorithm): Let our operation on line 17 be as follows: Algorithm 4 performs a single constant time *check* operation to verify if R_{eur} is assigned. If this round is unassigned, *eur* does not change, and we can continue the execution of the algorithm as usual. If this round is reverved, Algorithm 4 will make a series of *move-check* operations, incrementing *eur* by one and verifying if R_{eur} is assigned or not, repeating this incrementing and checking until it finds a unassigned round. Once it does, Algorithm 4 can continue the execution of the algorithm as usual. Observe that over the course of the entire algorithm, we make $O(k)$ *check* operations and at most only $O(t)$ *move-check* operations: We only run a *move-check* operation if the previous *check* or *move-check* operation found an assigned round, and we necessarily increment *eur* in this case. If $eur > t$, we return *False*. Thus, line 17 makes at most $O(k) + O(t) = O(t)$ constant-time operations.

Unfortunately, searching for R_j still remains an $O(t)$ operation (thus costing $O(tk)$ over the entire algorithm, due to the $O(k)$ for-loop) since we reserve rounds without much restrictions: mostly any round before or at R_j could be the latest unassigned round available at some time.

Therefore, Algorithm 4 has an $O(tk) = O(|V|k)$ running time. From this, Algorithm 3 has an $O(|V|k\log|V|)$ running time when using Algorithm 4.

This is significantly worse than *BroadcastBucket**'s running time: $O(k\log k)$, and thus $O(k\log k\log|V|)$ with the binary search wrapper. We can however get rid of the $\log k$ factor that comes with searching for the earliest unassigned round in *BroadcastBucket** with amortized analysis in a similar manner to *eur* above: therefore, we get that *BroadcastBucket** has a $O(t)$ running time, which, when wrapped with the binary search wrapper, means a $O(t\log|V|)$ algorithm. For small values of k relative to $|V|$, this running time may be significantly worse than $O(k\log k\log|V|)$, but as $k \to |V|$ this additional analysis is justified.

Chapter 4

A Heuristic for broadcasting on flower graphs

4.1 Preliminaries

It was observed in [9] that approximation algorithm S_{cycle}, the heuristic *BroadcastBucket* (and, it can be observed, Algorithms 1 and 4) all make the assumption that when broadcasting from v_c on any flower graph G, that the first call v_c makes should be to the largest cycle.

But this is not always the case: certain flower graphs have no optimal broadcast scheme with the first call going to the largest cycle. An example of this is given in [9].

Given a particular flower graph G, can we identify patterns where a particular round *must* or *must not* be assigned to a particular cycle in all optimal broadcast schemes? Heuristic 7, *ImprovedBroadcastBucket*, is a heuristic which seeks to do exactly this. When Heuristic 7 fails to identify any possible decision, it instead falls back to behaviour inspired from Algorithm 4.

Unlike Algorithm 1 and Algorithm 4, *ImprovedBroadcastBucket* only broadcasts from v_c. We assume that, in a similar manner to Algorithms 1 and 4, one can deal with cases where the originator $u \neq v_c$ on some cycle C_w by informing v_c as quickly as possible and using the broadcast scheme generated on $G' = (V(G) - V(C_w), E(G) - E(C_w))$.

Similarly to Algorithms 1 and 4, it takes as input a flower graph G with k cycles, and an integer t, the (suspected) broadcast time of G from v_c, its central vertex. It then either returns *False*, indicating it was unable to find a broadcast scheme for this t value, or it returns the scheme it managed to create for fully informing the graph G from v_c in t rounds. Using the wrapper from Algorithm 3, we can thus search for the smallest t

value for which we successfully find a broadcast time.

4.2 Intuition

Before giving our algorithm, we give the intuition behind the design. Suppose we are given some particular t value, and want to inform all cycles by round t. Consider looking at each round from the point of view of a particular cycle C_i.

C_i has two ways it may be fully informed by round R_t:

- v_c **makes a single call to a vertex of** C_i. So long as it does so at round R_{t-l_i+1} or earlier (assuming such a round exists), then C_i will be fully informed by round R_t.
- v_c **makes two useful calls to** C_i**, one to each vertices adjacent to** v_c**.** This can occur if the first call occurs at the earliest at R_{t-l_i+2}, requiring use to use a second call to finsih the cycle in time. How late the second call can be made depends on how early the first call was. So long as v_c makes the calls at rounds R_{t-l_i+1+x} and R_{t-x+1} at the latest, with $1 \leq x < l_i$, then C_i will be fully informed by round t.

Note that the above assumes that we do not assign any redundant calls: that is, if the first call to some C_i can fully inform it by R_t, then no second call is made to C_i. Of course, for any optimal broadcast scheme that exists and makes use of some redundant call, so does one exist without that redundant call, therefore this assumption is reasonable. In the rest of this section, we typically say that we only make *useful* or *non-redundant* calls.

Essentially, for each cycle, there exists some number of *candidate sets* containing either one or two rounds (which we will refer to as “single” or “pair” sets) which can be used to fully inform that cycle. Finding a broadcast scheme on t rounds then becomes simply the task of selecting, for each cycle, one particular candidate set such that we never have conflicting sets. Provided we have a way of considering all possible candidate sets, then we can use these sets to both identify where a call to a particular cycle *must* occur, and where a call to a particular cycle *cannot* occur (that is, the absence of some particular round(s) within all the candidate sets of one cycle can lead us to make decisions about the the other rounds or other cycles). We argue that one can thus look at these candidate sets to determine, for instance, that on some particular graph G and some number of rounds t, the first round *must not* be used for the largest cycle.

While we are building our partial broadcast scheme (that is, as we slowly assign rounds to particular cycles), considering all possible candidate sets and individually determining if a valid optimal scheme exists using that particular set rapidly becomes too computationally expensive. Likewise however, merely assuming that a particular candidate set can always be picked without considering the requirements of the other cycles, as does Algorithms S_{cycle}, $BroadcastBucket$, 1 or 4, can lead to locking ourselves into a partial broadcast scheme that cannot achieve the optimal broadcast time of the graph.

We have noticed a number of easily-identifiable situations where, when considering which candidate set to assign for some particular C_i, we can safely either:

- assign a particular candidate set to a particular cycle, guaranteeing that an optimal broadcast scheme exists using the resulting partial broadcast scheme.
- rule out an entire group of candidate sets, guaranteeing that no optimal broadcast scheme exists using any candidate set in the group.

On a high-level, *ImprovedBroadcastBucket* attempts to generate a broadcast scheme on exactly t rounds. It seeks to represent the candidate sets available using a small set of *indices* created for each cycle, which it places across the unassigned rounds according to particular rules described later. As it identifies situations where candidate sets can be assuredly used or ruled out, it moves these indices around across the set of unassigned rounds. The identification of these situations is done in a two-step process: First, it uses one set of rules to increment or decrement each indices. Then, it uses a second set of rules to assign rounds depending on the values of these indices. When none of the rules can be applied, it reverts to a default fallback behaviour, which is occasionally unoptimal and occasionally optimal. We show through simulations that this fallback rule is not optimal but manages to find near-optimal broadcast schemes. If at any point during execution, the algorithm finds that a cycle remains without any rounds assigned to it, yet has no more candidate sets remaining, it returns *False*, announcing that it failed to find a valid broadcast scheme.

4.3 The Heuristic

For our heuristic, we first need a formal way of representing a partial broadcast scheme. We also need a way of identifying whether the partial scheme can be made into an complete broadcast scheme or not.

Definition 1: Partial Scheme: *Given a flower graph G and a parameter $t \geq b(v_c, G)$, we call a* partial scheme *a set of calls from v_c to vertices adjacent to v_c.*
We say that this partial scheme is valid *if there exists a broadcast scheme that includes this set of calls, and successfully informs G in t rounds.*
When working on a valid partial scheme, when we assign some set of rounds to a particular cycle, we call this assignment a valid *assignment if our partial broadcast scheme remains valid after this assignment.*

Observe that, intuitively, provided $t \geq b(v_c, G)$, an empty partial scheme is initially valid. We can thus start with an empty partial scheme, and then use assignment rules that always make valid assignments to obtain a broadcast scheme.

We now highlight, relative to each unassigned C_i, that some particular rounds can be used to represent the *limit* of which rounds belong in a candidate set. Using these rounds, one can determine relatively quickly that whether a particular round can or cannot be used for a particular cycle, instead of having to verify the compatibility of every possible candidate set.

As we add assignments to our partial broadcast scheme, these particular rounds may change since candidate sets may become no longer viable.

Definition 2: Indices for a cycle C_i: *For any unassigned cycle C_i in a flower graph G and a valid partial broadcast scheme BR on t rounds, we define the following rounds:*

- *The **latest-single-call-round**: If it exists, the earliest round that meets the following two conditions:*
  - *the round is unassigned in BR.*
  - *For all broadcast schemes which fully inform G in t rounds and use all assignments in BR, if they inform C_i using a single call, this call uses the **latest-single-call-round** or an earlier round.*
- *The **first-split-round**: If it exists, the latest round that meets the following two conditions:*
  - *the round is unassigned in BR.*
  - *For all broadcast schemes which fully inform G in t rounds and use all assignments in BR, if they inform C_i using two non-redundant calls, their first call uses **latest-single-call-round** or a later round.*
- *The **halfway-point**: This round must exist for all unassigned rounds, else our partial scheme BR is invalid. The earliest round that meets the following two conditions:*
  - *the round is unassigned in BR.*
  - *For all broadcast schemes which fully inform G in t rounds and use all assignments in BR, their first (and possibly only) call to C_i uses **halfway-point** or an earlier round.*
- *The **last-split-round**: If it exists, the earliest round that meets the following two conditions:*
  - *the round is unassigned in BR.*

- *For all broadcast schemes which fully inform G in t rounds and use all assignments in BR, if they inform C_i using two non-redundant calls, their second call uses* ***last-split-round*** *or an earlier round.*

Note well that the above are defined only with respect to a particular valid partial broadcast scheme: In other words, given some valid partial scheme BR, there could be some unassigned round R_x which can fully inform C_i using a single call, but no existing complete broadcast scheme that uses all assignments in BR, and also informs C_i using a single call made at round R_x. In this case, we can have our **latest-single-call-round** necessarily appears strictly earlier than R_x.

Note additionally that for cycles containing exactly 3 vertices, since one cannot reasonably ponder whether to call these cycles once or twice from v_c. We thus acknowledge that the definition above doesn't have much meaning for cycles of exactly 3 vertices, but note that, if a graph has an optimal broadcast scheme on t rounds, one can seek to find such a broadcast scheme by first using the latest calls v_c can make (except during the very last round) to inform any cycle on 3 vertices this graph has. This is in fact exactly what Heuristic 7 does.

Given a particular partial broadcast scheme BR, Heuristic 7 will attempt to determine whether or not the rounds defined in Definition 2 exist and, if so, to approximate their location as closely as well as it can. For every unassigned cycle C_i, it will define the variables S_i, F_i, H_i, P_i, and L_i respectively, which it will refer to as the **indices** of C_i. These indices will meet the same requirements as in Definition 2, except that they may not be the *latest* or *earliest* round that fullfill the requirements given in Definition 2: for instance, Heuristic 7 may slightly overshoot the **latest-single-call-round** by instead referring to some R_{S_i} that appears significantly later. As Heuristic 7 progresses, it will move these indices across to different rounds depending on the availability and usefulness of the rounds, with respect to C_i.

Taking again the example of S_i and the **latest-single-call-round** for a particular C_i, Heuristic 7 will try to find the best value of S_i it can infer, such that R_{S_i} gets as close to the true **latest-single-call-round** of C_i as possible (provided it exists). It will additionally attempt to infer whether or not no such round exists, and delete S_i if it does infer this.

Refer to Figure 4.1 for a representation of the t rounds from the view of two potential cycles, C_1 with $l_1 >> t$ and C_2 with $l_2 << t$.

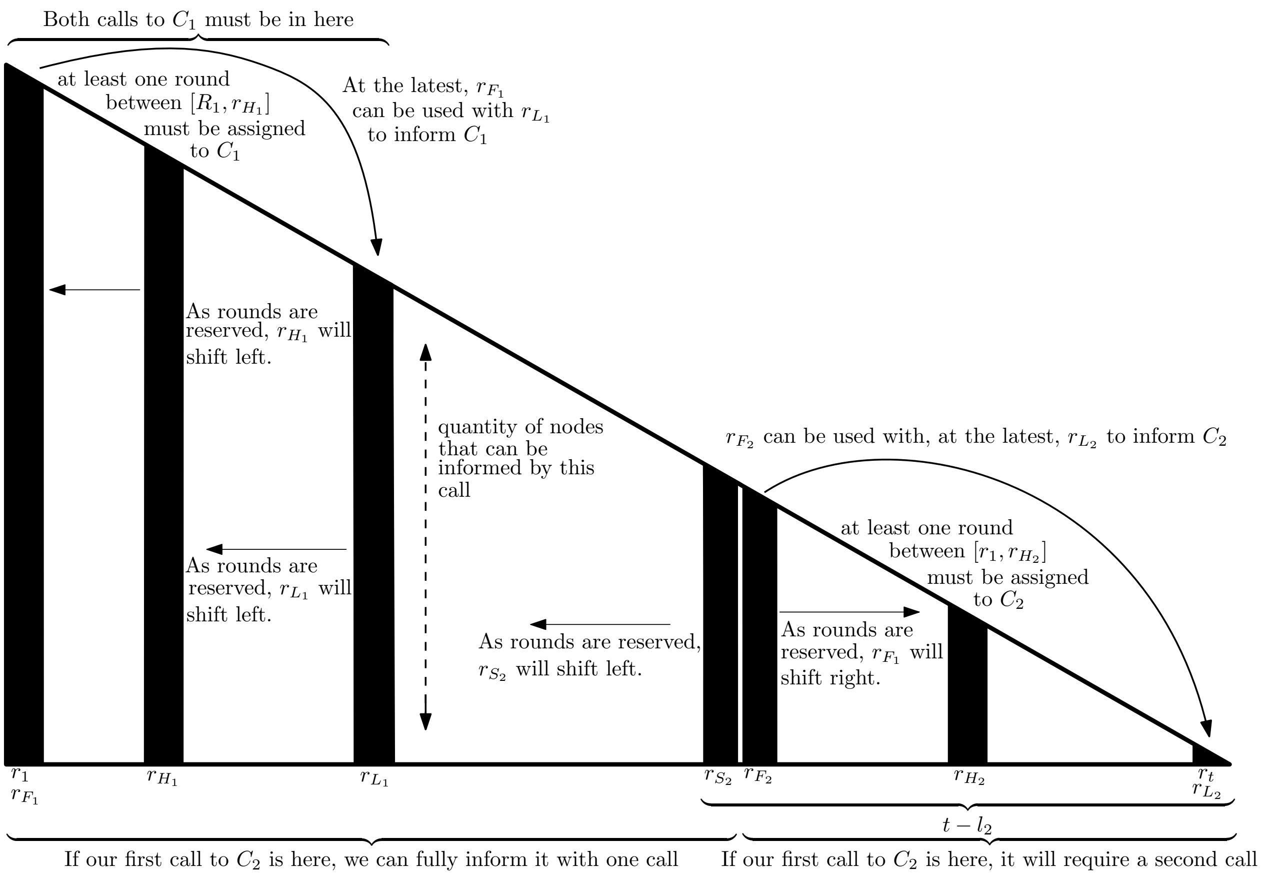

Both calls to C_1 must be in here
at least one round between $[R_1, r_{H_1}]$ must be assigned to C_1
At the latest, r_{F_1} can be used with r_{L_1} to inform C_1
As rounds are reserved, r_{H_1} will shift left.
quantity of nodes that can be informed by this call
As rounds are reserved, r_{L_1} will shift left.
As rounds are reserved, r_{S_2} will shift left.
r_{F_2} can be used with, at the latest, r_{L_2} to inform C_2
at least one round between $[r_1, r_{H_2}]$ must be assigned to C_2
As rounds are reserved, r_{F_1} will shift right.
r_1
r_{F_1}
r_{H_1}
r_{L_1}
r_{S_2}
r_{F_2}
r_{H_2}
r_t
r_{L_2}
$t - l_2$
If our first call to C_2 is here, we can fully inform it with one call
If our first call to C_2 is here, it will require a second call

Figure 4.1: A visual representation of the indices of two cycles C_1 and C_2, with with $l_1 >> t$ and C_2 with $l_2 << t$.

To ensure that the indices always “overshoot” the rounds of Definition 2, we define the following requirements, which we will always uphold:

Definition 3: Validity of Indices: *For any unassigned cycle C_i in a flower graph G and a partial broadcast scheme BR on t rounds, given any such index S_i, F_i, H_i, P_i, or L_i, we say that the index is* valid *if for all complete broadcast schemes that use all assignments in BR and fully inform G in t rounds:*

1. *If the index is of type S_i: If C_i is informed with one call then the call occurs at R_{S_i} or at an earlier round, and $S_i \leq H_i$.*
2. *If the index is of type F_i or L_i: If C_i is informed with two* non-redundant *calls, then the first call occurs at R_{F_i} or a later round, and the second call occurs at R_{L_i} or an earlier round.*
3. *If the index is of type H_i or P_i: If C_i is informed with two* non-redundant *calls, then the first call occurs at R_{H_i} or an earlier round. If the first call occurs exactly at R_{H_i}, then the second call occurs*

at and R_{P_i} or later.

4. *If the index is of type H_i: The first call to C_i is made at R_{H_i} or an earlier round.*

5. *If S_i does not exist, then C_i is never informed with a single call.*

6. *If F_i does not exist, then C_i is only informed with a single call.*

Note well that unlike Definition 2, we do not presume to work on a valid partial broadcast scheme. We merely instead state that, *if* a valid broadcasts scheme exists, indices that follow the requirements in Definition 3 can be used to somewhat reasonably simulate the rounds in Definition 2.

Notice also that we use the additional index P_i, which does not have an round it seeks to emulate in Definition 2. We will instead use this index to more accurately determine whether or not we need to move H_i to an earlier round.

At the beginning of Heuristic 7, we initialise these indices to the following values:

Definition 4: Indices Initialisation Procedure:

- *S_i:*
  - *if $l_i > t$, don't create S_i*
  - *if $l_i \leq t$, $S_i := t - l_i + 1$*
- $F_i := \max(1, t - l_i + 2)$
- $H_i := t - \lfloor \frac{l_i}{2} \rfloor$
- $P_i := t - \lfloor \frac{l_i}{2} \rfloor + 1$
- $L_i := \min(2t - l_i + 1,\ t)$

We first demonstrate that the above indices are initialised to be valid:

Lemma 11: *Given a flower graph G and a broadcast time $t \geq b(v_c, G)$, the indices generated by Definition 4 are valid, as according to Definition 2.*

Proof. Consider first claim 1 of Definition 3: if $l_i > t$, then we have to inform this cycle with two calls since no single call can completely inform C_i in time, and we correctly represent this by not creating an S_i. Recall the lower bound of 2 on cycle length from Section 3.1. If $2 \leq l_i \leq t$ then $1 \leq t - l_i + 1 \leq t - 1$, so R_{t-l_i+1} always exists. Using R_{t-l_i+1} lets us inform $t - (t - l_i + 1) + 1 = l_i$ vertices by R_t. Using a single call occuring during any rounds after it will fail to fully inform C_i by R_t. Thus, after initialisating S_i as dictacted in Definition 4, S_i correctly represents the index of the latest round that can fully inform C_i, if this is possible. For H_i, since we can inform C_i with one call then S_i exists. Since $l_i \geq 2$, then $S_i = t - l_i + 1 < t - \lfloor \frac{l_i}{2} \rfloor$, so we have that $S_i < H_i$ as required.

Consider now claim 2. Recall that a redundant call is a call that can be omitted from the broadcast scheme, and C_i would still be fully informed by R_t. Observe first that since $l_i \geq 2$ for all C_i, then $t - l_i + 2 \leq t$, so R_{F_i} always exists.

By the above, it thus must be that our earliest first round occurs at R_{t-l_i+2}, otherwise our second call would be redundant. Thus F_i's initial value correctly refers to the earliest round that can be used to inform C_i, but that requires a second call to be made to C_i. For the second call, observe first that it must be that $2t - l_i + 1 \geq 2$: we assumed that $t \geq b(v_c, G)$, and we know By Lemma 9 that $l_i \leq 2b(v_c, G) - 2i + 1 \leq 2t - 1 \rightarrow 2t - l_i + 1 \geq 2$. This is intuitively logical: we cannot use R_1 for our second call since the second call must occur after the first call, and the earliest round that the first call can use is R_1. Consider when $2t - l_i + 1 < t$: If we use any round strictly after R_{2t-l_i+1}, then that call informs $t - (2t - l_i + 2) + 1 = l_i - t - 1$ rounds at most. Thus, our first call needs to inform at least $l_i - (l_i - t - 1) = t + 1$ vertices, which isn't possible with t rounds. Thus it must be that our second call occurs at R_{2t-l_i+1} or earlier (but after the round for the first call, of course). Thus L_i's initial value correctly refers to the latest round that can be used as a second call to C_i, and successfully inform C_i if paired with an early-enough first call.

Next, consider claim 3. Observe that if we instead tried to use $R_{t-\lfloor \frac{l_i}{2} \rfloor + 1}$ as our first call we'd inform $\lfloor \frac{l_i}{2} \rfloor$ vertices, and thus have $\lceil \frac{l_i}{2} \rceil$ vertices left to inform. This cannot be fullfilled by any round strictly after R_{H_i+1}, thus this value fails to meet our definition of H_i. Observe however that using $R_{t-\lfloor \frac{l_i}{2} \rfloor}$ as our first call, we can pair it with $R_{t-\lfloor \frac{l_i}{2} \rfloor + 1}$ and successfully fully inform C_i by R_t. Thus, H_i's initial value correctly refers to the latest round by which a call to C_i must have been made. Observe finally that $R_{P_i} = R_{H_i+1}$, which is the earliest possible round we can use while still using a round that occurs strictly after R_{H_i}, so if R_{H_i} does end up being used for C_i, the second call must occur at R_{P_i} or later.

Finally, consider claims 5 and 6: if $l_i > t$, then it is indeed true that we cannot inform C_i with a single call since we can at most inform t vertices with a call at R_1. Additionally, F_i always exists, therefore claim 6 is always valid. □

How to store and handle these indices will be discussed later when dealing with the running-time of the algorithm. For now, we give a small subroutine for tightening the values of these indices as we assign rounds in our algorithm:

Algorithm 5 LowerBoundSubroutine
Input: a flower graph G with k cycles sorted from largest to smallest, a partial broadcast scheme BR, valid indices for all unassigned cycles of G
Output: updated, valid indices for all unassigned cycles of G, or $False$

```
1:  $UnassignedRoundsCount := 0$, $RoundsNeeded := 0$, $LeftIndex := 1$
2:  $Count :=$ a $k$-sized integer array initially all set to 0
3:  for $j == 1$ to $t$ do
4:      if $R_j$ is unassigned in $BR$ then $UnassignedRoundsCount += 1$
5:      for all indices $X_i = j$ do
6:          if $X_i$ is of type $H_i$ then $Count[i] += 1$, $RoundsNeeded += 1$
7:          else if $X_i$ is of type $L_i$ and $S_i$ does not exist then $Count[i] += 1$, $RoundsNeeded += 1$
8:      if if $UnassignedRoundsCount > 0$ and $UnassignedRoundsCount == RoundsNeeded$ then
9:          for all unassigned $C_i$ do
10:             if $Count[i] == 1$ then
11:                 if $H_i \geq LeftIndex$ and $H_i > 2t - l_i - j + 1$ then
12:                     if there exists at least one unassigned round $R_x$ where $x <= 2t - l_i - j + 1$ then, set $H_i$
    to the latest such round's index, else Return $False$
13:             else if $Count[i] == 0$ then
14:                 for $X_i$ of type $S_i$, $H_i$, or $L_i$ do
15:                     if $LeftIndex \leq X_i \leq j$ then
16:                         if there exists at least one unassigned round $R_x$ where $x < LeftIndex$ then, set $X_i$
    to the latest such round's index, else delete $X_i$
17:                 for $X_i$ of type $F_i$ or $P_i$ do
18:                     if $LeftIndex \leq X_i \leq j$ then
19:                         if there exists at least one unassigned round $R_x$ where $x > j$ then, set $X_i$ to the
    earliest such round's index, else delete $X_i$
20:     else if $UnassignedRoundsCount < RoundsNeeded$ then Return $False$
21:         $UnassignedRoundsCount = 0$, $RoundsNeeded = 0$, $LeftIndex = j + 1$
22:         Reset all values in $Count$ to 0
23: Return all indices
```

Lemma 12: *Given a graph G, a partial broadcast scheme BR, a set of valid indices for all cycles unassigned in BR, then:*

- *if Subroutine 5 returns False, then BR was invalid.*
- *if Subroutine 5 returns a set of indices, then these indices are still valid.*

Proof. We first highlight that this lemma does not claim that Subroutine 5 detects *all* cases where no broadcast scheme can be made using all rounds from *BR*, simply that the cases it does detect are indeed when *BR* is invalid.

We now explain how Subroutine 5 actually behaves. In short, it completes a single pass on the set of rounds from R_1 to R_t with a pointer R_j where $j = 1$, and where it slowly increments this pointer until reaching t. During this pass, it keeps track of two counts: first, the number of unassigned rounds from R_1 to R_j, stored in $UnassignedRoundsCount$. Second, the number of times a cycle declares it absolutely needs a round to be assigned to it from the set $\{R_{LeftIndex}, \dots, R_j\}$, stored in $RoundsNeeded$. It also keeps track of which cycles are declaring these instances, and how often they did so, using an array $Count$.

Whenever these $UnassignedRoundsCount$ and $RoundsNeeded$ are equal to each other (and are not 0), then it can use this to make inferences about which candidate sets can be completely ruled out: For instance, if some cycle C_i declared only needing exactly one round in $\{R_1, \dots R_j\}$, then using a candidate set that uses two rounds in $\{R_1, \dots R_j\}$ is out of the question: doing so would use an additional round for C_i that it didn't request, and would leave us with too few unassigned rounds in $\{R_1, \dots R_j\}$ to fulfill the requirements of the other unassigned cycles that declared needing rounds in $\{R_1, \dots R_j\}$.

Once Subroutine 5 moves the indices of each cycle to reflect these candidate sets being ruled out, it resets $UnassignedRoundsCount$, $RoundsNeeded$, $Count$, and sets $LeftIndex = j + 1$. Indeed, we can now consider that all of the unassigned rounds in the set $\{R_{LeftIndex}, \dots, R_j\}$ are reserved, even if we aren't exactly sure how they are assigned yet: all we know is that some cycle C_i may be assigned 0, 1, or 2 rounds in this set depending on its $Count[i]$ value. Therefore, Subroutine 5 can now look for another equality of the two counts at some later $j' > j$, and therefore find another place where it can reserve all the unassigned rounds of the set $\{R_{j+1}, \dots, R_{j'}\}$.

If $UnassignedRoundsCount$ ever goes below $RoundsNeeded$, then assuming we are correctly counting the number of unassigned rounds needed, this would thus imply that no complete broadcast scheme on t rounds and using all the assignments from *BR* can exist, since no broadcast scheme has enough unassigned rounds before and including R_j to fulfill the requirements of the unassigned cycles remaining.

If instead $UnassignedRoundsCount$ remains equal to or above $RoundsNeeded$, then there can exist such complete broadcast schemes. What we instead need to demonstrate is that our indices remain valid.

To do so, we need to do two things: First, we need to demonstrate that we are correctly counting the number of cycles absolutely *needed* by the unassigned cycles.

Second, we need to, from this, demonstrate that the changes applied to the indices, which are applied when the If statement at Line 8 is true, never invalidates an index.

Consider first the way in which we count the rounds needed by an unassigned C_i:

- At Line 6, we count the index H_i: by definition of the **halfway-point** and our assumption that the indices given to us are valid, we know that we will indeed need at least one round by R_{H_i} with which to make our first call. By the above, we know that all the rounds before $R_{LeftIndex}$ are reserved, and therefore cannot be used for us. Therefore, it is correct to increment the count here and claim we need a round in the set $\{R_{LeftIndex}, \ldots, R_{H_i}\}$. One can make the argument that we should increment P_i as well, however, this will be handled by another rule. Even when not incrementing P_i, we remain valid.

- At Line 7, we count the index L_i only if S_i does not exist: since our indices are valid, we first know that we must inform C_i with two calls, and we know from L_i's validity that, at the latest, the second call must occur that R_{L_i}. Wherever we incremented the count for H_i previously, this increment only counted the first call. Similarly to the case for H_i, we know that all the rounds before $R_{LeftIndex}$ are reserved, and therefore cannot be used for us. Therefore, it is correct to increment the count here and claim we need a round in the set $\{R_{LeftIndex}, \ldots, R_{F_i}\}$.

Therefore we correctly count the number of rounds needed by C_i.

We now show that the changes made when Line 8 is true never invalidate the indices.

Three changes are applied: The first is when $Count == 1$: in this case, at Line 12, if $H_i \geq LeftIndex$ and $H_i > 2t - l_i - j + 1$, H_i is set to $2t - l_i - j + 1$.

Consider first under which conditions this is true: since $H_i \geq LeftIndex$, then we granted one round to C_i for its first call. However, since $Count[i] == 1$, then either $L_i > j$, or S_i exists. In this case, we can effectively rule out any instance where C_i is granted two rounds in the set $\{R_{LeftIndex}, \ldots, R_j\}$: either C_i must be informed with a single call made at R_{S_i} or earlier, or it is informed with two calls, the first of which occurs early enough such that the second call is made at R_{j+1} or later. How early must this first call occur? The second call, at best, can inform a total of $t-(j+1)+1$ vertices. Therefore, we have at least $l_i-(t-(j+1)+1)$ left to inform with the first call. Thus, it must be that our first call occurs at the latest at R_x where:

$$t-x+1 \geq l_i-(t-(j+1)+1)$$
$$2t-l_i-j+1 \geq x$$

Thus we can safely set $H_i = 2t-l_i-j+1$.

Consider now when $Count[i] == 0$. In this case, C_i cannot use any rounds in the set $\{R_{LeftIndex},\ldots,R_j\}$. If we do use such a round for C_i, we will not have enough unassigned rounds left in the set to fullfill the demands of the other unassigned cycles that declared needing rounds in the set. We can therefore safely move all of C_i's indices out of the set as complete valid broadcast scheme on t rounds and using all of BR's assignments can exist that would use such a round for C_i. Since $Count[i] == 0$, then R_{H_i} is not in this set, so we don't need to change its value. In the case of S_i, F_i, P_i, or L_i, we need to move them to the first available unassigned round not in the set (if it exists) in the direction dependent on the type of index we are working with: S_i and L_i move to earlier rounds, P_i and F_i moves to later rounds. If no such round exists, we can then instead remove this index. Since using any round in $\{R_{LeftIndex},\ldots,R_j\}$ for C_i would invalidate our partial broadcast scheme, and since we move to the first available unassigned round not in the set, our indices remain valid.

Thus, Subroutine 5 returns valid indices. □

We now give small additional rules for updating the indices:

Definition 5: Rules for Updating The Indices: *For any C_i, Given any set of indices S_i, F_i, H_i, and L_i (or none of them), and some partial broadcast scheme BR, then:*

1. *If $L_i > 2t+2-l_i-F_i$, set L_i to the index of the latest unassigned round that appears at least as early as $R_{2t+2-l_i-F_i}$.*
2. *if $F_i == L_i$, delete F_i and L_i.*
3. *if only one of F_i or L_i exists, delete the other one.*
4. *if $S_i == H_i$, delete F_i and L_i.*
5. *if $H_i > 2t+2-l_i+P_i$, set H_i to the index of the latest unassigned round that appears at least as early as $R_{2t+2-l_i-F_i}$. Then, set P_i to the index of the latest unassigned round that appears at least*

as early as R_{H_i+1}.

6. *if C_i has no S_i, F_i, and L_i, return $False$.*

Similarly to Lemma 12, we will demonstrate that these rules never invalidate valid indices:

Lemma 13: *Given a graph G, a partial broadcast scheme BR, a set of valid indices for all cycles unassigned in BR, then applying any rule in Definition 5 returns a set of valid indices.*

Proof. Consider first Rule 1:

The rounds $\{R_{F_i} \dots R_{L_i}\}$ are only useful to C_i if they can be paired with another round within that set. If some solution used any round appearing before that set to inform C_i, then using any other round to inform C_i from the other side would be redundant. By definition of R_{L_i}, no solution exists using any round appearing after that set to inform C_i (unless that call is redundant, and thus can be ignored).

Consider thus instead the earliest round that C_i can use, which still needs a second call to completely inform C_i by R_t. By definition, this round is R_{F_i}. This round can only inform $t - F_i + 1$ vertices. This means that our second call needs to be able to inform at least $l_i - (t - F_i + 1)$ vertices: $t - x + 1 \geq l_i - (t - F_i + 1) \rightarrow 2t - l_i + 2 - F_i \geq x$. Thus, using some round strictly after $R_{2t-l_i+2-F_i}$ would mean we fail to fully inform C_i in time. Instead, using such a round would force us to use a round strictly after R_{F_i}. The first such unassigned round, if it exists, is R_{S_i}. This would make our second call redundant. Thus, we can safely set L_i to the latest unassigned round that appears at least as early as $R_{2t-l_i+2-F_i}$.

Consider now Rule 2: Since $F_i == L_i$, we cannot inform C_i using two (useful) calls. We must inform C_i using a single call. Thus, we can safely remove F_i and L_i since no broadcast scheme can usefully attempt to use any round after R_{S_i} to inform C_i. Similar logic can be appled for Rules 3 and 4.

Consider now Rule 5: very similarly to Rule 1, R_{H_i} can only usefully be used in conjunction with a round R_{P_i} or later. Thus, if $H_i > 2t + 2 - l_i + P_i$, then since we are assuming that P_i is valid (and no previous rule ever invalidates P_i), then no round strictly after $R_{2t+2-l_i+P_i}$ can be usefully paired with R_{P_i}. We must move H_i to a round with index at most $2t + 2 - l_i + P_i$. However, having now done so, it is possible that some round that appeared at or before H_i's previous value can now be used as a valid second round. Thus, we must move P_i as early as we can: to R_{H_i+1}.

Finally, for Rule 6, we can simply observe that if we no longer have either S_i (and thus cannot inform C_i with a single call) nor F_i and L_i (and thus cannot inform C_i with two useful calls), then we cannot inform C_i in t rounds with our current BR scheme. We need to exit out of this instance and try again with a larger t value. □

We now give the rules for assigning cycles based on the values of their indices. First, we give a similar rule to Subroutine 5:

Algorithm 6 UpperBoundSubroutine
Input: a flower graph G with k cycles sorted from largest to smallest, a partial broadcast scheme BR, valid indices for all unassigned cycles of G
Output: A partial scheme BR

```
1:  $UnassignedRoundsCount := 0$, $RoundsCanUse := 0$
2:  $Count$ := a $k$-sized integer array initially all set to 0
3:  for $j == 1$ to $t$ do
4:      if $R_j$ is unassigned in $BR$ then $UnassignedRoundsCount += 1$
5:      for all indices $X_i = j$ do
6:          if $X_i$ is of type $H_i$ or type $L_i$, then $Count[i] += 1$, $RoundsNeeded += 1$
7:      if if $UnassignedRoundsCount > 0$ and $UnassignedRoundsCount < RoundsCanUse$ then
8:          if there exists at least one unassigned $C_i$ where $Count[i] == 2$ and $S_i$ exists then
9:              Take the largest such $C_i$, and assign $R_{S_i}$ to $C_i$ in $BR$
10:             Return $BR$
11: Return $BR$
```

Lemma 14: *Given a graph G, a* valid *partial broadcast scheme BR, a set of valid indices for all cycles unassigned in BR, then Subroutine 6 returns a valid partial broadcast scheme BR.*

Proof. We refer to the proof of Lemma 12 for an explanation of the basic logic of the traversal of Subroutine 6, since it works in a similar but simpler manner to Subroutine 5.

The key difference is the following: instead of counting the number of rounds that C_i needs in the set $\{R_{LeftIndex}, \ldots, R_j\}$ (which, in this context, can be simplified to counting the number of rounds needed in the set $\{R_1, \ldots, R_j\}$), Subroutine 6 counts using variable *RoundsCanUse* the number of rounds that C_i can *at most* usefully use. That is, it assumes thus that *BR* decides to inform C_i using two calls instead of one. In essence, we seek to identify whether or not we have enough unassigned rounds to call *all* cycles that *can* be informed using two rounds: if not, then we can conclude that there is a cycle which *must* be called only once, even if there does exist some unassigned rounds which it could usefully use to be fully informed by R_t.

Indeed, we verify two conditions: is $H_i \leq j$? and: is $L_i \leq j$?

If the first condition is true, we will necessarily need (and thus can usefully use) one round in

$\{R_{LeftIndex},\ldots,R_j\}$. If the second condition is also true, we may inform C_i with a single call occuring at or before R_{S_i}, or we can inform C_i with one call occuring at or before R_{H_i} and a second occuring at or before R_{L_i}. Therefore we correctly calculate the guarantee that we can usefully use at most two rounds in $\{R_1,\ldots,R_j\}$.

If $UnassignedRoundsCount$ ever dips strictly below $RoundsCanUse$, We must therefore find a cycle which can afford to give up such a round. Recall that we demonstrated in the proof of Lemma 12 that the way to obtain the number of rounds a cycle *needs* in the set $\{R_1,\ldots,R_j\}$ is by essentially verifying these two conditions: is $H_i \leq j$? and: is $L_i \leq j$ and does S_i not exist?

By comparing this with the way in which we count $RoundsCanUse$, we can see thus that the only moment where these two counts differ is when $H_i \leq j$, $L_i \leq j$, and S_i exists: $RoundsCanUse$ will allow C_i to reserve an additional round, but not $RoundsNeeded$. Thus, we must use a cycle that meets this condition if we are to inform it with only one round: taking a round away from a cycle where $RoundsCanUse == RoundsNeeded$ would mean creating an invalid partial broadcast scheme.

Suppose, by assumption, that BR is a valid partial broadcast scheme and our indices are valid. Suppose also that we find some j wherein $UnassignedRoundsCount < RoundsCanUse$. If no cycle meets the condition described above, then we merely return BR unchanged, which will thus remain valid.

If instead there does exist a cycle C_i such that $H_i \leq j$ and $L_i \leq j$ and S_i exists. Then, $Count[i] == 2$ and S_i exists, so Line 8's condition is true. There, we take the largest C_i that meets the above condition, and assigns to it R_{S_i}. We now claim that the resulting BR' is still valid.

Suppose instead by contradiction that this BR' is invalid. Then, it must also be that any broadcast scheme which grants to C_i a round strictly before R_{S_i} is also invalid, otherwise we could simply take a valid broadcast scheme which grants such a round to C_i, swap these two calls and obtain a valid broadcast scheme where C_i is informed using R_{S_i}. However, BR is valid by assumption, therefore there does exist some particular set of assignments we can add to BR to make a valid complete broadcast scheme. By definition and validity of S_i, it must thus be that in any such broadcast scheme, C_i is informed using two calls with the first call occuring strictly after R_{S_i}. Consider any such complete broadcast scheme, call it $\hat{BR}$. Since $\hat{BR}$ made the same assignments as BR, then it, too, had to deal with the issue explained above: that is, it had a list of cycles which can potentially use, in total, $RoundsCanUse$ rounds in the set $\{R_1,\ldots,R_j\}$, but it did not have enough unassigned rounds in this set to grant all the cycles concerned the maximum quantity of rounds they requested. Thus, it must have chosen some C_x where $Count[x] == 2$ and S_x existed, and informed with a single call using R_{S_x} or an earlier round. Call this round R_y. It must be that $y > S_i$: otherwise, in $\hat{BR}$, we could merely

swap the calls for C_i for the call for C_x, and obtain a valid broadcast scheme. But, likewise, it must be that l_x isn't smaller than l_i, otherwise we can swap C_x in the call for C_i and inform C_x fully with the two calls that inform C_i, and thereby obtaining a complete broadcast scheme where C_i is informed with only one call. Contradiction. Thus, a a valid broadcast scheme must exist where C_i is informed in round R_{S_i}.
Therefore, Subroutine 6 returns a valid partial broadcast scheme BR. □

We now give a small set of additional rules for assigning cycles based on the values of their indices.

Definition 6: Assignment Rules: *Given a graph G, a* valid *partial broadcast scheme BR, a set of valid indices for all cycles unassigned in BR, then:*

- *If a cycle C_i is the largest unassigned cycle and S_i exists, assign R_{S_i} to C_i.*
- *If a cycle no longer has any F_i and S_i, assign R_{S_i} to C_i.*
- *If there exists a cycle C_i such that S_i doesn't exist, and:*
  - *if $F_i == H_i$, then assign R_{F_i} and R_{L_i} to C_i.*
  - *else if $P_i == L_i$, then assign R_{H_i} and R_{P_i} to C_i.*

Lemma 15: *Given a graph G, a* valid *partial broadcast scheme BR, a set of valid indices for all cycles unassigned in BR, then applying any of the rules in Definition 6 exactly once returns a valid partial broadcast scheme BR.*

Proof. For rule 6, consider the largest unassigned cycle C_l, and the earliest unassigned round R_j. Since we started with a valid partial broadcast scheme, there must be a some complete broadcast scheme that uses all assignments of BR. Consider such a broadcast scheme $\hat{BR}$, and suppose that in $\hat{BR}$, C_l is not informed using R_j. Consider the cycle C_x that $\hat{BR}$ does inform in R_j: By assumption, $l_x \leq l_l$. We thus can simply have C_x use the rounds that $\hat{BR}$ uses to inform C_l, and inform C_l with R_j: since S_i exists, $S_i \geq j$. So, if there exists a complete broadcast scheme that can inform G in t rounds, then there must exist a complete broadcast scheme that uses R_j to inform C_l, that can inform G in t rounds.
By very similar logic, rule 6 only makes valid assignments.
Rule 6 essentially says: If you must split your cycle into two useful calls and only one round exists in the set $\{F_i \dots H_i\}$ or $\{P_i \dots F_i\}$, assign that round and the latest round that can be paired with it for C_i.

Consider first the case where $F_i == H_i$. Since S_i doesn't exist, then we must split our cycle in two calls. Since $F_i == H_i$, then R_{H_i} must be used to inform C_i since no other rounds can be used for the first call, and we know that a complete broadcast scheme exists since BR is assumed to be valid. Additionally, the latest round that can be used with R_{H_i} to fully inform C_i is L_i: L_i would have its value decrease until it pointed to $t-(C_i-(t-F_i))$ by Rule 1 when updating indices. It must also be that $L_i \neq F_i$, otherwise Rule 2 would have remove F_i and L_i. So, R_{L_i} is available for C_i, can be used with R_{H_i} to fully inform C_i, and is the latest round available to be validly paired with H_i. A very similar logic can be used for the case where $P_i == L_i$.
Thus, for all rules in Definition 6, we obtain a valid partial broadcast scheme when starting with a valid partial broadcast scheme. □

We highlight that any time after assigning a round to a cycle using one of the rules from Definition 6, then all indices previously pointing to this round need to be updated to remain valid, since they are no longer pointing to an unassigned round. We will do this in Heuristic 7.
We finally a default assignment rule, which we can use when we get stuck and cannot take a decision, chosen based on the performance of Algorithm 4.

Definition 7: Default Rule: *Assign to C_i R_{L_i}. Then, assign to C_i the latest unassigned round that can be usefully paired with R_{L_i}.*

Note that the *the latest unassigned round that can be usefully paired with R_{L_i}* is not always R_{F_i}, it is occasionally R_{F_i+1} depending on whether l_i is odd or even, and depending on the availability of rounds.
We now group all of the above together in our heuristic:

Algorithm 7 ImprovedBroadcastBucket
Input: a flower graph G with k cycles sorted from largest to smallest, and a broadcast time t
Output: A broadcast scheme for u, or the boolean value $False$

```
1:  $L =$ list of indices initialised with Definition 4
2:  $BR = [0] * t$
3:  Assign all cycles $C_i$ of $l_i = 2$ to the latest rounds available (except the very last round), in any order.
4:  while Some cycles are still unassigned do
5:      $OldIndices := L$
6:      $L = LowerBoundSubroutine(G, BR, L)$
7:      if $L = False$ then Return $False$
8:      for every unassigned cycle $C_i$ do
9:          Apply all Rules of Definition 5 which can be applied.
10:         for every unassigned cycle $C_i$ do
11:             if $C_i$ does not have $S_i$, $F_i$, and $L_i$ then Return $False$
12:     for every Assignment Rule in $\{UpperBoundSubroutine$, rules from Definition 6$\}$ do
13:         Try to make an assignment in $BR$ with the Assignment Rule
14:         if the Assignment Rule made an assignment to some round $R_j$ then
15:             for All $1 \leq i \leq k$, for any index $S_i$, $H_i$, or $L_i = j$ do
16:                 if there exists at least one unassigned round $R_x$ where $x < LeftIndex$ then, set all these
        indices to the latest such round's index, else delete these indices.
17:             for All $1 \leq i \leq k$, for any index $P_i$ or $F_i = j$ do
18:                 if there exists at least one unassigned round $R_x$ where $x > LeftIndex$ then, set all these
        indices to the earliest such round's index, else delete these indices.
19:             Break out of Loop starting at line 10
20:     if $L == OldIndices$ and no Assignment $Rule$ could be applied then
21:         make an assignment in $BR$ using Default Rule 7
22: Return $BR$
```

Theorem 4: Heuristic Validity: *Given a broadcast scheme G and a value $t = b(v_c, G)$, Heuristic 7 returns an optimal broadcast scheme if it doesn't return False and never falls back to its Default Rule.*

Proof. According to Lemma 11, we initialise on valid indices. We also start with a valid broadcast scheme

BR, since our broadcasts scheme is empty and $t \geq b(v_c, G)$.

According to Lemma 12 and 13, given a set of valid indices, the indices remain valid after having been modified by their respective rules. According to Lemma 14 and 15, when given a set of valid indices and a valid partial broadcast scheme *BR*, the assignments made keep by their respective rules keep *BR* valid.

Thus, Heuristic 7 returns an optimal broadcast scheme if it never falls back to its Default Rule. □

One may question the utility of the above theorem, since, by definition, if $t = b(v_c, G)$ and Heuristic 7 successfully returns a broadcast scheme, then we have found an optimal broadcast scheme. However, the intent behind it is simple: provided we can find enough rules such that we can *always* make some valid assignment without falling back to some Default rule, then we can argue that our heuristic is an algorithm which solves the broadcast problem on Flower Graphs. Unfortunately however, we were unable to find rules which both cover *all* possible situations and can be proven to *always* make valid assignments. The strength of this theorem lies in the idea that, should a new rule be found, that it can be added to the Heuristic 7 such that we may perhaps eventually be able to prove that the Heuristic never has to fall back to a default rule.

4.4 Running Time

We now explain how to achieve a reasonable running time.

The crucial part is the data structure for referring to the indices of each unassigned cycle. In short, we will maintain each index in two structures:

First, initialising all index values using Definition 4 is just an $O(k)$ operation, since all values can be calculated in constant time.

For storage, we may keep an array *A* such that we can access the value of each index for each cycle in $O(1)$ time, since all cycles will keep four indices (some of which may be simply set to some Null value, to indicate their non-existence.)

We will secondly keep a linked list *B* of *t* buckets, with some bucket $B[x]$ containing any index with value x. For instance, if $H_i = x$, then it will be kept in bucket $B[x]$. Each bucket $B[x]$ will maintain a flag to indicate whether R_x is already assigned in our partial broadcast or not, and will maintain pointers to the closest *previous* and *next* buckets referring to unassigned buckets. Every bucket will maintain the indices stored in them using a binary heap, allowing $O(\log(k))$ inserts and removals. Thus, we can update the value of an index in both *A* and *B* in $O(1) + O(\log(k))$.

Whenever a round is assigned to a cycle, its bucket can in $O(t)$ time alert all other buckets that their *previous* and *next* pointers may need to be updated. *All* other buckets need to be alerted: we need to ensure that even the buckets representing an already-assigned round maintain correct pointers. Any bucket's pointers may be used by Subroutine 5, the Rules 5, or even Lines 15 and 17 in Heuristic 7 to determine where an index needs to be moved to (what value it needs to be changed to). By maintaining these pointers up to date, we can thus determine which bucket we need to move an index to (or if an index needs to be removed) in constant time, making these operations cost a total of $O(\log(k))$ purely to update the values in B.

Using this same intuition, we can conclude thus that Subroutine 5 is an $O(t) + O(k \log k)$ operation: We traverse the t buckets and count how many unassigned buckets there are. Observe that for indices S_i, H_i, or L_i, we may only move these indices to an earlier round. Thus, we can guarantee that we only need to move them (costing $O(\log k)$) at most once per execution of Subroutine 5. For P_i and L_i, we can efficiently push back our move such that it is only performed once per execution of Subroutine 5: P_i never affects the way the $Count[i]$ value is calculated, so we can simply store where we need to move P_i and only perform it at the end of the execution of the subroutine. As for L_i, while it does affect $Count[i]$, we know that whenever we do move this index, we move it on Line 18 to the first unassigned round that appears after our current R_j. We also know that we will reset our values on Line 20 immediately after finishing all moves for this j. Observe that since Line 8 verifies that $UnassignedRoundsCount > 0$ and by the way that $Count[i]$ is calculated, we are guaranteed that for any C_i, if we added 1 to $Count[i]$ because of L_i during this particular instance when Line 8 was True (that is, for this particular set of round $\{R_{LeftIndex}, \dots, R_j\}$), that we must again add 1 to $Count[i]$ on the next instance when Line 8 is True (that is, for the next set of rounds $\{R_{j+1}, \dots, R_{j'}\}$ for some $j' > j$ where Line 8 is True). Thus, we can simply add 1 to $Count[i]$ immediately, and avoid having to move L_i immediately, and instead locally remember that this particular L_i is a value that is supposed to be within the set $\{R_{j+1}, \dots, R_{j'}\}$. We can keep repeating this until the end of this subroutine execution, at which point we can perform our single move. Thus, this index also costs only $O(\log k)$ to move during the entire execution of Subroutine 5.

For the rest of this subroutine, we have $O(1)$ operations in the for-loop of Line 5, and a few initialisation and reset operations on Lines 1, 2, and 20. Since these are done in the worst case on every round, we end up with a $O(tk) + O(k \log k)$ subroutine.

All rules of Definition 5 can be checked in constant time, as is finding the new values for each index, since by previous arguments we can merely obtain pointers to these earliest or latest unassigned rounds that appear at or before some specific round. The actual movement of each index will take $O(\log k)$, as before. Since we may check these for all cycles, we thus find that we can apply them in $O(k \log k)$ time.

Subroutine 6, in a similar but less complicated argument to the one for Subroutine 5, can be executed (ignoring the assignment operation) in $O(t) + O(k \log k)$, since no reset operation takes place. Additionally, It performs a single round assignment which will require $O(t)$ operations to update all pointers of the other buckets, and $O(\log k)$ operations to remove the indices of the assigned cycle.

All rules of Definition 6 can be checked in constant time, and their assignments, index removals, and subsequent index moves of Lines 15 and Lines 17 can be done once again in $O(t) + O(k \log k)$ time.

The fallback rule we fall back to in case no index movement or assignment was made (which can be checked in $O(k)$ time) can be executed $O(t) + O(k \log k)$ time as well.

Thus, we are guaranteed to always make an assignment for every iteration of the while loop of Line 4. All other operations are simply constant time, or simple assignments which cost less than the execution of the while loop. Thus, provided we don't return *False*, our Heuristic 7 can execute in $O(k^2 \log k) + O(k^2 t)$. Pairing it with the binary-search on t of Algorithm 3, we get an $O(k^2 \log k \log t) + O(k^2 t \log t)$ heuristic.

Chapter 5

Simulation results

5.1 Preliminaries

We describe here the simulation results obtained from the comparisons of all algorithms when broadcasting. For Algorithms 1,4,7, and BroadcastBucket*[1], these were all run using the wrapper from Algorithm 3. We first wanted to compare the performance of our algorithms with the algorithms from the literature. Recall from Chapter 2 that approximation-algorithm S_{cycle} finds the optimal values on particular subfamilies of flower graphs. This spurs the following question: Where do our algorithms and heuristics succeed in finding an optimal broadcast scheme? Where do they fail? Which algorithms perform better than others, and under which conditions? What properties of particular subfamilies of flower graphs are difficult, and can this intuition lead to further development towards a better (and perhaps optimal) broadcast scheme, or give an intuition for a possible proof that the broadcast problem on flower graphs is *NP*-Hard?

For simplicity, we focused on two particular properties of flower graphs, and sought to measure the performance of our algorithms as these changed. In particular, we focus on the *cycle lengths*, and the *quantity of cycles*. All simulations were made in Python. For the random generation of numbers to generate the graph instances, we used NumPy [16].

5.2 Data

We first compared our algorithms on instances where the number of cycles stayed the same, but the size of their cycles slowly increased. To attempt to capture a relatively diverse profile of graphs, we uniquely randomly generated 1000 flower graphs of 5 cycles for each range of cycle lengths that we defined, and measured the mean, maximum, and minimum values of the number of rounds used by the broadcast schemes

[1] recall *BroadcastBucket** from Section 3.3

generated by each algorithm. We compared this value to the lower bounds provided in Lemma 6.

5.2.1 Comparisons of algorithm performances as cycle lengths increase, against lower bounds

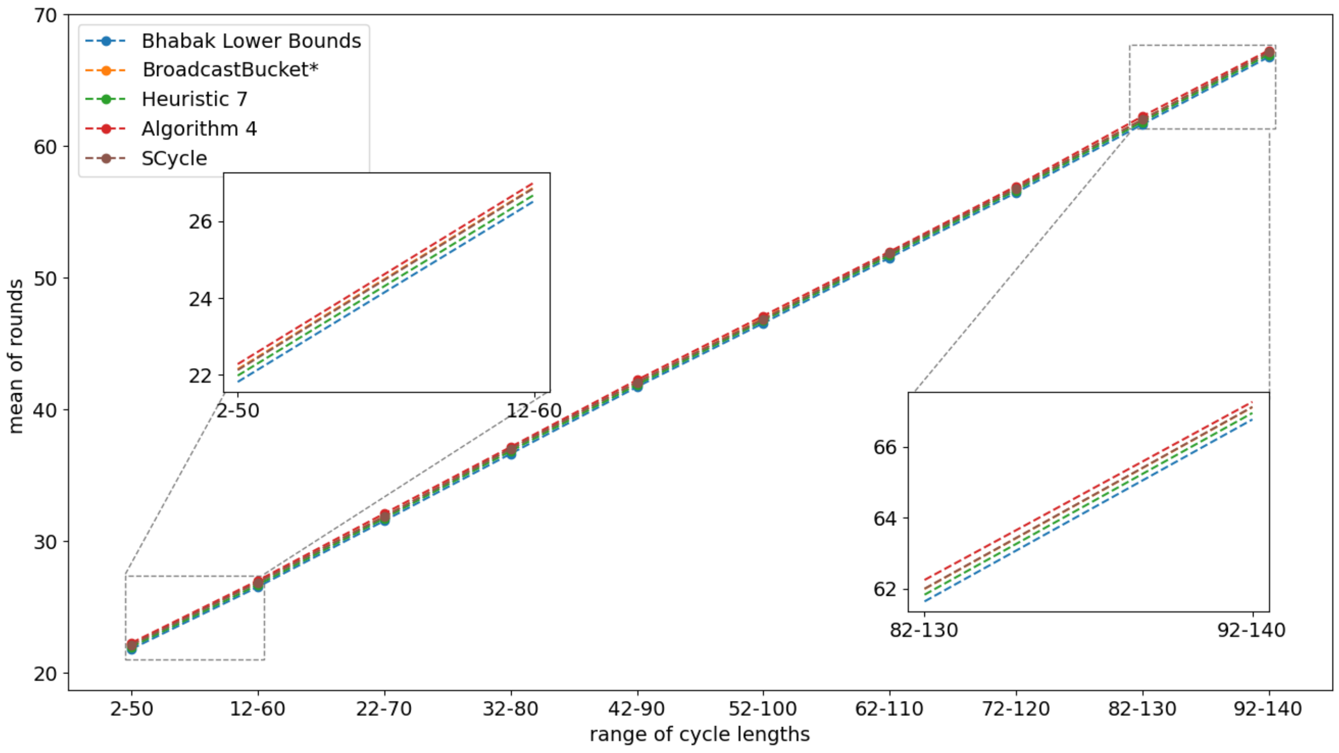


Figure 5.1: Mean of rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested. Algorithm 1 is not included, as its mean lies much further higher than the means of the other algorithms.

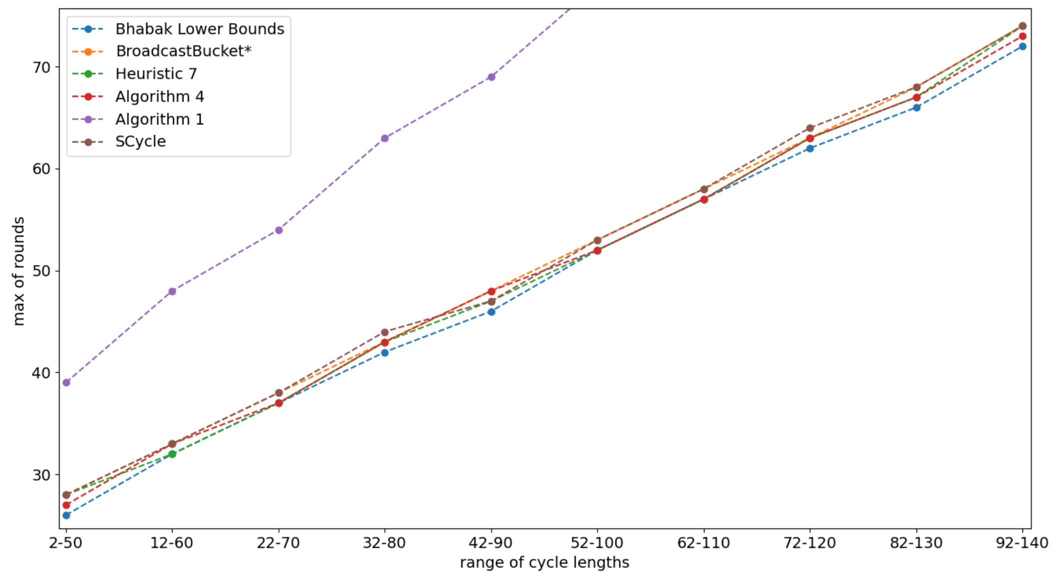


Figure 5.2: Max rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

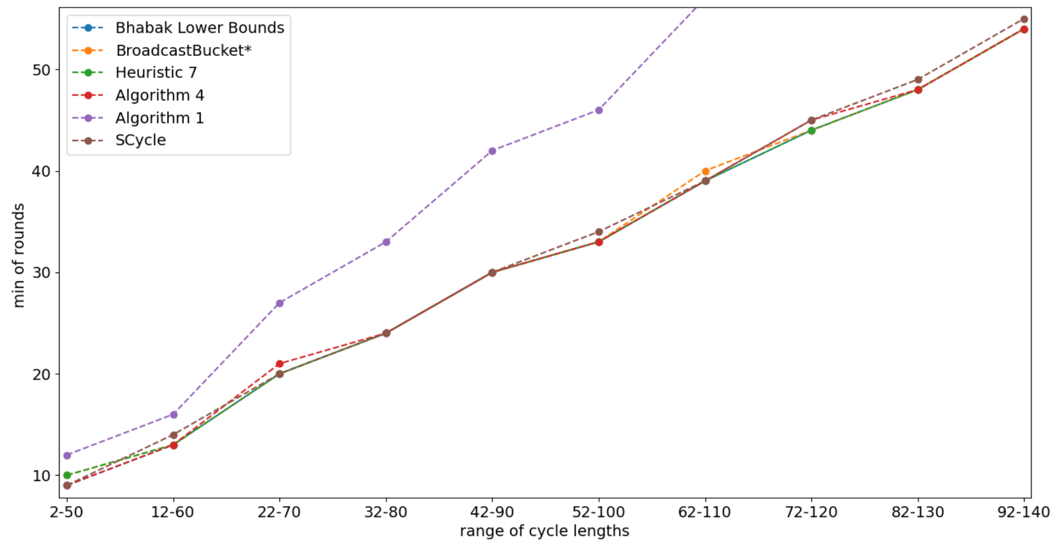


Figure 5.3: Min rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

It becomes interesting to compare the results of other algorithms with respect to the results of Heuristic 4. For this reason, for each instance tested, we subtracted the number of rounds they used from the number of rounds used by the broadcast scheme generated by Heuristic 7. We highlight that the maximum and minimum values plotted here represent the maximum number of rounds required for *any* instance within its respective set of 1000 instances. Thus, even if two algorithms obtained the same maximum or minimum values, it is entirely possible that these values were obtained on two completely difference instances: one cannot thus conclude that these two algorithms can be argued to perform in a similar manner.

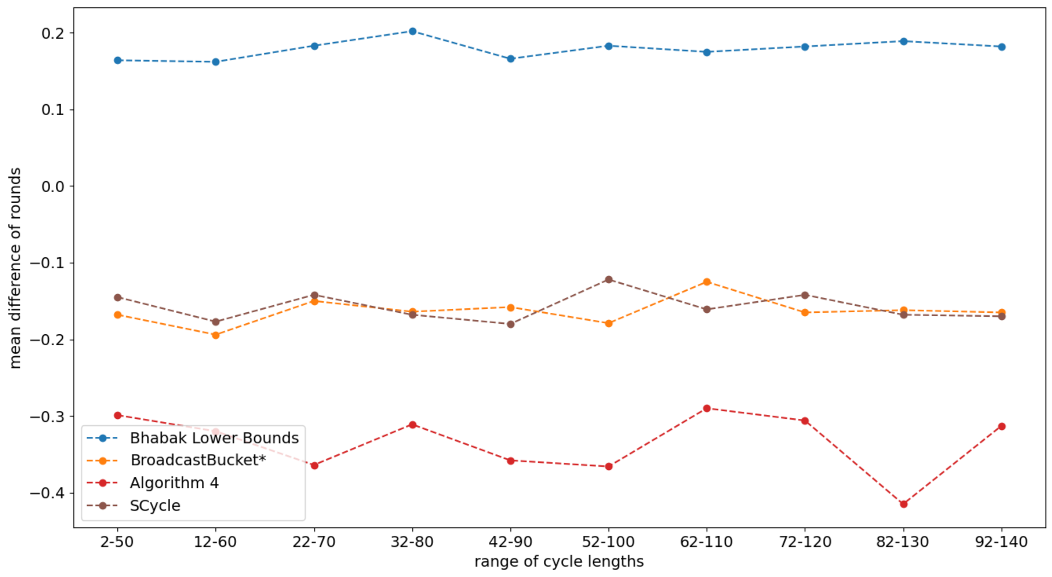


Figure 5.4: Mean difference obtained by substracting the number of rounds used by the broadcast schemes generated for each algorithm from the broadcast scheme generated by Heuristic 7. Algorithm 1 is not included as its mean difference lies much further down than the means of the other algorithms.

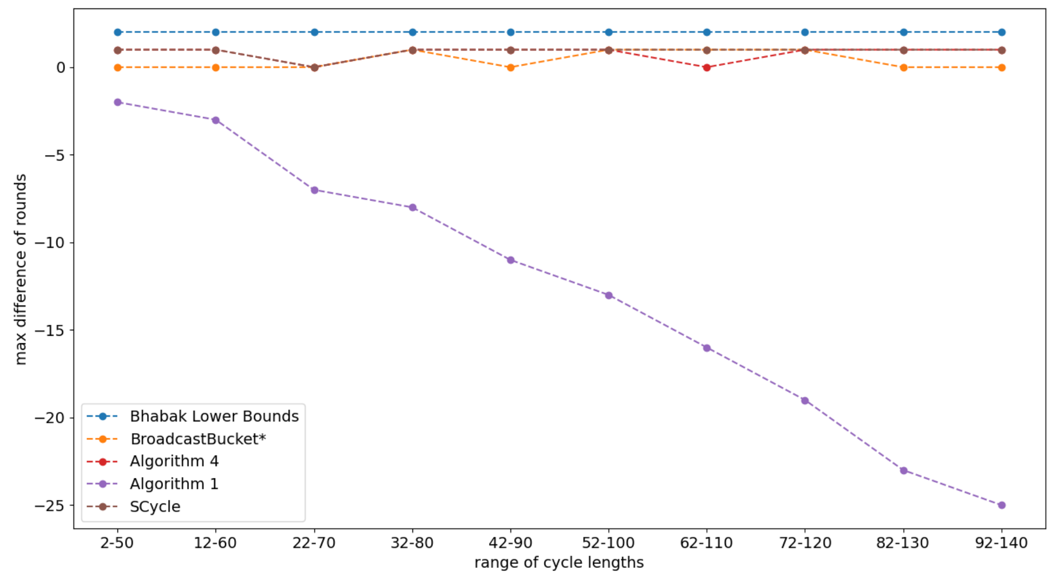


Figure 5.5: Max difference obtained by substracting the number of rounds used by the broadcast schemes generated for each algorithm from the broadcast scheme generated by Heuristic 7.

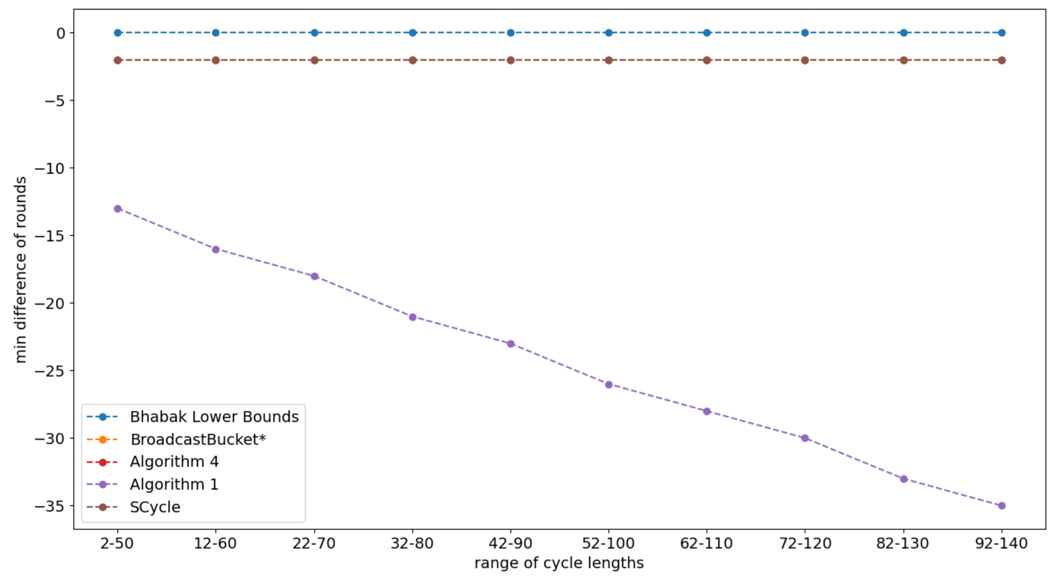


Figure 5.6: Min difference obtained by substracting the number of rounds used by the broadcast schemes generated for each algorithm from the broadcast scheme generated by Heuristic 7.

5.2.2 Comparisons of algorithm performances as cycle quantity increase, against lower bounds

We next compared our algorithms on instances where the number of cycles stayed the same, but the size of their cycles slowly increased from 5 cycles to 10 cycles. Again, we uniquely randomly generated 1000 flower graphs using cycle lengths $2-50$, and slowly increased the number of cycles used.

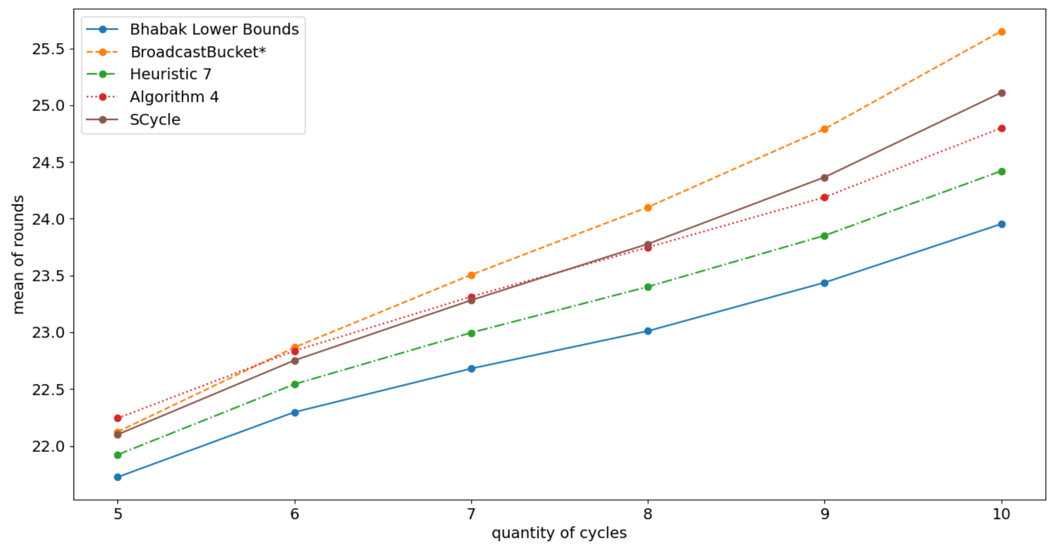


Figure 5.7: Mean of rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested. Algorithm 1 is not included, as its mean lies much further higher than the means of the other algorithms.

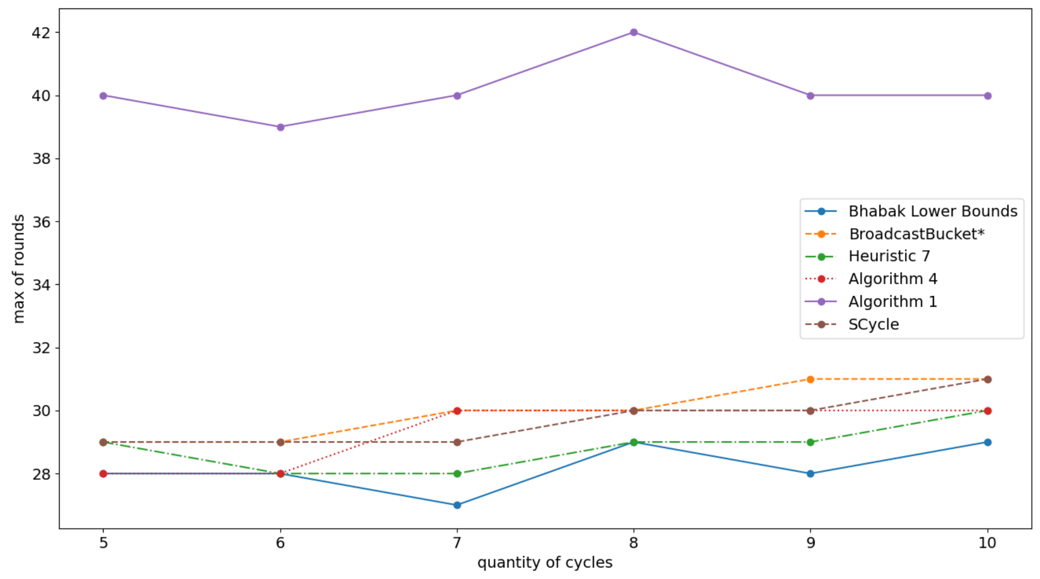


Figure 5.8: Max rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

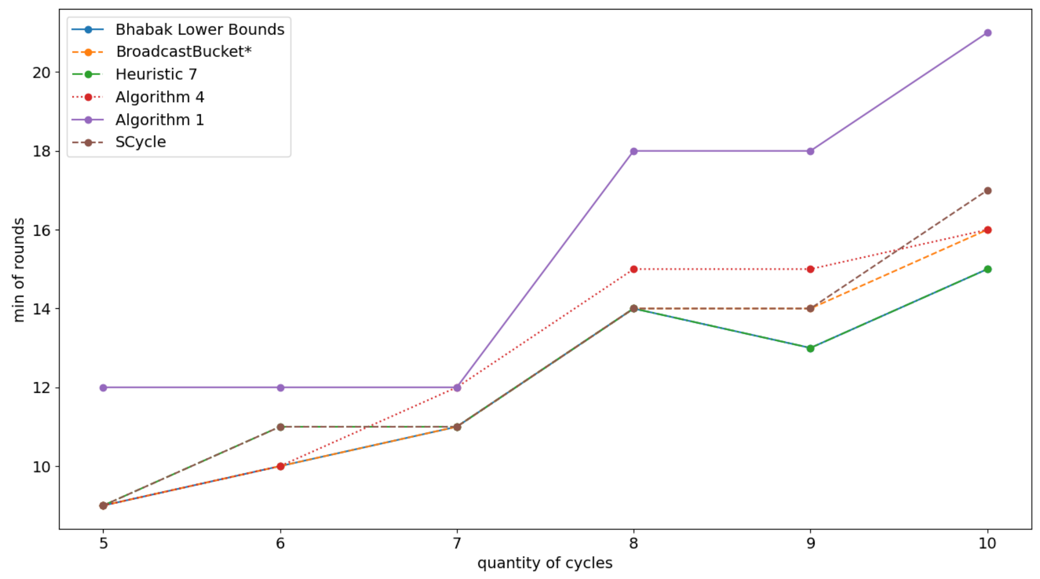


Figure 5.9: Min rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

Again, we compared the results of other algorithms with respect to the results of Heuristic 4, using the same procedure as in the previous section.

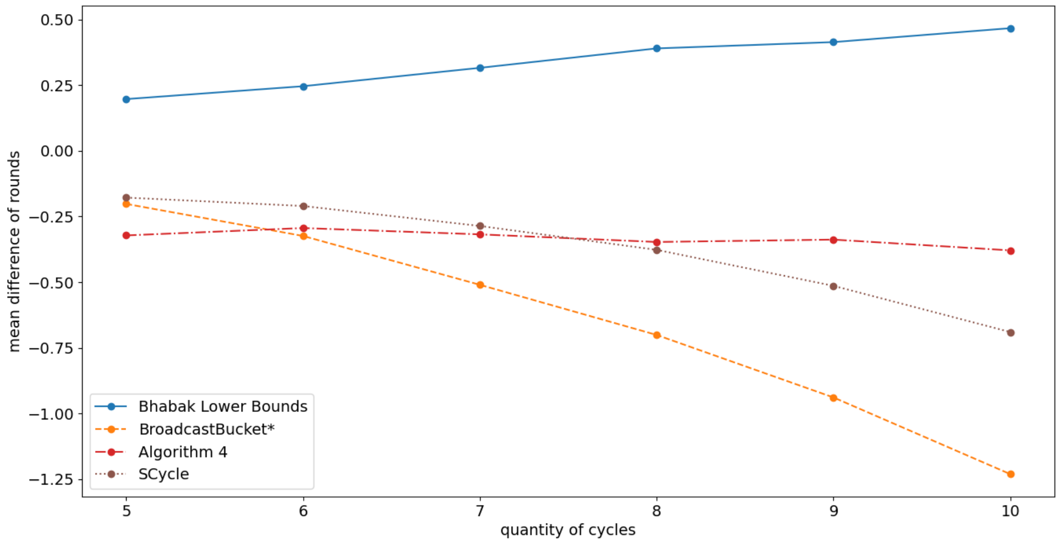


Figure 5.10: Mean difference obtained by substracting the number of rounds used by the broadcast schemes generated for each algorithm from the broadcast scheme generated by Heuristic 7. Algorithm 1 is not included as its mean difference lies much further down than the means of the other algorithms.

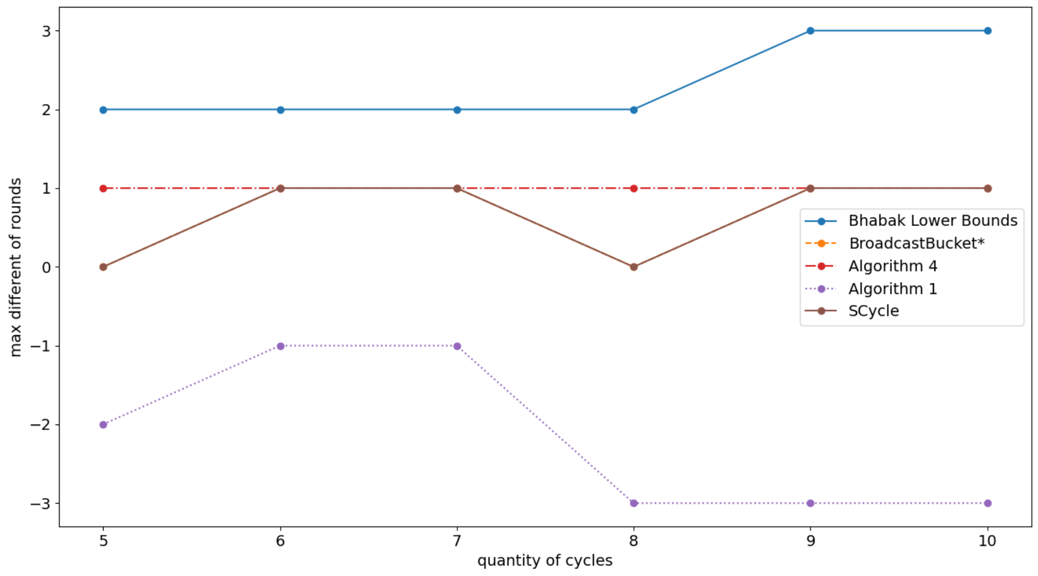


Figure 5.11: Max difference obtained by substracting the number of rounds used by the broadcast schemes generated for each algorithm from the broadcast scheme generated by Heuristic 7.

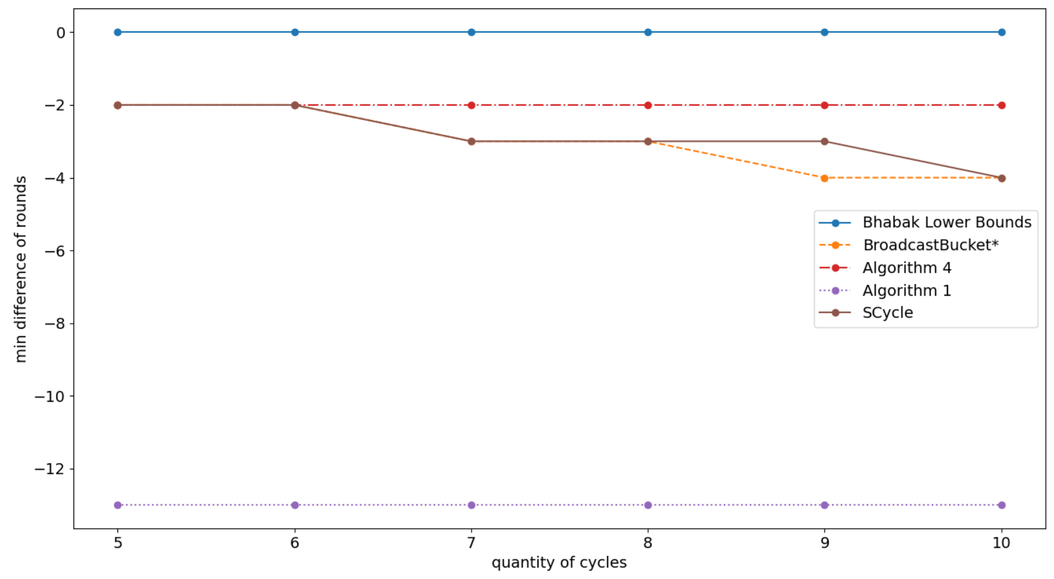


Figure 5.12: Min difference obtained by substracting the number of rounds used by the broadcast schemes generated for each algorithm from the broadcast scheme generated by Heuristic 7.

5.2.3 Comparisons of algorithm performances as cycle lengths increase, against optimal

While this is useful, we don't know how well the lower bounds fair compared to the true optimal schemes. For smaller instances of flower graphs, we can actually brute-force our way to finding the optimal solution: Observe that every broadcast scheme within a flower graph can actually be seen as a *spider graph*: a tree with a central vertex and paths of varying lengths all attached to this central vertex by one of their end vertices. To obtain the optimal broadcast time, one can thus generate every possible spider graph obtainable by cutting one edge from every cycle, and broadcast in each spider graph from u. The spider graphs with the smallest broadcast times can then be used as an optimal broadcast scheme for that flower graph. Of course, this method becomes especially inefficient when there are cycles of the same size as isomorphic spider graphs will be generated multiple times, but can be quickly alleviated by considering the combinations of paths possible, regardless of which cycle it happens to be created on. We were thus able to find the optimal solution for a large number of fairly small instances.

Using these instances, we can likewise check how close to $\lfloor 1.5t_{opt} \rfloor$ the solutions found by algorithms 1 and 4 are. In fact, we found that Algorithm 4 seems to perform even below a factor of $\lfloor 1.25t_{opt} \rfloor$, which we include in our plots for comparison.

As before, all of the subsequent simulations are performed on 1000 randomly generated instances, this time with smaller cycles of lengths $2-20$ (as larger cycles take too long to bruteforce through).

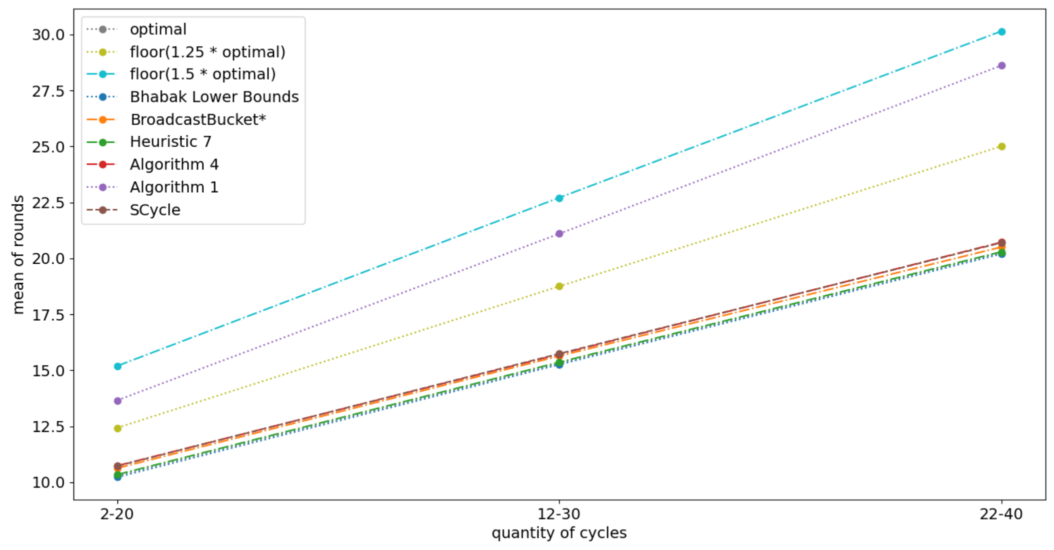


Figure 5.13: Mean of rounds used by the broadcast schemes generated for each algorithm tested.

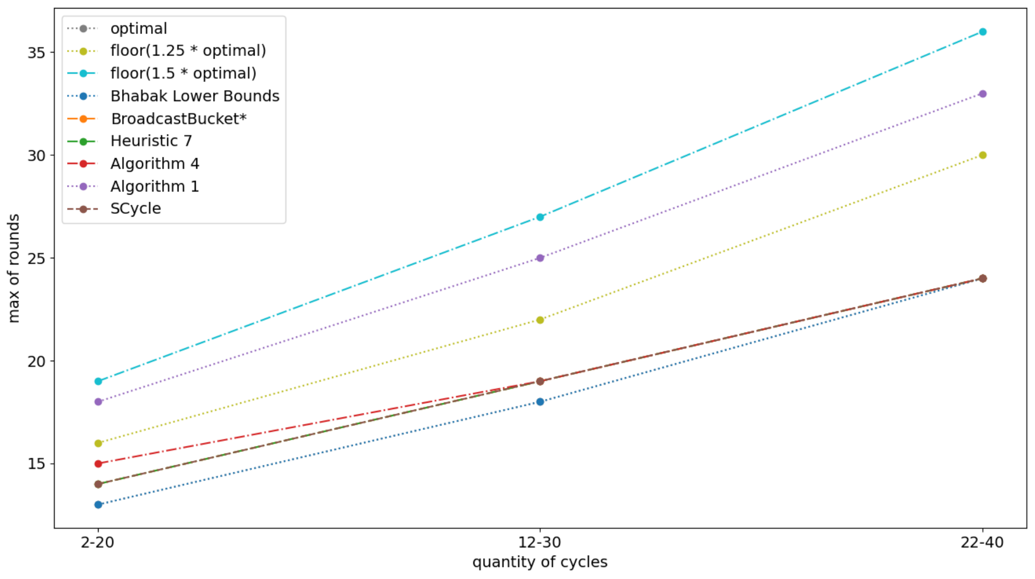


Figure 5.14: Max rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

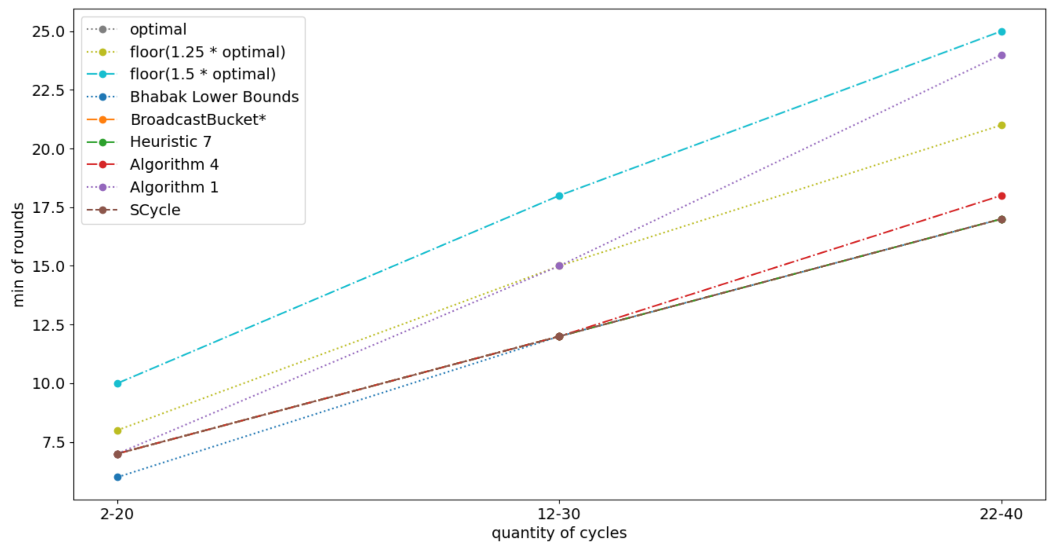


Figure 5.15: Min rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

Since the previous section made comparisons to Heuristic 7 with a larger set of instances, we don't do this again here against the optimal values.

5.2.4 Comparisons of algorithm performances as cycle quantity increase, against optimal

As before, all of the subsequent simulations are performed on 1000 randomly generated instances, with particular parameters in mind.

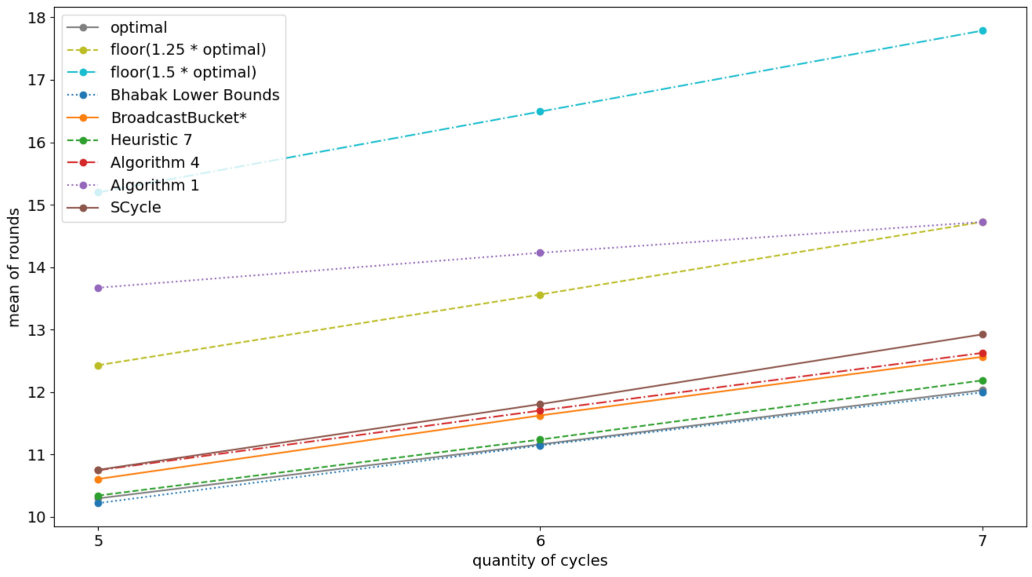


Figure 5.16: Mean of rounds used by the broadcast schemes generated for each algorithm tested.

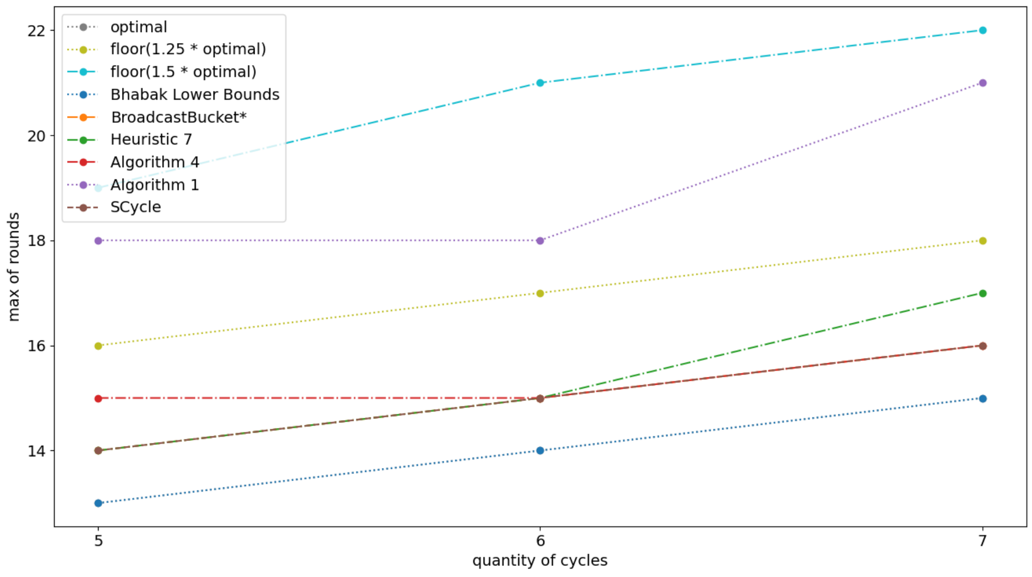


Figure 5.17: Max rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

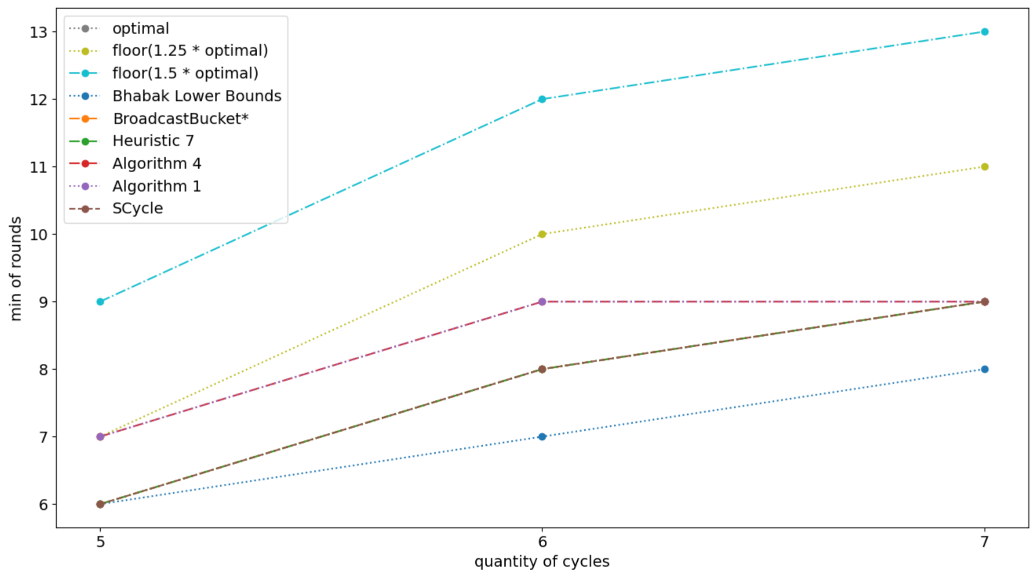


Figure 5.18: Min rounds used by the broadcast schemes generated for each algorithm, where each datapoint represents 1000 randomly-generated instances tested.

5.3 Discussion

From the data above, we can conclude that Heuristic 7 is a well-performing heuristic: On small instances it consistently finds, on average as per Figures 5.1 and 5.7, the most efficient broadcast schemes out of all other algorithms. Additionally, this seems to stay true across a wide array of instances, even as we increase our quantity of cycles or increase cycle lengths.

We can notice that as the cycle lengths are increased, all algorithms perform consistently, and judging by Figure 5.2 and Figure 5.3, that there does not seem to be particular unexpected instances on which any algorithm performs noticeably worse or noticeably better than in average, since all algorithms have their maximum and minimums stay fairly linearly related to increase in cycle length.

It is particularly interesting to note instead that as the number of cycles increase yet remain with the same range for cycle lengths, that Algorithms S_{cycle} and $BroadcastBucket^*$ both seem to get increasingly worse performances, unlike Heuristic 7, and difference can be especially seen in Figure 5.10. Yet, the performance of Algorithm 4 stays linear: thus, we can conclude that the issues lies not in the choice of always assigning the first call to the largest cycle, as Algorithm 4 makes this decision as well. It may be of interest to investigate exactly *which* decisions penalize S_{cycle} and $BroadcastBucket^*$ so strongly. It would be interesting to see if this performance drop generalises to *any* instance that can be qualified to have a relatively large quantity of cycles compared to their lengths.

Judging by Figure 5.9 and Figure 5.9, one can again conclude that the algorithms seem to stay within a similar range of performance, growning linearly with the increase in cycle length.

One can see from the comparisons done against Heuristic 7, particularly in Figure 5.5 and 5.11, that it is occasionally outclassed by Algorithm 4, S_{cycle}, and Algorithm *BroadcastBucket**, and will find a broadcast scheme that is slightly slower than the ones generated by these algorithms. Since we remain on fairly small instances, we cannot claim whether or not this difference remains small or worsens for particular subfamilies of flower graphs. Despite this, we can presume the possibility of a small, but easily fixable mistake in the design of the heuristic. It would be worth it to identify which decisions penalise our heuristic, and particularly whether this can be associated with the decisions made when using our default rule of Definition 7, (that is, if the broadcast schemes generated by Algorithm 4, S_{cycle}, or Algorithm *BroadcastBucket** also make the same assignments as Heuristic 7 when Heuristic 7 makes an assignment using one of its other rules, and if the main differences in assignments always concern assignments made by the default rule), which would hint towards a poorly-designed default rule.

Regarding the comparisons to the optimal values, we can notice a number of things: First, we can notice that the lower bounds from 6 seem to be in practice very good at estimating the true optimal values, and seem to be a good lower bound to test by. Secondly, with respect to approximation Algorithms 1 and 4, we find that the latter performs in a satisfactory manner, especially given its efficient running time compared to other algorithms, as well as its guarantee of a (1.5)-approximation ratio. In practice, we found that it occasionally hit, but never crossed, a (1.25)-approximation ratio.

In practice, we found that Algorithm 1 seemed to hit its $\lfloor 1.5t_{opt} \rfloor$ bound often. Instead, we found that Algorithm 4 never crossed a $\lfloor 1.25t_{opt} \rfloor$ bound, which would imply that a better approximation ratio may be possible, however further investigation needs to be made.

One particular weakness of our simulations is the size of the instances: it unfortunately becomes significantly expensive to run Heuristic 7 on larger instances, and thus becomes difficult to test on thousands of instances of large sizes. It remains to be seen whether or not the performance of that heuristic, or any of the other algorithms fall off drastically on signifcantly larger instances. Additionally, we only compare performances as the range of cycle lengths increase, or as the quantity of cycles increase. Additional simulations would greatly aid us in knowing how well Heuristic 7 performs with respect to the other algorithms.

Chapter 6

Discussion and Future Work

In this thesis, we reviewed broadcasting in a number of graph families: trees, then unicyclic graphs, then particular subfamilies of cactus graphs, and finally to Flower graphs. We particularly focused on Flower graphs, a graph family of high interest due to its relative simplicity, yet difficulty when it comes to finding an optimal broadcasting algorithm. Flower graphs seem to capture the difficulty present in Cactus graphs, and thus are a key to furthering broadcasting strategies in Cactus graphs and numerous other families that share similar properties as cactus graphs. We developed Algorithm 1, an algorithm which can be wrapped with a binary-search (given by Algorithm 3) on the value of $b(u,G)$ to obtain a (1.5)-approximation algorithm. This improves the current approximation ratio on Flower Graphs from 2, previously obtained by S_{cycle} in [5]. We then identified improvements which could be made into a heuristic ImprovedBroadcastBucket, which we compared to current heuristics in the literature, and showed that our heuristic often found significantly faster broadcast schemes than the best-performing algorithms in the literature.

Crucially, the question of whether or not the Broadcast Problem is NP-Hard on flower graphs (as well as cactus graphs) remains unsolved. We currently conclude that finding an optimal solution on flower graphs is a difficult problem, and thus it would be worthwhile to explore the possibility of proving this problem to be NP-Hard.

With respect to approximation on flower graphs, we have found that in practice the algorithm ImprovedApprox, an improvisation on our (1.5)-approximation algorithm, tends to obtain broadcast schemes on a (1.25)-approximation ratio. Thus, our next step would be to find out if this algorithm can indeed achieve a (1.25)-approximation ratio, as it remains a highly efficient algorithm that in practice finds reasonable non-optimal broadcast schemes.

As for our heuristic ImprovedBroadcastBucket, it would be interesting to exactly identify which subcases of

Flower graphs are optimally solved by ImprovedBroadcastBucket, and for which subcases ImprovedBroadcastBucket instead needs some additional rounds to completely inform the graph. This could perhaps lead to further intuition into the difficulty of Flower Graphs, as well as possible directions for improvements of the heuristic.

Then, we can seek to extend the uses of our algorithms in other graph families. Obviously, developing approximation algorithms as well as heuristics for finding the broadcast scheme on cactus graphs is worthwhile, but other families are also highly related: Notably, solutions found on flower graphs have proven to be useful for finding solutions on k-path graphs.

Bibliography

[1] G. R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*, volume 11. Addison-Wesley, Reading, Mass, 2000.

[2] B. Ben-Moshe, B. Bhattacharya, and Q. Shi. Efficient Algorithms for the Weighted 2-Center Problem in a Cactus Graph. In *Algorithms and Computation*, volume 3827, pages 693–703. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[3] J.-C. Bermond, H. A. Harutyunyan, A. L. Liestman, and S. Perennes. A Note on the Dimensionality of Modified Knödel Graphs. *International Journal of Foundations of Computer Science*, 08(02):109–116, June 1997.

[4] P. Bhabak. *Approximation Algorithms for Broadcasting in Simple Graphs with Intersecting Cycles*. PhD thesis, Concordia University, 2014.

[5] P. Bhabak and H. A. Harutyunyan. Constant Approximation for Broadcasting in k-cycle Graph. In *Algorithms and Discrete Applied Mathematics*, volume 8959, pages 21–32. Springer International Publishing, Cham, 2015.

[6] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, 1999.

[7] M. Čevnik and J. Žerovnik. Broadcasting on cactus graphs. *Journal of Combinatorial Optimization*, 33(1):292–316, Jan. 2017.

[8] M. Chen, S. Mao, and Y. Liu. Big Data: A Survey. *Mobile Networks and Applications*, 19(2):171–209, Apr. 2014.

[9] N. Conlan. *Heuristic Algorithms for Broadcasting in Cactus Graphs*. PhD thesis, Concordia University, 2017.

[10] N. Conlan, H. A. Harutyunyan, and E. Maraachlian. Heuristic Algorithms with Near Optimal Broadcasting in Cactus Graphs. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 253–257, Västerås, Sweden, Mar. 2020. IEEE.

[11] A. Farley, S. Hedetniemi, S. Mitchell, and A. Proskurowski. Minimum broadcast graphs. *Discrete Mathematics*, 25(2):189–193, Feb. 1979.

[12] A. M. Farley. Minimal broadcast networks. *Networks*, 9(4):313–332, 24.

[13] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematics*, 53(1-3):79–133, Sept. 1994.

[14] H. Grigoryan. *Problems Related to Broadcasting in Graphs*. PhD thesis, Concordia University, 2013.

[15] H. Grigoryan and H. A. Harutyunyan. New Lower Bounds on Broadcast Function. In *Algorithmic Aspects in Information and Management*, volume 8546, pages 174–184. Springer International Publishing, Cham, 2014.

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[17] H. Harutyunyan, G. Laza, and E. Maraachlian. Broadcasting in necklace graphs. In *Proceedings of the 2009 C3S2E Conference on - C3S2E '09*, page 253, Montreal, Quebec, Canada, 2009. ACM Press.

[18] H. Harutyunyan and E. Maraachlian. Linear Algorithm for Broadcasting in Unicyclic Graphs: (Extended Abstract). In *Computing and Combinatorics*, volume 4598, pages 372–382. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[19] H. Harutyunyan and E. Maraachlian. Linear Algorithm for Broadcasting in Networks With No Intersecting Cycles. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2009, Las Vegas, Nevada, USA, July 13-17, 2009, 2 Volumes*, pages 296–301, Oct. 2009.

[20] H. A. Harutyunyan and A. L. Liestman. Upper bounds on the broadcast function using minimum dominating sets. *Discrete Mathematics*, 312(20):2992–2996, Oct. 2012.

[21] S. Hedetniemi, R. Laskar, and J. Pfaff. A linear algorithm for finding a minimum dominating set in a cactus. *Discrete Applied Mathematics*, 13(2-3):287–292, Mar. 1986.

[22] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 24.

[23] M. Hilbert and P. Lopez. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025):60–65, Apr. 2011.

[24] J. Hromkovič, R. Klasing, A. Pelc, P. Ružička, and W. Unger. *Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance*. Springer, Berlin ; New York, 2005.

[25] K. Husimi. Note on Mayers' Theory of Cluster Integrals. *The Journal of Chemical Physics*, 18(5):682–684, May 1950.

[26] R. Labahn. A minimum broadcast graph on 63 vertices. *Discrete Applied Mathematics*, 53(1-3):247–250, 1994.

[27] T. Leighton and A. Moitra. Some results on greedy embeddings in metric spaces. *Discrete & Computational Geometry*, 44(3):686–705, 2010.

[28] N. A. Lynch. *Distributed Algorithms*. Elsevier, 1996.

[29] M. Maheo and J.-F. Saclé. Some minimum broadcast graphs. *Discrete Applied Mathematics*, 53(1-3):275–285, 1994.

[30] E. Maraachlian. *Optimal Broadcasting in Treelike Graphs*. PhD thesis, Concordia University, 2010.

[31] M. Markov, M. Ionut Andreica, K. Manev, and N. Tapus. A linear time algorithm for computing longest paths in cactus graphs. *Serdica Journal of Computing*, 6(3):287p–298p, 2012.

[32] B. Paten, M. Diekhans, D. Earl, J. S. John, J. Ma, B. Suh, and D. Haussler. Cactus Graphs for Genome Comparisons. *Journal of Computational Biology*, 18(3):469–481, Mar. 2011.

[33] B. Paten, D. Earl, N. Nguyen, M. Diekhans, D. Zerbino, and D. Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome Research*, 21(9):1512–1528, Sept. 2011.

[34] R. J. Riddell Jr. *Contributions to the Theory of Condensation*. University of Michigan, 1951.

[35] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison Wesley, Reading, Mass, 1998.

[36] Scheuermann and Wu. Heuristic Algorithms for Broadcasting in Point-to-Point Computer Networks. *IEEE Transactions on Computers*, C-33(9):804–811, Sept. 1984.

[37] P. Scheuermann and M. Edelberg. Optimal broadcasting in point-to-point computer networks. *Dep. Elec. Eng. Comput. Sci., Northwestern Univ., Evanston, IL, Tech. Rep*, 1981.

[38] C. Schindelhauer. On the Inapproximability of Broadcasting Time (Extended Abstract). page 12.

[39] B. Shao. *On K-Broadcasting in Graphs*. PhD thesis, Concordia University, 2006.

[40] P. J. Slater, E. J. Cockayne, and S. T. Hedetniemi. Information Dissemination in Trees. *SIAM Journal on Computing*, 10(4):692–701, Nov. 1981.

[41] A. S. Tanenbaum, D. Wetherall, and N. Feamster. *Computer Networks*. Pearson, Harlow, United Kingdom, sixth edition, global edition edition, 2021.

[42] G. E. Uhlenbeck and G. W. Ford. *Lectures in Statistical Mechanics*. Number 1 in Lectures in Applied Mathematics. American Mathematical Soc, Providence, RI, 4. printing edition, 1974.

[43] B. Zmazek and J. Žerovnik. The obnoxious center problem on weighted cactus graphs. *Discrete Applied Mathematics*, 136(2-3):377–386, Feb. 2004.

[44] B. Zmazek and J. Zerovnik. Estimating the Traffic on Weighted Cactus Networks in Linear Time. In *Ninth International Conference on Information Visualisation (IV'05)*, pages 536–541, London, England, 2005. IEEE.