VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Frankfurt University Of Applied Science
Faculty 2 : Computer Science and Engineering

A Combinatorial Approach To Divisorial Gonality

Full name : Nguyen The Viet

Matriculation number : 1191950 (9990)
Supervisor : Dr. Huong Tran Thi Thu
Co-supervisor : Prof. Manuel Clavel

BACHELOR THESIS

Submitted in partial fulfillment of the requirements
for the degree of Bachelor Engineering
In study program Computer Science, Vietnamese - German University, 2019

Binh Duong, Vietnam

Abstract

Given a graph G with a divisor that represents a configuration in chip-firing game on G, the gonality of G is the lowest degree of the divisor that reaches all vertices of G [2]. The thesis presents calculations for the gonality of special graphs like *fan graphs* and *flower graphs* using the *reachability* of a divisor. Additionally, an upper bound for the gonality of *tree with cycles* graphs is given.

Finally, we use the *reduction rules* in [2] to characterize tree wich cycles graphs of gonality 2. By that, the order of applying reduction rules on tree with cycles is determined and checking the compatibility on cycles is much more simple. Algorithms to move chips on tree and cycle are shown.

Acknowledgement

I would like to thank both of my supervisors, Dr. Tran Thi Thu Huong and Dr. Manuel Clavel for helping me complete the thesis. I also would like to express my deep appreciation for Dr. Tran Thi Thu Huong for her continuous support in giving orientation and assisting me in understanding more deeply about the topic during my thesis phase.

Finally, I would like to thank Nguyen Phuoc Bao Hoang, who finished his thesis last year on a topic of *Chip-firing game*, for assisting me in utilizing tools to write the thesis.

Contents

List of Figures

Chapter 1

Introduction

Chip-firing game was introduced by Spencer based on the results of research on *balancing game* in 1980s [12]. The game can be played on a graph by assigning an integer number (chips) to each vertex in the graph [1]. Such assignment is called a configuration or a divisor. The degree of a divisor is its total number of chips. A move is done by selecting a vertex and moving chips to each of its neighbors. A divisor is said to *reach* a vertex v if after a sequence of moves, we obtain a new divisor that has at least a chip at v and a non-negative number of chips on other vertices. Gonality of a graph is the minimum degree of a divisor on that graph which reaches all vertices. The concept of gonality is introduced by [2].

It is proved in [5] that the problem of computing the gonality of a general graph problem is NP-hard. For that reason, there are some efforts to find upper bound and lower bound for the gonality. For instance, a lower bound can be constructed by using *bramble*, which was presented in [4]. In addition, the *reachability* to a vertex of a divisor is an useful way for finding its upper bound [2]. In the thesis, we use this *reachability* property to construct an upper bound for the gonality on *tree with cycles* graph as well as to find the gonality of some graph families like *fan graphs*, *flower graphs* and their extensions.

Besides, Bodewes provided a set of *reduction rules* [2], which allows to determine a graph of gonality at most 2 by checking if that graph is reducible to the empty graph using reduction rules or not. The specific orders for applying these rules in a general graph are not stated explicitly, and it is not very simple to check the *compatibility* of the constraint set on a cycle. In the thesis, we enhance the rules applied for *tree with cycles* graphs. By that, checking the compatibility of pairs of constraints can be done more simply by considering *appended vertices*. Furthermore, it is noteworthy that these rules can be utilized to characterize *tree with cycles* graphs of gonality 2. We also conduct an algorithm to apply *reduction rules* in a specific order to reduce these graphs to the empty graph.

The remaining parts of the thesis are divided into four chapters. Preliminary definitions of graph theory and chip-firing game on graph are presented in chapter 2. Some graph families like *fan graphs* and *flower graphs* are defined in this chapter. We present the concept of gonality and its relevant known results in chapter 3. Chapter 4 is our contribution in this thesis. We calculate the gonality of *fan graphs, flower graphs* and their extensions using techniques in Chapter 3. A characterization for *tree with cycles* graphs of gonality 2 using *reduction rules* of Bodewes [2] is given. Finally, pseudo-codes for algorithms of firing chips in tree and cycle as well as an algorithm for checking the gonality of *tree with cycles* graphs are presented in the Appendix.

Chapter 2

Background On Graph Theory & Chip Firing Game

In this chapter basic definitions of graph theory and chip-firing game concepts on graph are given.

2.1 Preliminaries On Graph Theory

We define basic definitions of graphs including Laplacian Matrix and some familiar graphs in this section.

2.1.1 Basic Definitions

Definition 2.1. (Graph). A graph is a 2-tuple $G = (V, E)$ where V is the set of vertices (nodes), and E is the set of edges which are formed by un-ordered pairs of vertices in V. Formally, we denote $E \subseteq \{vw|v, w \in V\}$. Two vertices that connect via an edge are *adjacent* to each other. The vertices are said to be *incident* to the edge. Let $E(v)$ be the set of edges that are incident to $v \in V$, and $E(v) = \{vw \in E|w \in V\}$.

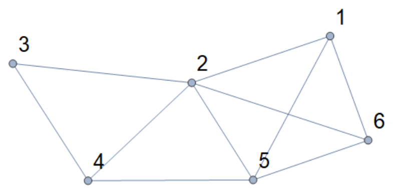
Definition 2.2. The set of neighbors of a vertex v consists of the vertices that are adjacent to v, which is denoted as $N(v) = \{w \in V|vw \in E\}$. Degree of vertex v is the number of its neighbors, or the number of edges incident to v. We denote the degree of vertex v as $deg(v) = |N(v)| = |E(v)|$.

Definition 2.3. A sub-graph is a graph whose vertices and edges belong to the subsets of another graph. A graph $H = (F, U)$ is called sub-graph of $G = (V, E)$ if and only if $F \subseteq V$ and $U \subseteq E$. An induced sub-graph $H' = (F', U')$ of G is a sub-graph such that for each un-ordered pair $v, w \in F'$, $vw \in U'$ if and only if $vw \in E$. In other words, U' contains all possible edges that exist in the original graph.

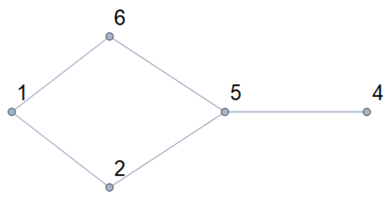
Definition 2.4. Given a graph $G = (V, E)$, a *path* of length k is a sequence $(v_0, v_1, ..., v_k) \in V$ with $v_i, v_{i-1} \in E(G)$ for $i \in \{1, 2, ..., k\}$. We denote $P_{u,v}$ as the subset of vertices in the path from u to v in that order. A *cycle* is a path that begins and ends with the same vertex. This means repetition only occurs for the first and the last nodes of the path. A graph that contains no cycle is called an *acyclic* graph.

Definition 2.5. A graph is *connected* if and only if there exists a path between any pair of vertices in the graph. Unless additional information is given, we assume graphs are undirected, connected and contain no multiple edges.

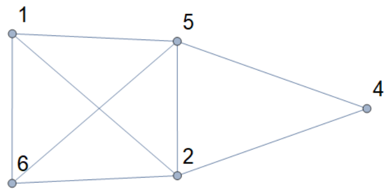
Example 2.1. Given the following figures:



(a) Graph *G*



(b) Subgraph *H*



(c) Induced subgraph H'

Figure 2.1: Simple Graphs

Figure (a) is a graph $G = (V, E)$ with six vertices. Figure (b) is a subgraph of G, and figure (c) is an induced subgraph of G. All three graphs are not *acyclic* graphs. In figure (a), we state some examples for illustrating the above definitions:

- Degrees of some vertices: $deg(2) = 5$, $deg(4) = 3$, and $deg(6) = 3$, etc.
- The subset of neighbors of vertex 2 is $N(2) = \{1, 3, 4, 5, 6\}$.
- A possible path from vertex 6 to vertex 3 is $(6, 5, 4, 3)$. Note that in this figure there are some other possible paths such as $(6, 2, 3)$, $(6, 2, 4, 3)$, etc.
- A cycle in graph G is $(1, 2, 3, 4, 5, 6, 1)$.

Definition 2.6. (Minimum Cuts). A cut in the graph G consists of a set of edges whose removal disconnects the graph. The minimum cut consists of a cut set that is minimal. As we work only with undirected, connected graphs having no *weight* on edges, finding minimum cut problem is the same as finding the minimum set of edges for the cut.

Example 2.2. Given a graph below:

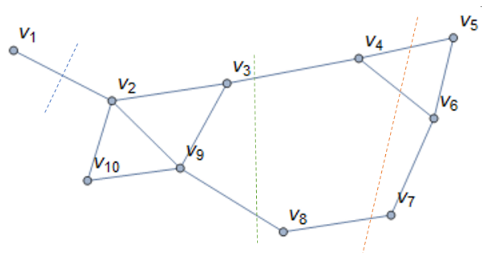


Figure 2.2: Cuts in a graph

The sets $c_1 = \{(v_1, v_2)\}, c_2 = \{(v_3, v_4), (v_8, v_9)\}$ and $c_3 = \{(v_4, v_5), (v_4, v_6), (v_7, v_8)\}$ are the cut sets in the graph. The minimum cut in the graph is $c_1 = \{(v_1, v_2)\}$.

Finally, we present *Laplacian Matrix* and its properties, which are useful in terms of expressing transformations of *divisors* on graph:

Definition 2.7. Given a graph $G = (V, E)$ and the cardinality $|V| = m$, the Laplacian matrix of G is the mxm matrix given by:

$$\Delta = \mathcal{D}_{i,j} - \mathcal{A}_{i,j}$$

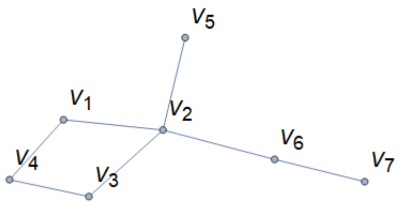
Where $\mathcal{D}$ is the diagonal matrix of G:

$$\mathcal{D}_{i,j} = \begin{cases} deg(i) & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

And $\mathcal{A}$ is the adjacent matrix of G:

$$\mathcal{A}_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

Example 2.3. Consider the following graph with 7 vertices and its Laplacian Matrix:



$$\begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Figure 2.3: Laplacian Matrix Example

2.1.2 Some Graph Families

We introduce definitions of several familiar graphs in this chapter along with examples.

Definition 2.8 (Tree graph)**.** An undirected, connected graph is called *acyclic* if and only if it does not contain any cycle. A graph that contains no cycles, or a connected acyclic graph, is a tree. Formally, let T_n with $n \geq 2$ be a tree graph with n vertices.

Tree graph has an important property that can be utilized often for proofs in the thesis which was proven in [11].

Lemma 2.1. A graph is a tree if and only if there exists a unique path for each pair of vertices in the graph.

Definition 2.9 (Cycle graph)**.** A cycle graph is a graph that contains a single cycle and all vertices are a part of the cycle. A cycle graph with n vertices is denoted as C_n with $n \geq 3$.

Remark 2.1. There are two distinct paths between any pair of vertices in a simple cycle.

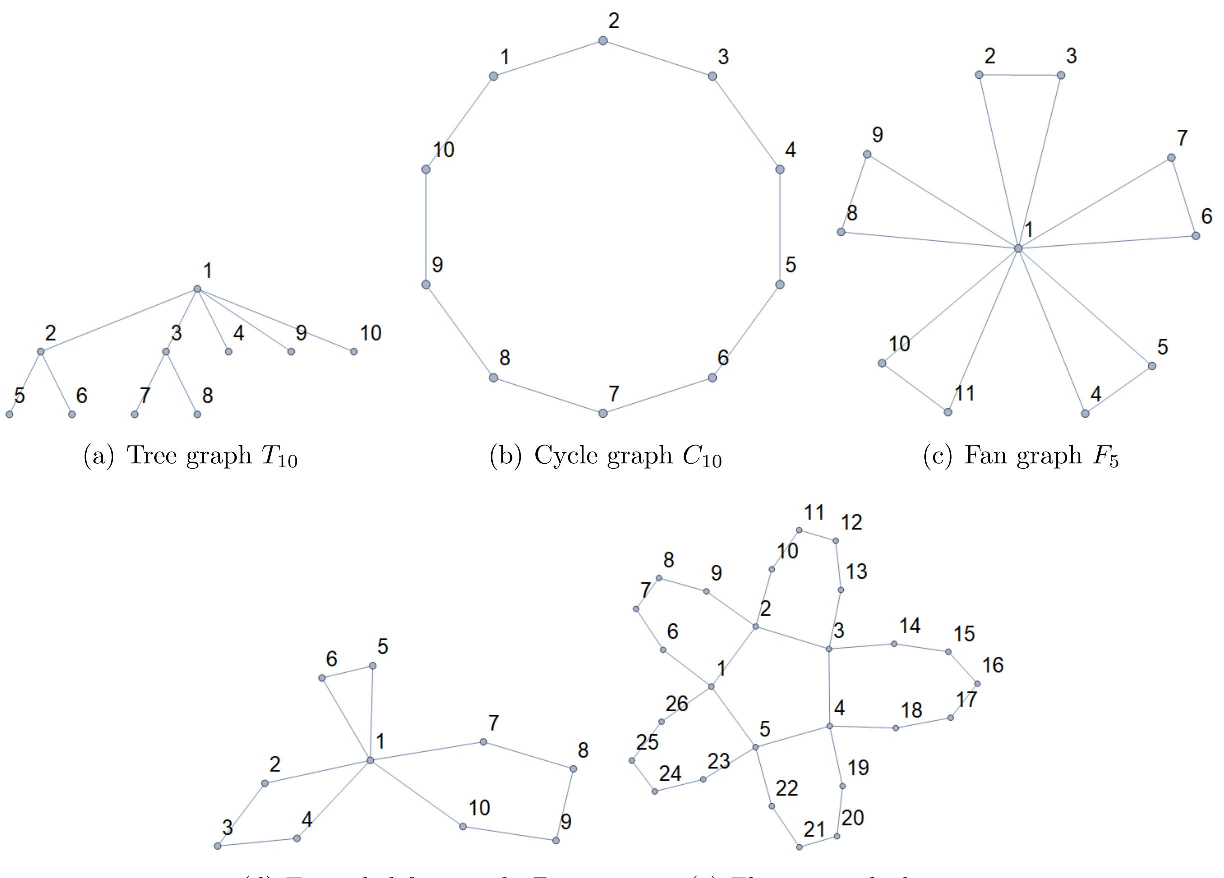
We introduce *fan graph* which was described by *Kavitha* and *Davis* [7], then we continue giving a new definition for the extented version for the original graph:

Definition 2.10 (Fan graph)**.** (Fan Graph). A fan graph F_n with $n \geq 2$ is the graph that is constructed by joining n copies of cycle graph C_3 with a common vertex. An *extended fan graph* $\mathcal{F}_n$ is a fan graph formed by n cycles with the number of vertices larger than 2. Let $C_{i,j}$ be the set of cycles in $\mathcal{F}_n$ such that i represents the index of cycles and j represents the number of nodes in the cycles, with $i \in \{1, ..., n\}$ and $j \geq 3$.

Another interesting graph family was introduced by *Eunice Mphako-Banda* [10]:

Definition 2.11. (Flower Graph). A graph G is called a $(nxm)-$flower graph if it has n vertices which form a n-cycle with n vertices and n set of $(m-2)$ vertices which forms m-cycles such that each m-cycle uniquely intersects with the n-cycle a single edge. This graph is denoted as f_{nxm}, where n-cycle is called the *center*, and m-cycles are called *the petals* of the graph. A graph f_{nxm} has $n(m-1)$ vertices and nm edges. The vertices of n-cycle have degree 4, and the vertices of m-cycles have degree 2.

Example 2.4. Several examples for the previous definitions are given below:



(a) Tree graph T_{10} (b) Cycle graph C_{10} (c) Fan graph F_5

(d) Extended fan graph $\mathcal{F}_3$ (e) Flower graph f_{5x6}

Figure 2.4: Some Family Graphs

Figure (a) represents a tree with 10 nodes. Note that we can construct different trees T_{10}. In figure (b), there are two distinct paths from vertex 9 to vertex 4, which are $(9, 10, 1, 2, 3, 4)$ and $(9, 8, 7, 6, 5, 4)$. In figure (d), the extended fan graph is formed by joining three cycles C_3, C_4 and C_5 with a common vertex.

2.2 Chip-Firing Game

In this section, we present chip-firing game concept as well as divisors on graph. Research in [2] and [4] has shown that working with *divisors* in gonality is similar to working with *configurations* in chip-firing game on graph. Hence, we can define chip-firing game as following:

Definition 2.12. (Chip-firing game). Let $G = (V, E)$ be a graph. A *configuration* (or *divisor* D) on G is an assignment of integer numbers, called *chips*, to each vertex of G. The sum of chips in graph is the degree of divisor D, which is denoted by $Deg(D)$.

We first recall some formal definitions of divisors in [4] before presenting the firing process on chip-firing game.

Definition 2.13. (Principal Divisor). Given a graph G, a divisor P is called principal divisor if there exists a divisor D such that $P = \Delta D$ with Δ as the Laplacian Matrix of G. The set of all principal divisors is denoted by $Prin(G)$.

Definition 2.14. (Effective Divisor). A divisor $D \in \text{Div}(G)$ is effective if and only if $D(v) \geq 0 \ \forall v \in V$.

Definition 2.15. (Equivalence Divisor). Let D and D' be two divisors on G. D and D' are said to be equivalent, denoted by $D \sim D'$, if and only if $D - D' = P \in Prin(\text{G})$. We call $D - D'$ a transformation from D to D'.

Definition 2.16. (Firing a vertex) Given a graph $G = (V, E)$ and a divisor D. Firing a vertex $v_i \in V$ causes the vertex to lose $deg(v)$ chips, and each chip is moved along an incident edge to each neighbor of v. In terms of Laplacian Matrix, firing a vertex $v_i \in V$ results in an equivalent divisor D' such that $D' = D - \Delta v_i$. We denote the firing (transformation) process as $D \xrightarrow{v_i} D'$, where v_i represents a vertex to be fired. A vertex that holds negative number of chips is called *in debt.*

Definition 2.17. (Firing a subset). A firing multiset A contains the vertices to be fired. Given a graph $G = (V, E)$, firing A results in an equivalent divisor D' such that $D' = D - \Delta\mathcal{H}$ where $\mathcal{H}$ is the divisor that describes the number of times each vertex is fired. We denote the firing process (transformation from D to D') as $D \xrightarrow{A} D'$.

A *firing sequence* can be constructed by firing some multisets on a divisor. Thus, we recall an important lemma in [2]:

Lemma 2.2. Let D and D' be divisors of G and c, c' be the corresponding configurations of D and D' in chip-firing game respectively. D and D' are equivalent if and only if there exists a sequence of firings that transforms c to c'.

Firing a subset of vertices causes chips to move to vertices that are not in the firing set through the incident edges. For instance, given a firing set $A = \{v_1, v_2\}$ and a cycle C_5. We know that $E(v_1) = \{v_2, v_5\}$ and $E(v_2) = \{v_1, v_3\}$. Firing A causes v_1 to receive one chip from v_2 and v_2 to receive one chip from v_1 via the edge v_1v_2. Hence, the number of chips remained unchanged for v_1 and v_2, only v_5 and v_3 receive a chip from v_1 and v_2 respectively. Based on this property, we introduce a definition of *out-degree*:

Definition 2.18. (out-degree). Let A be a firing set on divisor D in $G = (V, E)$. The *out-degree* set of vertex $v \in A$ is the number of incident edges to v that leave A:

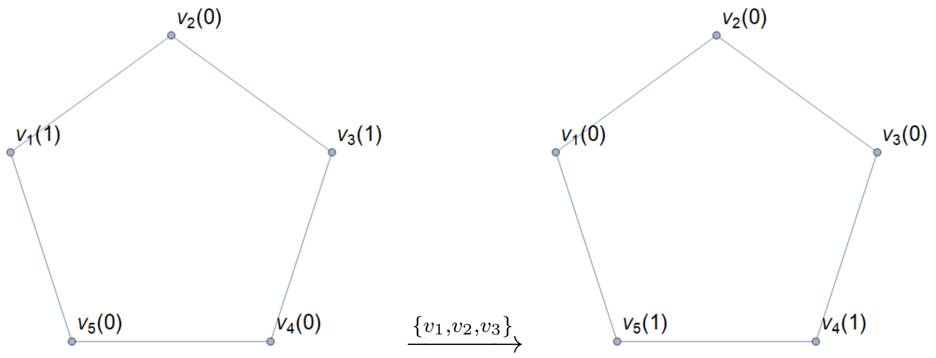
$$outdeg_A(v) = \{(v, w) \in E \mid w \in V \setminus A\}$$

Let D be a divisor D and a non-empty set $A \subsetneq V$ be a firing set in G. A is said to be *valid* if and only if all vertices in G are not in debt after firing A. In other words, A is *valid* if and only if $D(v) \geq |outdeg_A(v)| \ \forall v \in V$.

The out-degree of a vertex v describes how many chips vertex v loses after firing the subset A. Thus, we have the following remark:

Remark 2.2. Let A be a firing set $G = (V, E)$. Firing A causes chips to move out of A through edges in the out-degree set of each vertex in A to its neighbors, and chips on vertices in A that has no edge in the out-degree set remain unchanged after firing A.

Example 2.5. Consider a cycle C_5:



(a) Before Firing (b) After Firing

Figure 2.5: Chip-Firing Game On C_5

Let D, D' be the divisors that represent the configurations in (a) and (b) respectively. Each vertex is assigned with an integer number as chips on the divisors. Firing a set $A = \{v_1, v_2, v_3\}$ resulted in D transforming to D', which is denoted as $D \xrightarrow{\{v_1,v_2,v_3\}} D'$, or $D \xrightarrow{A} D'$. A chip is moved from v_1 to v_5, and the other chip is moved from v_3 to v_4. Since v_2 has no edge in the outdegree set, the number of chips in v_2 remains unchanged after firing A.

We have defined that given a divisor D on a graph G, D can be transformed to an equivalent divisor D' by applying Laplacian Matrix to a divisor $\mathcal{H}$ that describes how often a vertex is fired. Hence, we have the following concept based on the idea of splitting the divisor $\mathcal{H}$ into different layers:

Definition 2.19. (Level Set Decomposition). Let $D, D' \in \text{Div}(G)$ be equivalent divisors such that $D' = D - \Delta\mathcal{H}$ where Δ is Laplacian Matrix and $\mathcal{H} \in \text{Div}(G)$. Let $m = \max\{\mathcal{H}(v) \mid v \in V\}$ and $k = m$ - $\min\{\mathcal{H}(v) \mid v \in V\}$. The level set decomposition of $\mathcal{H}$ is the collection of subsets $A_0 \subset A_1 \subset A_2 \subset ... \subset A_k$ that is given by:

$$A_i = \{v \in V \mid \mathcal{H}(v) \geq m - i\} \text{ with } i \in \{0, 1, ..., k\}$$

Note that we can choose different divisors by integer multiple of all-ones vector plus $\mathcal{H}$ that result in the same level set decomposition. For convenience, choose an effective divisor $\mathcal{H}$, thus $k = m$.

Example 2.6. Given a graph $G = (V, E)$ with $|V| = n$ and two equivalent divisors D and D'. There exists some divisors $\mathcal{H}$ such that $D' = D - \Delta\mathcal{H}$. Choosing any divisors below would result in the same $\Delta\mathcal{H}$:

$$\mathcal{H}' = \mathcal{H} + t \left.\begin{bmatrix} 1 \\ 1 \\ .. \\ 1 \end{bmatrix}\right\} n \text{ with } t \in \mathbb{Z}$$

Definition 2.20. (Divisors of level set decomposition). Let $D_0, D_1, ..., D_k$ be the *associated divisors* of the level set decomposition $A_0, A_1, ..., A_k$ in which D_i is given by:

$$D_0 = D,$$
$$D_{i+1} = D_i - \Delta\mathcal{H}_{A_i} \text{ with } i \in \{1, 2, ..., k\}$$

Where divisor $\mathcal{H}_{A_i}$ is defined as $\mathcal{H}_{A_i}(v) = 1$ if $v \in A_i$ and $\mathcal{H}_{A_i}(v) = 0$ otherwise for all $v \in V$.

Example 2.7. Take the graph of **Example 3.1**. Let D and D' be the effective divisors that present configurations in figure (a) and figure (c) respectively, we have $D \sim D'$. Choose a divisor $\mathcal{H}$ such that:

$$\mathcal{H} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

By taking the divisor with Laplacian matrix Δ, we obtain:

$$D' = D - \Delta\mathcal{H}$$

Let $m = \max\{\mathcal{H}(v) \mid v \in V\} = 2$ and $k = m - \min\{\mathcal{H}(v) \mid v \in V\} = 0$. We denote the level set decomposition of $\mathcal{H}$:

$$\begin{aligned} A_0 &= \{v \in V \mid \mathcal{H}(v) \geq 2\} = \{v_4\}, \\ A_1 &= \{v \in V \mid \mathcal{H}(v) \geq 1\} = \{v_1, v_2, v_4\}, \\ A_2 &= \{v \in V \mid \mathcal{H}(v) \geq 0\} = \{v_1, v_2, v_3, v_4, v_5, v_6\} \end{aligned}$$

The sequence of divisors that are associated with the set decomposition are:

$$D_0 = D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix}, D_1 = D_0 - \Delta \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, D_2 = D_1 - \Delta \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = D'$$

Now we can recall some results in [2] which show relationship between level set decomposition and minimum cuts:

Theorem 2.3. Let $A_0, A_1, ..., A_k$ be the level set decomposition that transforms D to D' and $D_0, D_1, ..., D_k$ be the associated divisors to the level set decomposition, we have:

$$D_i(v) \geq \min(D(v), D'(v)) \ \forall i \in \{0, ..., k\} \text{ and } \forall v \in V$$

Consequently, all divisors D_i are effective and the firing sets A_i corresponding to the divisors are *valid*.

Additionally, an important theorem of *cuts* in graphs which was proven in [2] is given by:

Theorem 2.4. Given a graph $G = (V, E)$, let $v \in V$ and $W \subset V$, with $v \notin W$. If $|W| \geq n - 1$ and for each $w \in W$, the minimum cut between w and v is at least n then $gon(G) \geq n$. The theorem is based on the main idea that the number of chips moved out of a firing set are at least the minimal cut between a vertex in the firing subset and a vertex outside it.

Finally, Concept of *rank* of a divisor is not mentioned in the thesis since we assume to work with graphs that have the rank of divisors larger than one, which does not affect the results of finding gonality.

Chapter 3

Gonality Of A graph

We present preliminary concepts on gonality and some known results by other authors in [4] and [2].

3.1 Gonality

Study of Jelco M. Bodewes [2] showed three equivalent definitions of gonality and several important related theorems. The first definition is for better understanding at intuitive level. The second definition is conducted based on the *rank* of divisors on graph [9]. Although it is formal, reasoning about gonality based on *rank* of a divisor is difficult. Thus, the author introduced the third definition with the concept of *reachability* of divisors on graphs, which is the main concept utilized for the thesis. For that reason, we only mention the first and the third definitions of gonality, which were proven to be equivalent to the original definition. The most crucial property in chip-firing game used for reasoning about gonality of graphs is the winning configuration in chip-firing game:

Definition 3.1. Let $G = (V, E)$ be a graph and an initial configuration (divisor D) on G, a winning configuration is achieved if for each vertex v in G, there exists a firing sequence that v ends up holding at least one chip and the remaining vertices are not in debt.

Definition 3.2. (Gonality). Let $G = (V, E)$ be a graph. The divisorial gonality $gon(G)$ of G is the minimum number of degree of all winning configurations for chip-firing game.

This means given an effective divisor as an initial configuration which is prepared for the graph before starting the game, we try to assign a smallest possible number of chips on graph (divisor with lowest degree), and see whether a winning configuration is achieved. Finding a winning configuration with the minimum number of chips is equivalent to the goal of finding the divisorial gonality of that graph. A method to achieve such configuration is through the *reachability* property of a divisor:

Definition 3.3. Let D be a divisor on $G = (V, E)$ and a vertex $v \in V$. D is said to *reach* v if and only if there exists an effective divisor D' such that $D \sim D'$ and $D'(v) \geq 1$.

Definition 3.4. (Gonality). Given a graph $G = (V, E)$, the gonality $gon(G)$ is the lowest degree of the effective divisor D that reaches all vertices $v \in V$.

We recall a theorem and a corollary related to gonality in [2]:

Theorem 3.1. Let D be a divisor on G, D reaches all vertices v if and only if the corresponding configuration of D is a winning configuration.

Corollary 3.1.1. $gon(G) \geq 1$ for any given graph G.

Example 3.1. Given a graph G in the figures below:

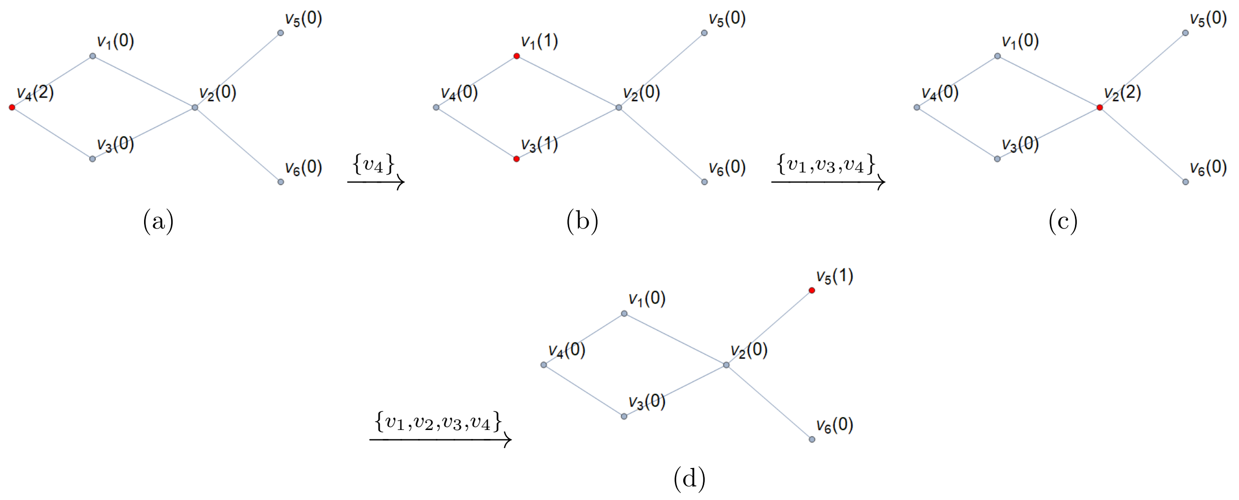


Figure 3.1: Gonality on a graph

Given a divisor D as the initial configuration in figure (a), we see that D reaches all the vertices in the graph by the firing sets described in the example, thus $gon(G) \leq 2$. Since the graph contains a tree, so $gon(G) \geq 2$ according to theorem 3.2. We conclude that $gon(G) = 2$.

3.2 Gonality of special graph families

We present gonality of *tree* and *cycle* graphs in this section. The firing patterns in these two graph families are important for conducting proofs for the remaining parts of the thesis.

Theorem 3.2. Given a graph $G = (V, E)$, the gonality of G is 1 if and only if G is a tree.

Proof. We first assume G is a tree and show that chip can be moved between any pair of vertices in G. By **Lemma** 2.1, there exists a unique path between any pair of vertices in G, thus chips are moved from one vertex to another via a unique path between them. Choose a random vertex $v \in V$ and a divisor D on G such that $D(v) = 1$ and $D(w) = 0$ for all $w \in V$ and $w \neq v$. Choose a vertex v' and $D(v') = 0$, we move chip from v to v'. Let u be the neighbor of v as well as the first vertex in the path from v to v' and G_v be the induced subgraph attached to v that does not contain u. Firing G_v causes one chip to move from v to u through the only out-degree edge in the path to v'. From u, let u' be the neighbor of u and also a vertex on the path from u to v'. We construct the same pattern, thus chip is able to move from u to u'. By repeating the same constructions, a chip from v can be moved to v' with an effective divisor. We also claim that this process can be applied between any chosen pair of vertices v, v' such that $D(v) = 1$ and $D(v') = 0$ on G. Hence D reaches all vertices in V, and $gon(G) \leq 1$. By **Corollary** 3.1.1, $gon(G) \geq 1$ for any given graph, we conclude $gon(G) = 1$.

Now assume there exists a graph G that contains a cycle and has $gon(G) = 1$. Choose two vertices $v, u \in V$ that belong to the cycle in G. We construct a set W such that $u \in W$ and $v \notin W$. For any chosen pair of vertices with the defined properties, we have the minimum cut between them at least 2 since they are in the same cycle. By **Theorem** 2.4, we have $|W| = 1$ and the minimum cut between w and v is at least 2, thus $gon(G) \geq 2$, which is a contradiction. We claim that such graph with gonality 1 does not exist, and if $gon(G) = 1$, G must be a tree. Consequently, we conclude that given a graph G, gonality of G is $gon(G) = 1$ if and only if G is a tree. □

Example 3.2. Consider the tree in **Figure 3.2** with the initial configuration in figure (a). We assume red vertex holds a chip, and the colored vertices (yellow and red) belong to the firing A_i in each figure.

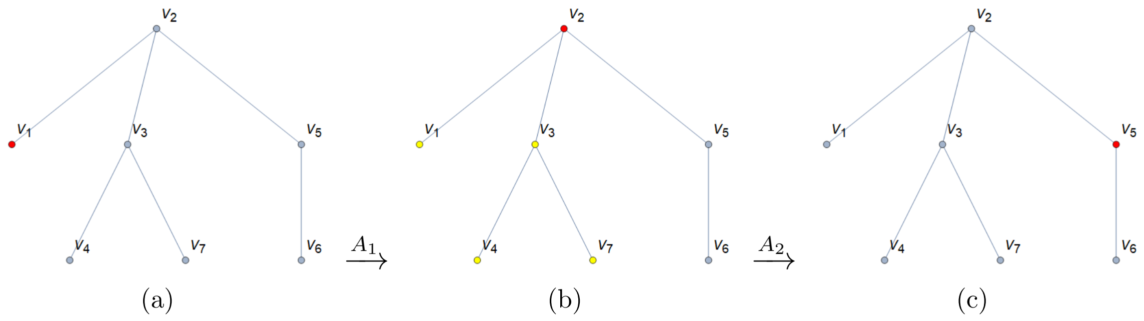


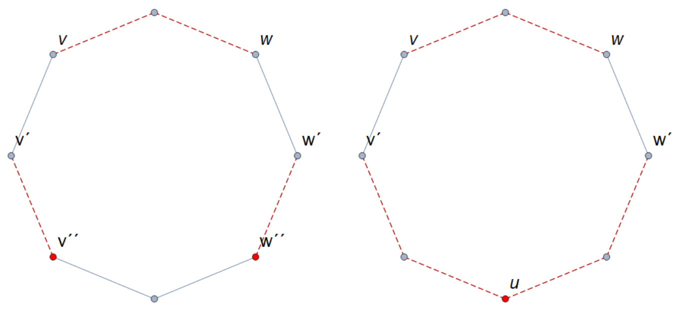
Figure 3.2: Moving chip from v_1 to v_5

In this example, firing $A_1 = \{v_1\}$ and $A_2 = \{v_1, v_2, v_3, v_4, v_7\}$ causes a chip to move from vertex v_1 to vertex v_5.

Theorem 3.3. Given a simple cycle graph C_n with $n \geq 3$, then $gon(C_n) = 2$.

Proof. By **Theorem 3.2**, $gon(G) \geq 2$. Now we only need to show a divisor of degree 2 that is able to reach all vertices of C_n. Choose two vertices $v, w \in C_n$ with $v \neq w$ and a divisor D such that $D(v) = D(w) = 1$. Let $u \in C_n$ be a vertex such that $u \neq v, w$. We will move either a chip from v or a chip from w to u. In a cycle, we observe that there are always two distinct paths between any pair of vertices. Let P_v, P_w be subsets of the paths from v to u and from w to u respectively, with $w \notin P_v$ and $v \notin P_w$. Let v', w' be the first vertices on the path from v to u and from w to u in that order. Let $G_{v,w}$ be the induced subgraph attached to v, w that does not contain v' and w'. Firing $G_{v,w}$ causes one chip to move from v to v' and one chip to move from w to w'. By repeating the same constructions, two chips are moved simultaneously each time, each chip ends up on the next vertex of each path. After at least one chip ends up on u, the remaining chip is distributed as following:

- Case 1: If $|P_v| < |P_w|$ and one chip reaches u, the other chip ends up in a vertex w'' such that the path from w to w'', whose subset is $P''_w \subset P_w$, is equal to the path from v to u, that is $|P''_w| = |P_v|$. The similar case also occurs if $|P_v| > |P_w|$, but one chip ends up in a vertex $v'' \in |P_v|$ with same property as w''.
- Case 2: If $|P_v| = |P_w|$, u ends up having two chips sent from both v and w.



(a) Case 1 (b) Case 2

Figure 3.3: Moving chips in cycle

Hence, it is always possible to move at least one chip to u from v and w. This pattern can be applied for any vertices v, w and u in the cycle. We conclude that $gon(G) = 2$. □

The firing pattern is useful for checking the compatibility of cycles using *Reduction Rules* in section 4.3. We end the chapter with a remark:

Remark 3.1. Any pair of vertices in the cycle induces a minimum cut of two, thus firing valid sets must move at least two chips at the same time out of the firing sets. Furthermore, two chips can only be fired in two different directions along the two arcs of the cycle towards a target vertex.

Chapter 4

Results On Gonality

In this chapter, we present results on gonality of several graphs introduced in **Chapter 2**. Then we recall a set of *reduction rules* [2] and state a lemma to check the compatibility of the constraint set on cycles. Finally, we define special cases that gonality of *tree with cycles* graphs is 2.

4.1 Gonality on other classes of graphs

For some graphs, their gonality can be computed or restricted with an upper bound by applying the same firing pattern as the one for simple cycles in **Theorem 3.3**. In this section, we present gonality on some graph families given in section 2.1.2.

Theorem 4.1. Given a fan graph F_n and an extended fan graph $\mathcal{F}_n$, then:

(i) $gon(F_n) = 2$;

(ii) $gon(\mathcal{F}_n) = 2$.

Proof. We have $gon(F_n)$, $gon(\mathcal{F}_n) \geq 2$ by **Theorem 3.2**. Let D be an effective divisor with degree 2 on F_n such that $D(v) = 2$ with v as the common vertex that joins the cycles. Let C be a cycle in F_n and G_v be an induced subgraph such that $G_v = F_n - C \backslash \{v\}$. Let $u, w \in C$ be the remaining vertices of C and $u, w \neq v$. By **Remark 2.2**, firing G_v causes v to lose 2 chips, and each vertex u, w receives a chip from v. We claim D reaches vertices in C. Repeat the same construction for the remaining cycles, we claim D reaches all vertices in F_n, thus $gon(F_n) = 2$.

Now let D be an effective divisor with degree 2 on $\mathcal{F}_n$ such that $D(v) = 2$ with v as the common vertex that joins the cycles. Let C be a cycle with any number of vertices larger or equal 3. We construct the same firing pattern in F_n, which ends up having a chip on u and a chip on w with $u, w \in C$. Repeat the process in proof of **Theorem 3.3**, D reaches all vertices of C. Consequently, D reaches vertices of all cycles in the graph with the same process, we conclude $gon(\mathcal{F}_n) = 2$. □

Example 4.1. Consider a fan graph F_3 below:

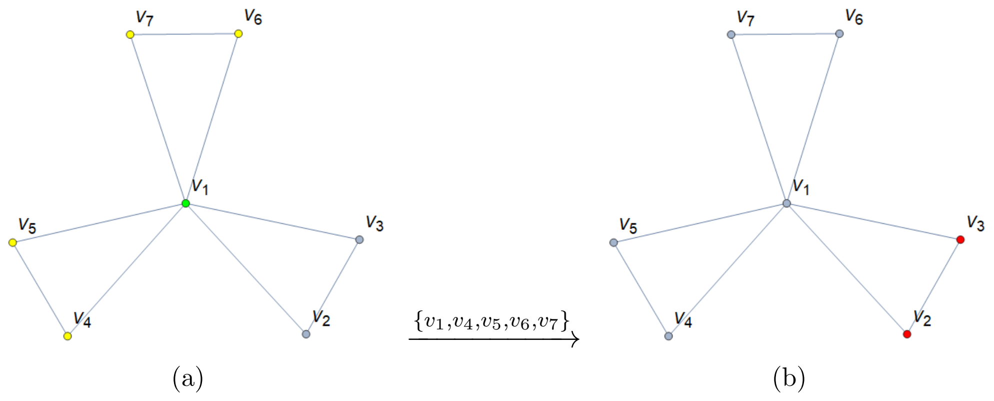


Figure 4.1: Moving chips from v_1 to v_2 and v_3 in F_3

In figure (a), v_1 holds 2 chips. Firing the set of colored vertices moves a chip to v_2 and v_3 (Figure (b)). Repeat the process, we have $gon(F_3) = 2$.

Example 4.2. Consider a graph $\mathcal{F}_2$ formed by two cycles C_3 and C_5 which is joined by vertex v_1 in the figure below:

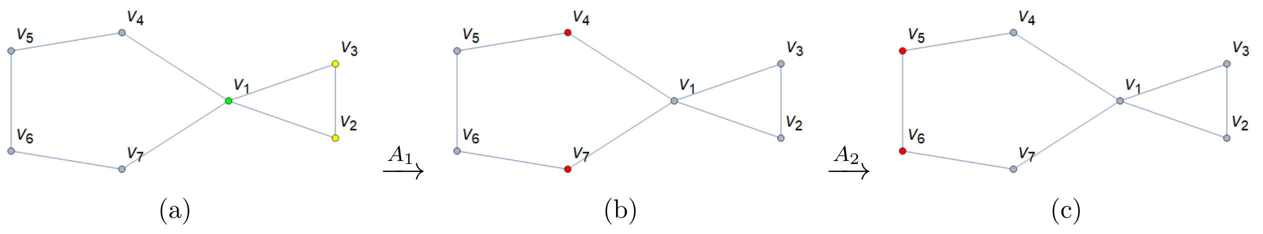


Figure 4.2: Moving chips from v_1 to v_5 and v_6 in $\mathcal{F}_2$

Firing a subset $A_1 = \{v_1, v_2, v_3\}$ causes a chip to end up on v_4 and v_7. We continue firing the set $A_2 = A_1 \cup \{v_4, v_7\}$, which leads to the configuration in figure (c), thus D reaches all vertices of C_5. Repeat the same steps for C_3, we conclude $gon(\mathcal{F}_2) = 2$.

Next, we show the upper bound of gonality on flower graphs in section 2.1.2.

Theorem 4.2. Given a flower graph f_{nxm}, then $gon(f_{nxm}) \leq n$, where n is the number of vertices in the center of the flower graph.

Proof. Let D be an effective divisor on f_{nxm} with degree n such that each vertex of the center cycle holds a chip. Let C be a petal cycle and v, w be two vertices in C that joins the center cycle. Let $G_{v,w}$ be the induced subgraph such that $G_{v,w} = f_{nxm} - C \backslash \{v, w\}$. Firing $G_{v,w}$, by **Remark 2.2**, causes a chip to move from v to v' and a chip from w to w' with v', w' as neighbors of v and w on C respectively. Construct firing process in cycle as proof of **Theorem 3.3**, then D reaches all vertices in C. This process can be repeated for any petal cycle, thus D reaches all vertices in the graph. We conclude $gon(f_{nxm}) \leq n$. □

Corollary 4.2.1. Given an extended version of a flower graph where its petals have more than 2 vertices, then the gonality of this graph is at most the number of vertices in the center cycle.

Proof. It it straightforward that we can construct the same firing set in **Theorem 4.2** to move 2 chips to any petal cycle, then apply the firing process as proof for **Theorem 3.3** regardless of the number of vertices in the petal. Hence, the corollary holds. □

Example 4.3. Consider a flower graph $f_{5\text{x}4}$. Let D be an effective divisor with degree 5 such that each vertex of the center cycle holds a chip:

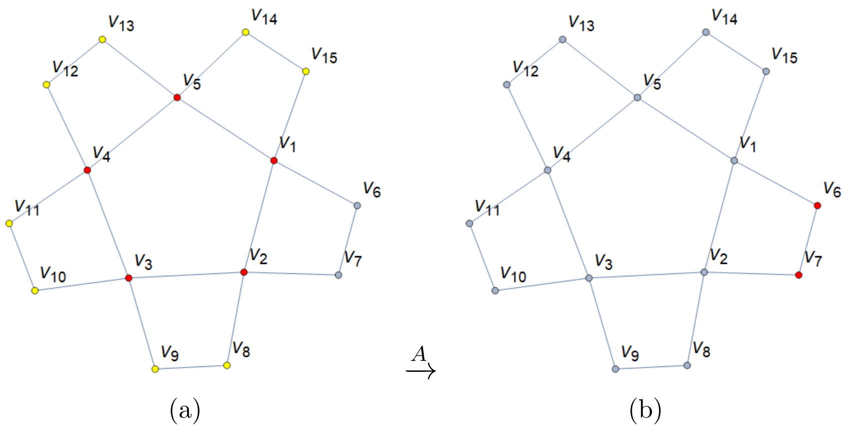


Figure 4.3: Moving chips from v_1 and v_2 to v_6 and v_7 in $f_{5\text{x}4}$

Let A be the firing set that contains all colored vertices in figure (a). Firing A causes a chip to move from v_1 to v_6, and a chip to move from v_2 to v_7 (in figure (b)). Repeating the process with the remaining petals, we have $gon(f_{5\text{x}4}) \leq 5$.

4.2 Tree Graph With Cycles

In this section, we define a new graph class. *Tree graph with cycles* is a graph that contains at least a cycle and every vertex of the graph belongs to at most a cycle. We first present the formal definition of this type of graph:

Definition 4.1. Let $T_{m,n} = (V, E)$ be an undirected, connected graph that contains no multiple edges between any pair of vertices. $T_{m,n}$ is a *tree graph with cycles* with the following properties:

- The number of vertices in the graph is $|V| = m$.
- The number of cycles in the graph is n, with $n \in \mathbb{Z}$ and $n \geq 1$.
- Each node belongs to at most a cycle in the graph. In other words, cycles have disjoint subset of vertices in the graph.

Let $\mathcal{T}_{m,n}$ be the class of all possible graphs $T_{m,n}$ that share the same properties.

Remark 4.1. Every vertex is a part of at most one cycle. Thus, if we consider cycles as nodes, then cycles are connected by some subgraphs that have the structure of a tree.

In order to work with proofs, we define some important properties of the graph:

Definition 4.2. Given a *tree with cycles* graph $T_{m,n}$. Let C_i and C_j be two cycles in the graph with $v_i \in C_i$ and $v_j \in C_j$:

(i) C_i and C_j are *acyclic-connected* (or v_i and v_j are *acyclic-connected*) if and only if there exists a unique path between v_i and v_j that does not contain any other vertex in either C_i or C_j.

(ii) The unique path between v_i and v_j is called *acyclic bridge*.

(iii) The vertices that are *acyclic-connected* are called *appended vertices.*

(iv) A cycle is called a *cycle-leaf* if and only if is has one *appended vertex.*

Example 4.4. Given a graph $T_{30,3}$ with 2 cycles C_6 and a cycle C_8:

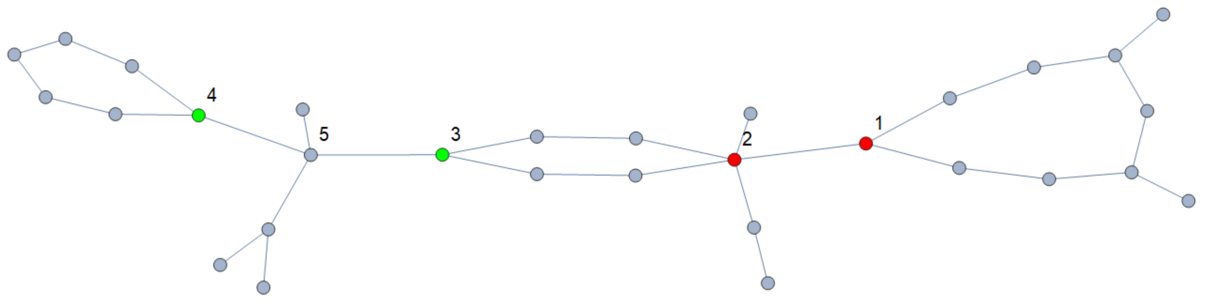


Figure 4.4: A graph $T_{30,3}$

In the figure, the pair of green vertices (3 and 4) and the pair of red vertices (2 and 1) are acyclic-connected. However, vertex 3 and vertex 1, for example, are not acyclic-connected, but they are both appended vertices. Hence, all colored vertices are appended vertices. Path from 4 to 3 and path from 2 to 1 are the acyclic bridges.

Example 4.5. Consider another graph $T_{30,3}$ with the same properties in **Example 4.4**:

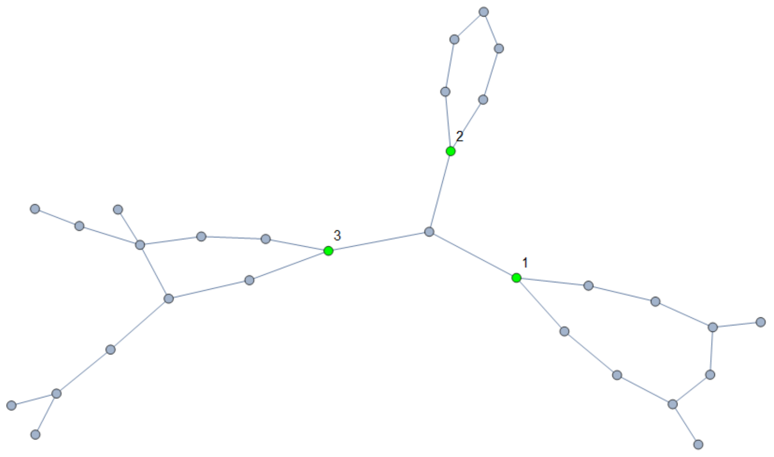


Figure 4.5: Another graph $T_{30,3}$

Note that each pair of cycle is acyclic-connected, and the unique paths between each pair of cycle share some common vertices with each other.

Lemma 4.3. For any graph $T_{m,n}$ with $n \geq 2$, there are at least two *cycle-leaves.*

Proof. It is clear that for $n = 2$, both of the cycles are cycle-leaves. For $n > 2$, assume there is only one cycle-leaf in $T_{m,n}$, then the remaining $(n-1)$ cycles have at least 2 appended vertices. This leads to a creation of an additional cycle that shares some common vertices in these cycles. This breaks the property of the original definition and results in a contradiction.

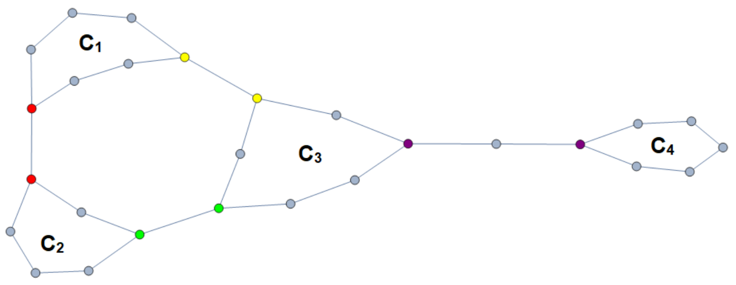


Figure 4.6: Contradiction in $T_{m,n}$ with 1 cycle-leaf

We conclude that given a tree with cycles graph with more than 1 cycle, there must be least two cycle-leaves in the graph. □

Before calculating the gonality of these graphs, we introduce an important lemma that is used to move chips between *acyclic bridges*:

Lemma 4.4. Given a tree with cycles graph $T_{m,n}$ with $n \geq 2$. Let C_i and C_j be any cycles in the graph such that C_i and C_j are acyclic-connected by two appended vertices $v_i \in C_i$ and $v_j \in C_j$, then a chip from v_i can be moved to v_j, or the other way around.

Proof. C_i and C_j are acyclic-connected by v_i and v_j and they have disjoint subsets of vertices. Since the unique path P from v_i to v_j contains vertices that are not a part of any cycle except v_i and v_j, P has a structure of a tree by **Remark 4.1**. By constructing the firing process in a tree as proof of **Theorem 3.2**, we can move a chip from v_i to v_j or the other way around as they are in the same tree. We conclude the lemma holds. □

Theorem 4.5. Given a graph $T_{m,n}$ with $1 \leq n \leq 2$, then $gon(T_{m,n}) = 2$.

Proof. By **Theorem 3.2**, $gon(T_{m,n}) \geq 2$ since the graph contains at least a cycle. For $n = 1$, $T_{m,1}$ has a cycle with some vertices having degree larger than 2 whose subgraphs are trees. Let D be an effective divisor with degree 2 on the graph such that two random vertices in C hold a chip. We apply firing process in cycle as proof for **Theorem 3.3**, then D reaches all vertices of C. Let v be a vertex in C with $deg(v) > 2$ and G_v be the sub-tree attached to v. Constructing the same firing process in a tree as proof for **Theorem 3.2**, we claim D reaches all vertices in G_v. Similarly, this process can be applied for any vertex with degree larger than 2 in the graph, thus D reaches all vertices in $T_{m,1}$. We conclude $gon(T_{m,1}) = 2$.

For $n = 2$, let C_i and C_j be two cycles in $T_{m,2}$ that are acyclic-connected by $v_i \in C_i$ and $v_j \in C_j$. Let D be an effective divisor with degree 2 in $T_{m,2}$ such that $D(v_i) = D(v_j) = 1$. By **Lemma 4.4**, a chip from v_i can be moved to v_j, thus we can obtain an effective divisor D' such that $D \sim D'$ and $D'(v_j) = 2$. Let $u, w \in C_j$ be neighbors of v_j, firing the induced subgraph that does not contain u and w causes a chip to move from v_j to u and to w respectively. Repeat the firing process for the case $n = 1$, we claim D reaches all vertices in C_j and sub-trees attached to non-appended vertices in C_j. We can repeat the same process for C_i by moving a chip from v_j to v_i. Since the subgraph attached between v_i and v_j also has the structure of a tree, a chip from v_i or v_j can reach all the vertices in the subgraph due to firing process in proof of **Theorem 3.2**. We conclude that D reaches all vertices in $T_{m,2}$, thus $gon(T_{m,2}) = 2$. □

Example 4.6. Given a graph $T_{8,1}$ with a cycle C_5:

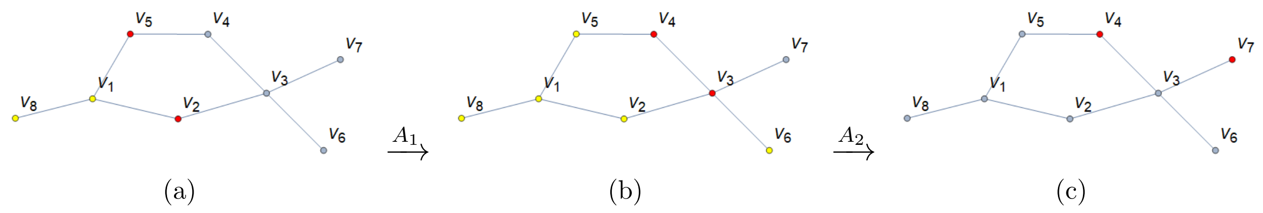


Figure 4.7: Moving chips in $T_{8,1}$

Let A_1, A_2 be the firing set containing the colored vertices in figure (a) and figure (b) respectively. Firing A_1 and A_2 causes a chip to move to v_7, which belongs to the sub-tree attached to v_3. We can construct similar firing set as A_1 to move chips on vertices of the cycle and as A_2 to move chips to the vertices of the sub-trees.

Example 4.7. Consider a graph $T_{9,2}$ with two cycles C_3 and C_4. We show how to move a chip from an appended vertex to another cycle.

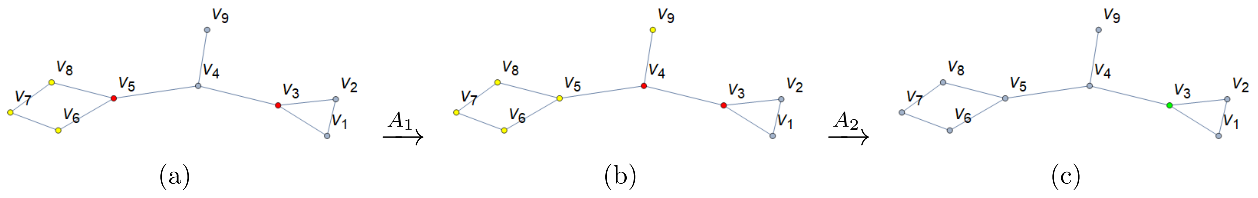


Figure 4.8: Moving a chip to another cycle in $T_{9,2}$

Let A and A' be the firing subset containing the colored vertices in figure (a) and figure (b) respectively. Firing $A_1 = A\backslash\{v_3\}$ and $A_2 = A'\backslash\{v_3\}$ causes a chip from v_5 to reach v_3. Let G_v be the subgraph attached to v_3 such that $G_v = T_{9,2} - C_3\backslash\{v_3\}$. Firing G_v causes a chip to move to v_2 and a chip to move to v_1 from v_3. We can repeat the similar construction for C_4 for a chip to reach v_8 and a chip to reach v_6. Then we construct the firing set as proof in **Theorem 3.3** to move chips to all the vertices in C_4. For the configuration in figure (b) with a chip on v_4, firing the set $T_{9,2}\backslash\{v_9\}$ moves the chip from v_4 to v_9. We conclude D reaches all vertices in the graph, thus $gon(T_{9,2}) = 2$.

Finally, we can present the upper bound of $T_{m,n}$ having more than 2 cycles:

Theorem 4.6. Given a graph $T_{m,n}$ with $n \geq 3$, then $gon(T_{m,n}) \leq n$.

Proof. By proof of **Theorem 4.5**, a divisor D that assigns two chips on a cycle can reach all vertices of the cycle, sub-trees attached to non-appended vertices. The induced subgraphs between 2 or more appended vertices (acyclic bridges that share some common vertices with each other in the case $n \geq 3$) also have the structure of a tree, thus firing process in proof of **Theorem 3.2** can also be applied for D to reach all the vertices in these subgraphs. Now we prove that for each cycle, there exists an effective and equivalent divisor that has 2 chips on the cycle.

By **Lemma 4.3**, $T_{m,n}$ has at least two cycle-leaves. Let C_i and C_j be two cycle-leaves with $v_i \in C_i$ and $v_j \in C_j$ as two appended vertices. Let D be an effective divisor with degree n on $T_{m,n}$ such that each cycle contains an appended vertex that holds a chip. Note that we have $D(v_i) = D(v_j) = 1$ as they are the only appended vertices on C_i and C_j. Let C_l be a cycle in $T_{m,n}$ with $l \neq i$, we move a chip from C_i to C_l. In order to reach C_l, a chip from C_i needs to cross some acyclic bridges through the cycles between C_i andn C_l. Let C'_i be the first cycle that is acyclic-connected to C_i. By **Lemma 4.4**, a chip from v_i can be moved to $v'_i \in C'_i$ and

v_i, v_i' are acyclic-connected. With two chips on C_i', by proof of **Theorem 3.3**, we can move a chip to the appended vertex that is acyclic connected to another cycle C_i'' from the path to C_l. Repeat the same construction for each cycle until reaching C_l, we can finally move a chip from v_i to C_l. We claim a chip from C_i can be moved to any cycle C_l with $l \neq i$.

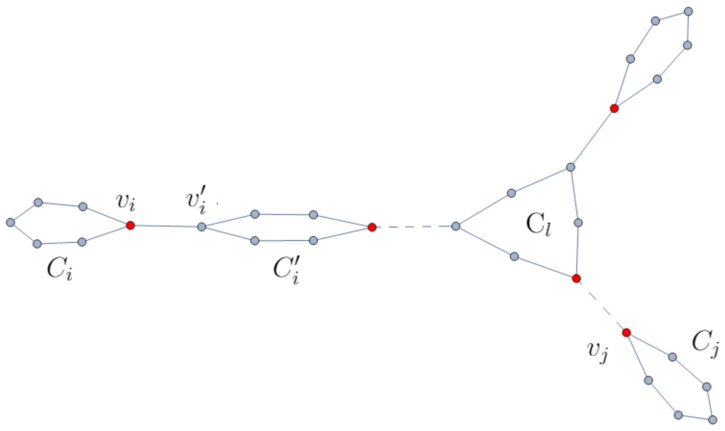


Figure 4.9: Moving chips between acyclic bridges

Conversely, a chip from v_j in C_j can be moved to C_i by the same construction. We claim that for such initial configuration D, each cycle is guaranteed to receive 2 chips. Thus, we claim D reaches all vertices in $T_{m,n}$. We conclude that $gon(T_{m,n}) \leq n$. $\square$

In the next section, we can use *reduction rules* in [2] to show another cases that $T_{m,n}$ has gonality 2 with $n \geq 3$.

4.3 Reduction Rules

We recall the *reduction rules* introduced in [2], which are the rules that transform graphs by removing some vertices and edges. The rules have two properties, namely *completeness* and *safeness*. *Safeness* ensures that the reduced graphs stay in the same graph class of the original graph and *completeness* ensures that the graph can be reduced to the empty graph. A rule can be used if and only if it satisfies both properties. We will use these rules to show that there are some *trees with cycles* graphs of gonality 2.

The most essential elements which are used to keep track the changes of reductions rules are *constraints*.

Definition 4.3. Given a graph $G = (V, E)$, the *set of constraints* $\mathscr{C}$ is a set of pairs (v, w) where $v, w \in V$. We denote the constraint (v, v) of a vertex v in G as the *self-constraint* of v.

Definition 4.4. A divisor D is said to *satisfy* a constraint $c = (v, w)$ if from D we can obtain an effective divisor D' by removing one chip at v and one chip at w. Any firing set should either contain both v and w or neither. For $v = w$, D' is obtained after removing 2 chips on v or w.

We now consider the compatibility of a constraint set on a cycle which plays an important role in studying reduction rules:

Definition 4.5. Let C be a cycle and $\mathscr{C}_C \subseteq R$ be the set of constraints that contain a vertex in C. $\mathscr{C}_C$ is *compatible* if:

- If $(v, w) \in \mathscr{C}_C$, then $v, w \in C$.

- For each pair of sets of constraints (v, w) and (v', w') in $\mathscr{C}_C$, the divisor given by assigning a chip to v and w must be equivalent to the divisor that assigns a chip to v' and w'. In other words, given a divisor D with $D(v) = D(w) = 1$, there must be a sequence of firing subsets that transform D to D' such that $D'(v') = D'(w') = 1$. We also call (v, w) and (v', w') are *compatible* to each other.

We can now present the set of reduction rules:

Rule E_1^d: Given a graph with exactly one vertex, remove that vertex.

Rule E_2^d: Given a graph with exactly two vertices connected to each other by a single edge, and $\mathscr{C} = \{(u, v)\}$, remove both vertices.

Rule T_1^d: Let v be a leaf and v has no constraint in $\mathscr{C}$, remove v.

Rule T_2^d: Let v be a leaf that has only one constraint (v, v). Let u be its neighbor, remove v and add the constraint (u, u) if it is not existent.

Rule T_3^d: Let v_1, v_2 be two leaves such that their only constraint on $\mathscr{C}$ is (v_1, v_2). Let u_1 and u_2 be the neighbors of v_1 and v_2 respectively, remove v_1 and v_2 and add constraint (u_1, u_2) if it is not existent.

Rule C_1^d: Let C be a cycle of vertices with degree two, replace C by a new vertex if $\mathscr{C}_C$ on C is *compatible.*

Rule C_2^d: Let C be a cycle with only one vertex v with degree greater than two. If $\mathscr{C}_C$ plus the constraint (v, v) is compatible, remove all vertices in C except v and add constraint (v, v) if it is not existent.

Rule C_3^d: Let C be a cycle with only two vertices u and v with degree greater than two. If there exists a path between u and v which does not share any edge with C and the constraint set $\mathscr{C}_C$ plus the constraint (u, v) is compatible, then remove all vertices of C except u and v, remove all edges in C and add constraint (u, v) if it is not existent.

Theorem 4.7. (Bodewes [2]). Let G be graph and G is not a tree. If G is reducible to the empty graph with the reduction rules above, then $gon(G) = 2$.

Lastly, we present a sufficient condition to check the compatibility of a pair of constraints in the same constraint set of a cycle:

Lemma 4.8. Let (v, w) and (v', w') be two constraints in the same constraint set on a cycle C. Let $P_{vw'}$ be the path from v to w' that does not contain w, and $P_{ww'}$ be the path from w to w' that does not contain v. Assume $|P_{ww'}| < |P_{vw'}|$, let P_{vt} be the path from v to t by removing some vertices from the end of the path $P_{vw'}$ such that $|P_{ww'}| = |P_{vt}|$. If $t = v'$, then (v, w) and (v', w') are compatible.

Proof. Let D be an effective divisor with degree 2 such that $D(v) = D(w) = 1$. Let D' be an effective divisor in the graph such that $D \sim D'$ and $D'(w') = 1$. Call A_i the level set decomposition that transforms D to D'. If $v' \in A_i$ and $w' \notin A_i$, by **Definition 4.4**, the constraint (v', w') is not satisfied. By proof of theorem 3.3, we can move either a chip from v

or from w to w'. Since $|P_{ww'}| < |P_{vw'}|$, a chip from w will be moved to w', and a chip will end up on a vertex t from v by assumption. Hence, if $t = v'$, then we have $D'(v') = D'(w') = 1$. We claim the pair of constraints is compatible. For $|P_{ww'}| = |P_{vw'}|$, we have $t = v' = w'$, and the proof still holds. $\square$

For checking the compatibility of the constraint set on a cycle, we simply check whether each pair of constraints on the cycle is compatible.

Example 4.8. Let C_n be a simple cycle with $n \geq 3$. Since $deg(v) = 2$ for all $v \in C_n$, the set $\mathscr{C}_C$ is compatible as it is empty in the beginning. Apply C_2^d to transform cycle into a single vertex, then apply E_1^d to obtain the empty graph. We conclude $gon(C_n) \leq 2$.

Example 4.9. Let F_n be a fan graph, with $n \geq 2$ and $\mathscr{C}$ be the constraint set in F_n. Call v as the common vertex that joins all cycles in F_n. Take any cycle C_3 with an empty constraint set $\mathscr{C}_{C_3}$, we have one vertex with degree greater than 2, that is vertex v. Note that $\mathscr{C}_{C_3}$ plus (v, v) is obviously compatible, since it is the only constraint, we apply C_2^d and obtain a reduced graph by removing C_3 except vertex v, then add (v, v) into $\mathscr{C}$. Note that the constraint (v, v) is in the constraint sets of all remaining cycles C_3'. Thus C_3' plus (v, v) is compatible. We repeatedly apply C_d^2 until there is one cycle left. Note that the only added constraint in the set of this cycle is (v, v), thus the constraint set on the last cycle is compatible. Apply C_1^d then E_1^d to obtain the empty graph. We conclude $gon(F_n) \geq 2$.

Example 4.10. Let $\mathcal{F}_n$ be an extended fan graph. Note that all cycles have only a common vertex, and the number of vertices in a cycle do not affect the *reduction rules*, thus we can apply the same process like in **Example** 4.9 to obtain the empty graph. We conclude $gon(F_n) \leq 2$.

Example 4.11. Given a graph $G = (V, E)$ with the constraint set $\mathscr{C}$:

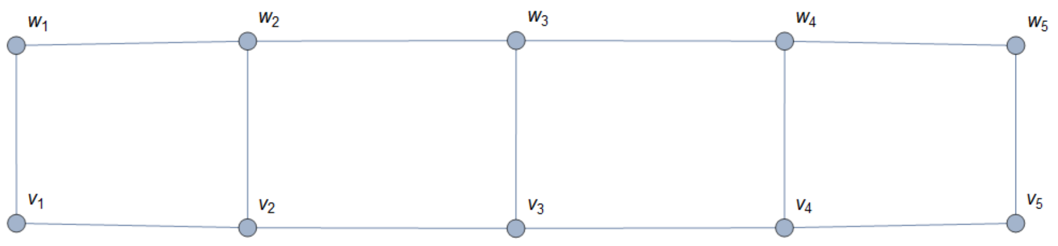


Figure 4.10: Reduction rules on a grid panel graph

We start applying the reduction rules:

- Consider cycle with four vertices w_1, w_2, v_1 and v_2. There is a path from w_2 to v_2 outside of the cycle, thus we can apply C_3^d to remove the edges of the cycles and two vertices w_1 and v_1, then add constraint (w_2, v_2)
- Apply T_3^d on w_2 and v_2, remove w_2, v_2 and the edges (w_2, w_3) and (v_2, v_3), then add constraint (w_3, v_3).
- Consider the cycle with four vertices w_3, w_4, v_3 and v_4, there is a path between w_4 and v_4 outside the cycle. It is easy to see that by **Lemma** 4.8, (w_3, v_3) and (w_4, v_4) are compatible, thus the constraint set on the cycle is compatible. Hence, apply C_3^d to remove the edges of the cycle and two vertices w_3, v_3, then add constraint (w_4, v_4). Now the graph became a tree with four vertices: w_4, w_5, v_4 and v_5.

- Apply T_3^d on w_4 and v_4, remove these two vertices and the edges (w_4, w_5) and (v_4, v_5), then add the constraint (w_5, v_5). Finally, there are only two vertices w_5 and v_5 remained. Apply E_2^d, we obtain the empty graph. We conclude $gon(G) \leq 2$.

Finally, we characterize the special cases where gonality of *tree with cycles* graphs have gonality 2 using reduction rules:

Theorem 4.9. *Let $T_{m,n}$ be a tree with cycles with $n \geq 3$. We denote $\mathscr{C}$ as the constraint set of $T_{m,n}$ and $\mathscr{C}_{C_i}$ as the constraint set on any cycle C_i. We have $gon(T_{m,n}) = 2$ if for each pair of appended vertices u and t in the same cycle, the pair of constraints (u, u) and (t, t) is compatible.*

Proof. We apply reduction rules to the graph in the following steps:

1. Apply T_1^d to remove all leaves of the graph. Repeat until there is no more leaf, we obtain the graph containing cycles and the nodes in acyclic bridges only. This means no more vertex with degree 1 remains in the reduced graph.

2. Let C_i be a cycle-leave with an appended vertex $v_i \in C_i$. The empty constraint set on C_i plus (v_i, v_i) is compatible, thus apply C_2^d to remove $C_i \backslash \{v_i\}$ and add constraint (v_i, v_i) into the constraint set of the graph if it does not exist. Note that if the constraint set on C_i is not empty, C_i is obtained by removing some cycles and the added constraints are only self-constraints on appended vertices. Thus, the constraint set on C_i plus (v_i, v_i) is still compatible and we can apply C_2^d on C_i.

3. Let C_j be one of the cycles that is acyclic-connected to C_i with a vertex v_j as appended vertex. Call the subset of unique path from v_i to v_j as P_{v_i,v_j}, we now have $deg(v_i) = 1$. Let v_i' be the first vertex in P_{v_i,v_j} from v_i. Apply T_2^d on v_i, remove v_i and add constraint (v_i', v_i'). Repeat the process of applying T_2^d for the next vertex in the path, which results in two cases. The first case is we reach a vertex in the path that has a neighbor $u \in P_{v_i,v_j}$ that belongs to some other paths between C_i (or C_j) and another cycle, then we remove that vertex and add constraint (u, u) if it does not exist and stop. The second case is we can repeat the process until reaching v_j, which means these vertices belong to only one path between C_i and C_j. Then we can remove all vertices in the set $P_{v_i,v_j} \backslash \{v_j\}$ and add the constraint (v_j, v_j) into constraint set of the graph and into the constraint set of C_j if it does not exist.

4. By **Lemma 4.8**, there are at least 2 cycle-leaves if the number of cycles is larger than 1. Thus, we can always repeat **step 2** and **step 3** since the constraint sets on cycle-leaves are always compatible by assumption until there is one cycle left.

5. The last cycle also contains added constraints that are self-constraints on some appended vertices, thus its constraint set is compatible. We apply C_1^d to reduce the cycle into a single vertex, then apply E_1^d to reduce it into the empty graph. we conclude that $gon(T_{m,n}) = 2$.

□

Example 4.12. Consider a tree with cycles graph $T_{30,4}$ with constraint set $\mathscr{C}$. $T_{30,4}$ consists of four cycles C_4, C_5, C_6 and C_8 with their cycle constraints as $\mathscr{C}_{C_4}$, $\mathscr{C}_{C_5}$, $\mathscr{C}_{C_6}$ and $\mathscr{C}_{C_8}$ respectively. The graph is shown as following:

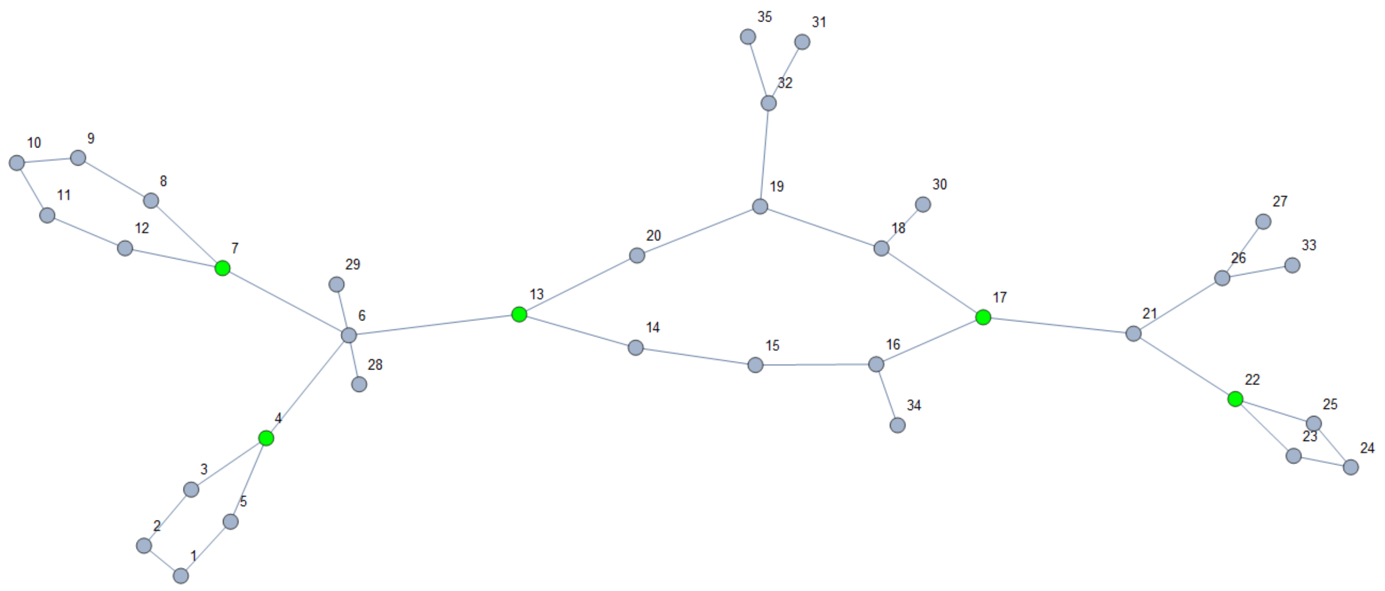


Figure 4.11: Reduction rules on a $T_{30,4}$

We begin the process by removing all the leaves:

- Apply T_1^d for the nodes: 27, 28, 29, 30, 31, 33, 34, 35, we obtain the first reduced graph with two new leaves: 26, 32.
- Apply T_1^d for the leaves on T_1: 26, 32, we obtain new reduced graph that contains no leaf.
- Take the cycle-leaf C_4 with appended vertex 22, empty set $\mathscr{C}_{C_4}$ plus constraint $(22, 22)$ is compatible, remove $C_{1,4}$ except vertex 22, add constraint $(22, 22)$ to $\mathscr{C}$. We obtain reduced graph with three cycles left.
- Apply T_2^d to 22 then to 21 to remove these two vertices and add constraint $(17, 17)$ in $\mathscr{C}$ and $\mathscr{C}_{C_8}$ (second case of **Step 3** in proof for **theorem** 4.9. Now C_8 is a cycle-leaf with one remaining appended vertex 13. We have $\mathscr{C}_{C_4} = \{(17, 17)\}$. Since there exists to equal from 17 to 13, by **lemma** 4.8, $(17, 17)$ and $(13, 13)$ are compatible. Thus apply C_d^2 to remove C_d^2 except vertex 13, add constraint $(13, 13)$ to $\mathscr{C}$.
- Apply T_2^d to vertex 13, add constraint $(6, 6)$ and stop since $deg(6) > 1$ (case 1 of **Step 3** in theorem 4.9), we obtain the reduced graph whose vertices are a part of a cycle except vertex 6. Take cycle $C_{2,5}$ with appended vertex 4. The empty set $\mathscr{C}_{C_5}$ plus $(4, 4)$ is compatible, thus apply C_d^2 to remove C_5 and add constraint $(4, 4)$ to $\mathscr{C}$. Then apply T_d^2 on vertex 4, remove vertex 4 and do not add constraint $(6, 6)$ since it already exists. Continue applying T_2^d on 6 and add constraint $(7, 7)$ on $\mathscr{C}$ and $\mathscr{C}_{C_6}$ (case 2 of **Step 3** in theorem 4.9). We obtain the reduced graph exactly as C_6.
- Consider cycle C_6 with constraint set $\mathscr{C}_{C_6}$ that is compatible, replace C_6 with a single vertex by rule C_1^d. We can finally apply E_1^d and obtain an empty graph. Thus, we claim $gon(T_{30,4}) = 2$.

Chapter 5

Conclusion

In the thesis, we calculated the gonality of *fan graphs*, *flower graphs* and their extensions by using the *reachability* to a vertex of a divisor.

In addition, we introduced *tree with cycles* graph, and constructed an upper bound for this graph in general. Finally, we provided a new approach to check the compatibility of the constraint set on a cycle and used *reduction rules* to characterize the special cases that the gonality of *tree with cycles* is 2. Algorithms for the firing patterns on a tree and a cycle were shown explicitly. An algorithm to apply *Reduction rules* in a specific order to check whether a *tree with cycles* graph has gonality of 2 was given.

Question And Possible Future Work

A question that remains in the thesis is whether given the sufficient condition that two constraint sets on a cycle are compatible, it is possible to conduct two paths in **Theorem 4.9**. On intuitive level, it seems that the condition holds. However, due to insufficient proofs and lack of time, we keep the question for our interest in future research. If the condition can be proven, the termination condition for algorithm to check gonality of a tree with cycles graphs can be greatly enhanced.

Another potential work for the future is to investigate whether it is possible to enhance the algorithm to extract all cycles and their subset of vertices given some graphs with special properties. Since finding a Hamiltonian cycle is known to be NP-complete [8], it is nearly impossible to extract all the cycles given a general graph. The only solution is to use Brute-force to search for all combinations of subsets of vertices, which is extremely time-consuming for large graphs.

Finally, we can research on some techniques to calculate the lower bound for gonality of flower graphs, and see if we can also restrict the upper bound for their gonality in the future.

Bibliography

[1] P. W. Shor A. Björner L. Lovász. “Chip-firing games on graphs”. In: (1991).

[2] Jelco M. Bodewes. “Divisorial gonality of graphs”. Master Thesis. Utrecht University, 2017.

[3] J. Chen. *Dijkstra’s Shortest Path Algorithm.* 2003.

[4] J. van Dobben de Bruyn. “Reduced divisors and gonality in finite graphs”. Bachelor Thesis. Leiden University, 2012.

[5] D. Gijswijt. “Computing graph gonality is hard”. In: (2015), arXiv:1504.06713.

[6] D. Holmes K. Arnold J. Gosling. *The Java Programming Language.* 3rd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN: 0201704331.

[7] N. G. David K. Kavitha. “Dominator Coloring Of Some Classes Of Graphs”. In: *International Journal of Mathematical Archive* (Nov. 2012), pp. 3954–3957.

[8] D. Johnson M. Garey. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979.

[9] Madhusudan Manjunath. “The rank of a divisor on a finite graph: geometry and computation”. In: *arXiv e-prints* (Nov. 2011), arXiv:1111.7251.

[10] E. Mphako-Banda. “Some polynomials of flower graphs”. In: *International Mathematical Forum* 2 (Jan. 2007), pp. 2511–2518.

[11] Kenneth H. Rosen. *Discrete Mathematics and Its Applications.* 7rd. Raghothaman Srinivasan, 2011.

[12] J. Spencer. “Balancing vectors in the max norm”. In: *Combinatorica* 6.1 (Mar. 1986), pp. 55–65. ISSN: 0209-9683. DOI: 10.1007/BF02579409.

Appendix

Algorithm

This chapter provides pseudo codes of some algorithms using Java language that I have personally worked on during my thesis. Since the original codes have various test cases and quite lengthy to put into the thesis, some important parts of the codes are extracted as pseudo codes for this section. A graph G is presented as a class *Graph*, which consists of two classes *Edge* and *Node* which represents set E and V respectively. Class *Edge* contains a multi-map (duplicate keys, values) that stores (node, node) as an edge in G. Apart from the basic attributes (degrees, node id, etc.), each node (vertex) has four main special attributes:

- *isCyclic* (boolean): return *true* if the node is a part of a cycle.
- *chip* (integer): number of chips assigned for the node.
- *isVisited* (boolean): to check whether a vertex is visited or not. This attribute is used for **Depth First Search (DFS)** algorithm to decide whether a graph contains a cycle or not.
- *lastVertexCycle* (string): Whenever **DFS** detects a cycle by trying to put a visited vertex v in the *stack*, the visiting node is marked as *isCyclic* = ***true***, and the *id* of v is stored in this attribute.

Each of the attribute has a *getter* and a *setter* methods, which are used to get the value of the node or assign a new value of the attribute to the node. A graph G can remove vertex by using an *id* of a node and remove edge by using an *id* or input of two nodes. We assume the functions like *getPath()* and *getShortestPath()* are used to retrieve the subset of nodes in the shortest path between a source node and a target node. Function *getSecondShortestPath()* is to take the subset of the second shortest path from the source to target node. These functions can be implemented using **Dijkstra Algorithm**[3].

Moving chip in tree

The general idea is we find a unique path from u to v. From each node in the path, we *clone* the original graph, and perform removing some vertices to obtain a sub-graph which serves as the firing set in the order from u to v. As Java is an object-oriented programming language [6], cloning a graph means instantiating a completely new object instead of using reference, thus cloning takes $O(V + E)$ as complexity.

```
moveChipInTree(Graph G, Node source, Node target)
    //Input: a tree graph, source and target vertices
    //Output: return a firing set if it is moved successfully to target node,
    return NULL otherwise.
    //Each node/edge has an id, thus the cloned elements also have the same
    id as a string.
    sourceId = source.getId();
    targetId = target.getId();
    List<Node> pathNodes = getPath(G, sourceId,targetId); //Path by ids
    List<Node> firingSet;

    For node in G:
        node.setVisit(false);
    Endfor

    For node in pathNodes
        If node = target:
            break;
        Endif
        nodeId = node.getId();
        Graph clonedG = cloneGraph(G) ;
        node.setVisit(true);
        If node is in pathNodes and is adjacent to target:
            clonedG.removeNode(targetId);
        Else:
            List<Node> neighbors = getNeighbors(G,node);
            For nNode in neighbors:
                If nNode is in pathNodes:
                    clonedG.removeNode(nNode.getId());
            Endfor
        Endif

        For cNode in clonedG:
            path = getPath(clonedG,cNode.getId,nodeId);
            If path!=NULL:
                firingSet.add(G.getNode(cNode.getId());
            Endif
        Endfor

        fireSubset(firingSet); //Fire vertices
    Endfor
```

Moving chips in cycle

We conduct an algorithm that must begin with a divisor D such that $D(v) = D(w)$ for v and w in the cycle and $v \neq w$. Firstly, we find the shortest path from v and w to the target vertex and choose it as the **main path**. Then a **sub-path** is created with either v or w that has the longer path to the target vertex. We reduce the size of this path by remove some vertices starting from the end of the path until the two paths have the same length. We then start conducting sub-graph and move chip along the two arcs of the cycles. A chip from v or w (or both) will end up in the target vertex.

```
moveChipsInCycle(Graph G, Node source1, Node source2, Node target)
    //Input: a cycle graph, two source and target vertices
    //Output: return a firing set if it is moved successfully to target node,
    return NULL otherwise.
    //Each node/edge has an id, thus the cloned elements also have the same
    source1Id = source1.getId();
    source2Id = source2.getId();
    targetId = target.getId();
    List<Node> firingSet;
    List<Node> mainPath = $getShortestPath(G, source1, target)$
    List<Node> subPath = $getShortestPath(G, source2, target)$;
    If mainPath.size() > subPath.size():
        $swap(mainPath, subPath)$;
    Endif
    Graph clonedG = $clone(G)$ //Start making sub-path
    String firstVertexSubpathId;
    If mainPath.get(0) = source1:
        clonedG.remove(source1Id);
        firstVertexSubpathId = source2Id;
    Else:
        clonedG.remove(source2Id);
        firstVertexSubpathId = source1Id;
    Endif
    List<Node> clonedSubPath = $getPath(clonedG,\ firstVertexSubpathId,\ targetId)$;

    For ($i =$ clonedSubPath.size()-1; $i >$ mainPath.size(); $i--$):
        clonedSubPath.remove(i); //Remove by index
    Endfor

    subPath.clear(); red//Remove old nodes in sub-path
    For each node in clonedSubPath:
        Add node into subPath;
    Endfor

    If mainPath.size() == 2:
        Graph clonedG = $clone(G)$; //Clone a completely new graph
        Remove last nodes of mainPath and subPath;

        For each cNode in clonedG:
            cNodeId = cNode.getId();
            If cNode can reach node with source1Id in clonedG:
                firingSet.add(G.getNode(cNodeId));
            Endif
        Endfor

        fireSubset(firingSet); //Fire vertices
    Else:
        For ($i = 0$; $i$ <mainPath.size()-1;$i++$):
            Graph cloneG = $clone(G)$;
            If $i$ <mainPath.size()-2:
                clonedG.remove(mainPath.get($i+1$).getId();
                clonedG.remove(subPath.get($i+1$).getId();
                For each cNode in clonedG:
                    If cNode cannot reach targetId in clonedG and node
                    with cNode.getId() in G is not in the firingSet:
                        firingSet.add(cNode);
                    Endfor
            Else:
                firingSet.add(mainPath.get(i));
                firingSet.add(subPath.get(i));
            Endif
    Endif
```

Reduction rules on trees with cycles

We modify iterative **Depth First Search** algorithm slightly to fit in our case:

```
DFS(Graph G, Node currentNode)
    //Input: any graph G and a beginning node to start DFS. However, a tree
    with nodes as cycles graph as input will maximize the potential of this
    algorithm.
    //Output: nodes are marked as visited, cycles are detected and some nodes
    whose attribute isCyclic and lastVertexCycle is modified.
    Initialize an empty stack;
    While stack is not empty:
        Node visitingNode = stack.peek();
        stack.pop();
        If visitingNode.isVisited = false:
            visitingNode.isVisited = true;
        Endif
        List<Node> neighborNodes = getNeighborNodes(G,visitingNode);

        For each neighbor in neighborNodes:
            If neighbor is not visited:
                If neighbor is in stack:
                    visitingNode.isCyclic = true;
                    visitingNode.lastVertexCycle = neighbor.getId();
                Else:
                    stack.push(neighbor);
                Endif
            Endif
        Endfor
```

We continue conducting algorithm to extract cycles of *tree with cycles as nodes* graph:

```
getCycles(Graph G, Node inputNode)
    //Input: a tree graph with nodes as cycle graph and a node
    to perform DFS.
    //Output: list of cycles with index in G.
    $DFS$(G, inputNode);
    Initialize Map<Integer, List<Node>> cycleMap;
    Initialize List<Node> cycleNodes, cyclicNodes;

    For each node in G:
        If node.isCyclic = true:
            cyclicNodes.add(node);
        Endif
    Endfor

    int iterator = 0;
    For each cNode in cyclicNodes:
        Graph clonedG = $clone(G)$;
        cNodeId = cNode.getId();
        lastId = cNode.lastVertexCycle;
        clonedG.removeEdge(cNodeId, lastId);
        List<Node> path = $getShortestPath$(clonedG, cNodeId, lastId);

        For each node in path:
            nodeId = node.getId();
            Node oriNode = G.getNode(nodeId);
            cycleNodes.add(oriNode);
            oriNode.isCyclic = true;
        Endfor

        cycleMap.put(iterator, cycleNodes);
        iterator++;
    Endfor
    return cycleMap;
```

Now we have extracted all cycles such that the attribute *isCyclic* of their vertices are *true*. The number of cycles are determined with the number of entries in *cycleMap*. The main part of checking gonality is illustrated:

```
$gonalityCheckForTreeWithCyclesAsNodes$(Graph G)
    //Input: a tree with cycles as nodes graph.
    //Output: returns true if it G is reduced to empty graph and
    false otherwise.
    Node node = G.getNode(0); //Get node by index
    Map<Integer, List<Node> cycleMap = getCycles(G, node);
    While G is not empty:
        If G has only one vertex:
            Apply $E_1^d$ on the vertex;
            break;
        Endif
        While G has at least a leaf:
            Select a leaf $v$;
            If constraint $(v, v)$ does not exist:
                Apply rule $T_1^d$ on $v$;
            Else:
                Apply rule $T_2^d$ on $v$; //node in cycle bridge after
                                        removing a cycle
        Endwhile
        int n = cycleMap.size();
        If n == 1:
            If the set of constraint on the last cycle is compatible:
                Apply $D_1^d$ on cycle;
            Endif
        Else:
            For($i = 0$;  $i < n$;  $i++$):
                If cycle cycleMap.get(i) has only one vertex $v$
                with $deg(v) > 2$:
                    If cycle constraint set on the cycle and $(v, v)$
                    is compatible:
                        remove cycle, add $(v, v)$ if it does not exist;
                        break;
                    Endif
                Endif
                If $i = n-1$ and $C_2^d$ cannot be applied on cycle cycleMap.get(i):
                    return false;
                Endif
            Endfor
        Endif
    Endwhile
    return true;
```

One big enhancement for this algorithm is that we can always conduct a method to check the compatibility of a constraint set on a cycle. **Theorem 4.9** is just a sufficient condition, so it only implies that if we can find such paths, the two sets are compatible, but not the way around. We decided to put the idea of proving the necessary condition for the theorem in the future work due to lack of time and proofs.