Vietnamese - German University
Department of Computer Science

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering

Bachelor Thesis
towards partial fulfillment of the
Degree of Bachelor Computer Science

Critical configurations of Chip Firing Game on graphs

presented and publicly submitted by

Nguyen, Phuoc Bao Hoang
Matriculation number: 1148181

September 17, 2018
before a jury composed of:

Dr. Tran, Thi Thu Huong	Assessor
Dr. Nguyen, Hoang Thach	Co-Assessor

Declaration

I hereby declare that this thesis, which is submitted to Vietnamese-German University for the degree of Bachelor of Computer Science is my own work, unless otherwise referenced. This thesis is finished under the guidance and supervision of Dr. Tran Thi Thu Huong, lecturer of Vietnamese – German University and Dr. Nguyen Hoang Thach, researcher of Vietnam Academy of Science and Technology - Institute of Mathematics.

Also, I declare that I have not obtained any degree in Vietnamese – German University or elsewhere on the basis of this work.

Nguyen Phuoc Bao Hoang
Computer Science Student
Vietnamese German University
Date: October 28, 2019 Signature: ____________________

Acknowledgements

First of all, I would like to express my deep appreciation to my assessor Dr. Tran Thi Thu Huong for her continuous support. Throughout my thesis study, her guidance kept me on the right track of researching and writing. I could not have imagined having a better advisor in this particular subject.

Besides my assessor, I would like to express my sincere gratitude to my co-assessor Dr. Nguyen Hoang Thach for his support in formatting the thesis. As one of the professional researcher about the same field that my thesis have been written, I believe I can improve a lot throughout your insightful comments and questions.

I would also like to thank my fellow classmates of the computer science, class of 2014, for their suggestion and assistance. My thanks also goes to my colleagues who would always empathize for my circumstances and help me to handle my works so that I can spend more time writing the thesis.

Last but not the least, I would love to thank my parents and my brother for their spiritual support throughout the time.

Contents

List of Figures

List of Tables and Algorithms

Abstract

Many problems related to the dynamical system have caught the researchers' attention for years. One of them, the well-known Chip Firing Game Model is also taken under the consideration. Over the past three decades, positive results of this model have been conducted, especially under the undirected graph. This thesis tries to continue expanding that results.

In particular, Chip Firing Game (CFG) on an undirected graph contains a special set of graph states, called the sandpile group of this graph. Under the specific operation, this sandpile group is abelian and hence contains an identity element. This element has been studied in many other papers [2, 5, 6].

Several results have solid proofs but others remain unproven. As a consequence, this thesis will take advantages of the results from the above state-of-the-art papers and establish the relationship between this identity element with some other basic properties of this model with the hope that it can produce some fresh perspectives on the model. Furthermore, this thesis can be useful for students who are passionate in mathematics and want to learn more about the CFG.

To begin with, this thesis introduces the CFG in a top-down approach. After that, it provides the definitions and terminologies that are related, including the preliminaries of graph theory. Then, some examples during the thesis studies will be presented, including the identity configuration computing on some particular classes of graph such as flower, wheel, cycle and complete graphs. Based on the result, we continue to establish our proof about the identity configuration in those graphs. At last, some conclusions and future works will be listed.

Keywords
Identity configuration
Maximal stable configuration
Non-trivial natural solution

Chapter 1

Introduction

Discrete Dynamical System

“If one knows the fish population this year and the advance years, will he be able to deduce the population for the following ones?”

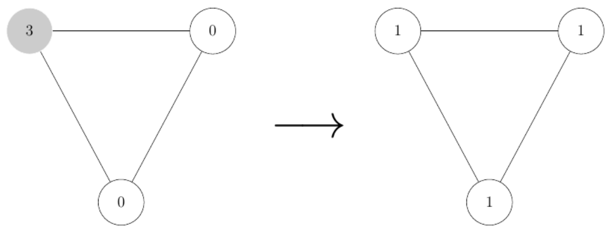
In many scenarios in physics, economics, chemistry or biology such as the example above, people care about whether will they be able to determine the outcomes (e.g. population of the fish) based on its previous results (e.g. population of the fish during last year). Such situations are modelled in, what is called, the dynamical system. More often than not, these systems are determined by time, state (e.g. the number of fish) and evolution rule (e.g. the breeding rate of fish).

To go further, a dynamical system whose time factor is measured in real numbers is a discrete one. In other way, this system use discrete time steps. Therefore, system of this kind can sometimes be modelled by two the remaining factors, namely the set of system states and the set of evolution rules (rules, in short) for transferring between states. In summary, this field of science studies the phenomenon in reality where changes can be performed by some specific, time-independent rules.

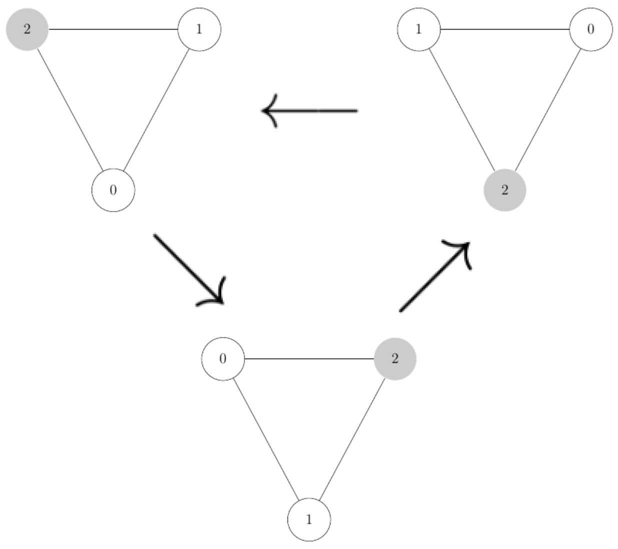
There exists many models of this kind, the sand-pile model for instance. However in this thesis, we will take the Chip Firing Game Model, one of the well-known examples of Discrete Dynamic System, as our main theme.

Chip Firing Game

In the simplest way, the model can be defined as a simple connected graph where every vertex consists an integer value representing a number of chips. Each state (also known as configuration) of the graph can be defined by observing the number of chips of every single vertex. The rule of CFG, applies for the vertex that has the number of chips larger than the number of its neighbors. Such a vertex exists, it will send a chip to all of its neighbors. The game will end when there is no vertex that can satisfy the rule. Notice that there exists plenty of scenarios where the game will never finish, for example, if the total number of chips is exceeded the total number of vertices' degree.



(a) Finite CFG



(b) Infinite CFG

Figure 1.1: The firing vertex will have a darker color than the others. In 1.1a, the graph will only fires one time and leads to the final configuration where every vertex get one chip, while in 1.1b is an infinite game

Chip Firing Game was first introduced in around 1990 by Lovász et. al. [3], Per Bak et. al. [1] and Dhar [5]. For instance, Lovász and the others worked on the game finality and the number of steps the game would take. After the introduction, the study of this field continued to expand in terms of generalizations, state characterizations or relationships with others related aspects like spanning trees, parking functions or special polynomials. One remarkable work derived from the above framework is the introduction of abelian sandpile group from Dhar [5].

Configuration types and the Sandpile Group

Firstly, there exists many types of configuration in CFG, namely:

- Stable configuration is the one in which exists no fire-able vertex.
- Accessible configuration is the one from which any other configurations can reach, given a selective adding and then selective firing chips mechanism.
- Recurrent configuration must be both stable and accessible. In some other context, recurrent configuration is also known as critical configuration [2]. For the sake of simplicity, we will use recurrent configuration as the official term for this class of configuration in this thesis.

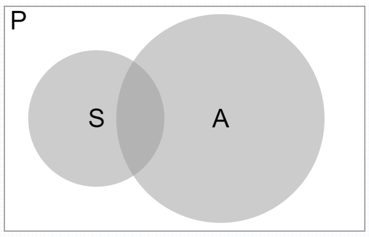


Figure 1.2: Configuration venn diagram

In the diagram shown in Figure 1.2, P is the configuration sample space of the specific graph. Consecutively, S and A are the set of stable and accessible configurations. From the definition above, the recurrent configuration set is the interception between S and A. It is also worth notice that there exists configurations that are neither stable nor accessible.

Dhar denoted the set of all recurrent configurations of a graph G as a sandpile group of G and provided a testing mechanism for recurrent configuration [5]. Later on, R. Cori and D. Rossin proved that this is, in fact, an abelian group under specific adding-and-firing operation [4]. This sandpile group is also mentioned in the work of Levine and the others[6]. From that, we are interested on a specific element of this group - the identity element.

Identity element

From the rudimentary algebra, the identity element of one set under one binary operation is the element which remains the other elements when combine altogether using this binary operation.

Example 1.1. *Given the set of real number $\mathbb{R}$, under the adding and multiplying operation consecutively, the identity element are 0 and 1.*

$$I(\mathbb{R}, +) = 0 \textit{ and } I(\mathbb{R}^*, *) = 1$$

Levine et. al. stated that there exists an identity element given the sandpile group of graph G under the special adding-firing operator, denote as identity configuration [6]. They also provided one way to find this configuration.

Continue the work of the mentioned paper, we will study the relationships between the identity configuration with some other properties of CFG. For convention, in this thesis, we only consider the simple connected graph in which we make sure that the game will always be finite. In addition, we will use the term configuration and firing rule as a representation for some CFG general terms: state and evolution rule.

Thesis Outline

Our main purpose is to investigate the identity element in the sandpile group of simple connected graph. As a result, in this thesis, we will first implement an application to compute the identity configuration. This program can perform various calculations, including finding the identity configuration of specific simple graphs using two different algorithms given by Dhar[5] and Levine et. al.[6]. Moreover, it can also check whether a configuration is recurrent or not. The results can also be exported under either text or LaTeX format.

Secondly, we use the application to perform some experiments on special classes of simple graph, such as flower graphs, cycles, wheel graphs or complete graphs. From the experimental results, we conduct an overview and establish a relationship between the identity configuration and maximal stable configuration in a general way. Precisely, we compute explicitly the identity configuration of some classes of graph listed above.

The thesis is organized as follows.

In chapter 2, fundamental definitions about graph theory and chip firing game that related to the problem will be presented.

In chapter 3, we introduced the firing simulator we have created to support the thesis study. After that, we conducted some experiments with specific classes of graph. In the end, we provide and prove the general observation from the results we yields above.

Chapter 4 presents the thesis conclusions and some open questions that we cannot answer and leave it as the future work.

Chapter 2

Background on graph theory and chip firing game

In this chapter, we begin with some basic definitions of graph theory and some useful representations along with remarks. Then, we describe the chip firing game and its operations. After that, we focus on defining the recurrent and identity configuration, we also introduce some algorithms to check and to calculate it.

2.1 Graph theory - Definitions and fundamental concepts

We begin with the standard definition of graph, taken from [7]:

Definition 2.1 (**Definition of Graph** [7])**.** *A graph $G = (V, E)$ consists of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.*

Theoretically, a graph can be defined by two factors - the vertices and the edges connecting those vertices. Using the notation $G = (V, E)$, the graph G is formed from two sets V and E. Assuming there are n vertices and m edges, while $V = \{v_1, v_2, ..., v_n\}$ is the set of all vertices, $E = \{e_1, e_2, ..., e_m\}$ is the set of edges representing the association of V. Since we only consider simple graphs in this thesis, we extend the definition of the above:

Definition 2.2 (**Simple Graph** [7])**.** *A graph in which each edge connects two different vertices and no two edges connect the same pair of vertices is called a simple graph.*

There exists one or no association between two distinct vertices. Moreover, there is no association between a vertex and itself. In graph theory, there are many special classes of simple graph and they have been used for conducting experiments in many other applications. Let us introduce some of the graphs we will use in our thesis:

Example 2.1. ***(Flower graph)*** *A flower graph of n, denoted by F_n, is a simple graph which has one vertex at the center and the others will distinctively be grouped into group of two and they, together with the common center vertex, create a 3-cycle sub-graph. Each of the 3-cycle graph is called a petal of F_n.*

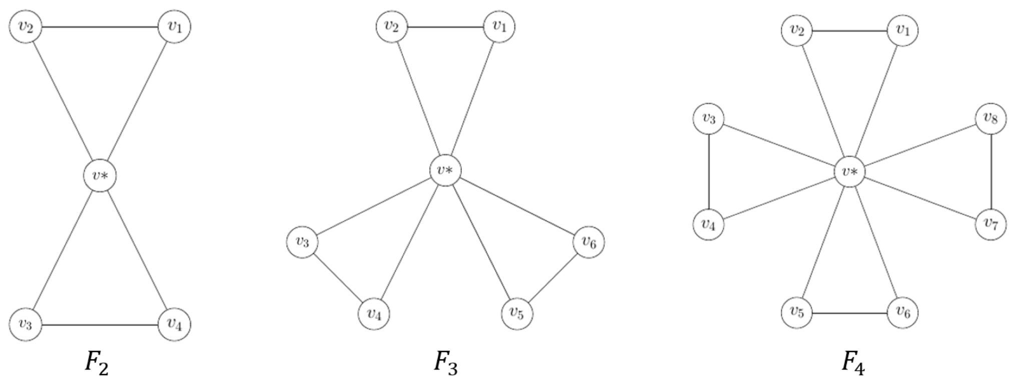


Figure 2.1: Flower graph of n with n = 2, 3, 4

Example 2.2. ***(Cycle graph [7])*** *A cycle C_n, $n \geq 3$, consists of n vertices v_1, v_2, ..., v_n and edges (v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n), and (v_n, v_1).*

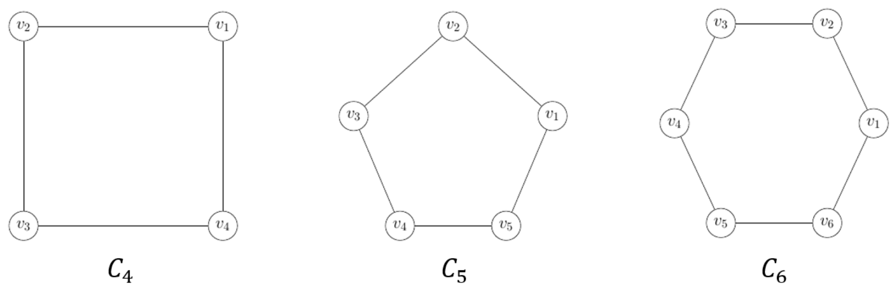


Figure 2.2: Cycles C_4, C_5 and C_6

Example 2.3. ***(Wheel graph [7])*** *A wheel graph W_n is a cycle graph C_n with an additional common vertex that associates with every other vertices.*

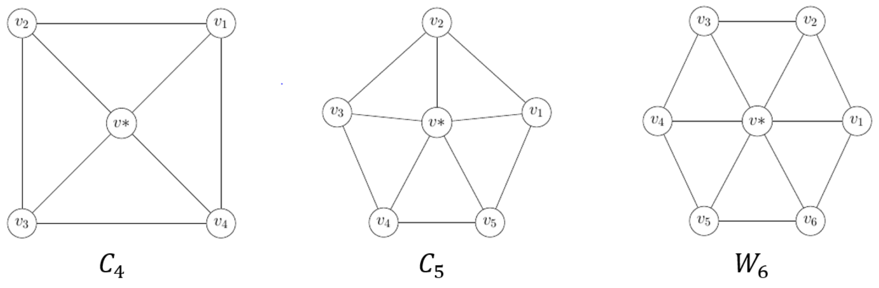


Figure 2.3: Some examples of wheel graph

Example 2.4. ***(Complete graph [7])*** *A complete graph on n vertices, denoted by K_n, is a simple graph that contains exactly one edge between each pair of distinct vertices.*

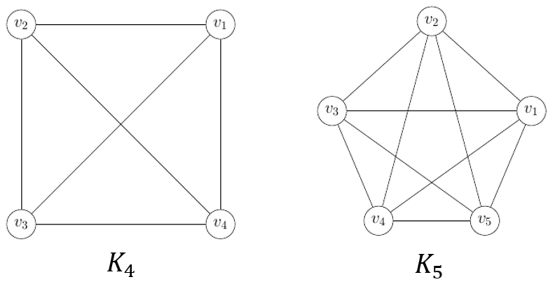


Figure 2.4: Complete graph of 4 and 5 vertices

Since we are going to perform some experiments of the chip firing game on some of these special graphs, further properties of these graphs will be defined in the later section. For the purpose of this chapter, let us take a random simple connected graph as an example:

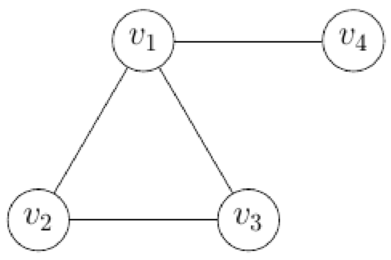


Figure 2.5: Example of simple graph

Example 2.5. *Consider the simple graph in Figure 2.5, we have* $V = \{v_1, v_2, v_3, v_4\}$ *and* $E = \{(v_1, v_4), (v_1, v_2), (v_1, v_3), (v_2, v_3)\}$.

In chip firing game model, whenever a vertex is fired, it is crucial to know to which vertices this vertex is associated with. Thus, we come to the neighbor terminology:

Definition 2.3 (**Neighbors of a vertex**). *Two vertices are neighbors in* G *if there exists an edge in* E *such that two vertices are the endpoints of it. The set of all neighbors of a vertex* u*, denoted by* $N(u)$*, is called the neighborhood of* u.

Example 2.6. *As the above example shown, in* E *there are three edges that have* v_1 *as one of the endpoints, namely* (v_1, v_4), (v_1, v_2) *and* (v_1, v_3). *Thus, vertex* v_1 *has* v_2, v_3 *and* v_4 *as its neighbors, which is also equivalent to* $N(v_1) = \{v_2, v_3, v_4\}$.

Needless to say it has no-use to consider the simple graph in which exists a vertex that has no neighbor. Therefore, our considered graphs will always be connected.

Remark 2.1. *A simple connected graph is a simple graph that every vertex has at least one neighbor.*

To keep track of how many neighbors that one vertex may have, the following definition about the degree of a vertex is made:

Definition 2.4 (**Degree of a vertex**)**.** *In simple graph, the degree of a vertex is the number of edges incident with it. The degree of a vertex v is denoted by $deg(v)$.*

Example 2.7. *For the graph given in figure 2.5, we have $deg(v_1) = 3$, $deg(v_2) = 2$, $deg(v_3) = 2$ and $deg(v_4) = 1$.*

From this, we easily arrived to the relationship between the neighborhood of a vertex with its degree:

Remark 2.2. *The degree of a vertex v is the cardinality of its neighborhood: $deg(v) = |N(v)|$.*

There exists many other representations of graph and the most convenient one will be chosen depend on the specific situation. Let us introduce some of the representation that will be involved during our work:

Definition 2.5 (**Degree Matrix**)**.** *A degree matrix of a simple connected graph is the $n \times n$ diagonal matrix which shows the degree of the vertices. It is denoted by D.*

Example 2.8. *For the graph given in figure 2.5, the degree matrix:*

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since the matrix above only contains the information about the degree of vertices, we want to introduce one matrix that represent another property of the graph - the association between vertices:

Definition 2.6 (**Adjacency Matrix**)**.** *An adjacency matrix of a simple connected graph, denoted by A, is the $n \times n$ symmetric matrix which contains the information about the association between vertices.*

Example 2.9. *For the graph given in figure 2.5, the adjacency matrix:*

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

As the example shown, the adjacency matrix is a symmetric binary $n \times n$ matrix where:

$$a_{ij} = \begin{cases} 1 \text{ if there exists an association between } v_i \text{ and } v_j \\ 0 \text{ otherwise} \end{cases}$$

Remark 2.3. *The i-th value of the degree matrix is equal to the i-th row sum of adjacency matrix:*

$$\forall v_i \in V : d_i = \sum_{j=1}^{n} a_{ij}$$

Indeed, we can easily derive the above result since the degree of a vertex is equal to the number of the vertices that it is associated with.

We introduced the adjacency matrix A and the degree matrix D. Last but not the least, we calculated the Laplacian matrix Δ which is the difference between D and A. In fact, in simple graph we can define:

$$\Delta_{ij} = \begin{cases} d_i \text{ for i = j} \\ -a_{ij} \text{ for i} \neq \text{j} \end{cases}$$

It is worth notice that the i-th row of the laplacian matrix Δ is called the laplacian vector of the vertex v_i.

Example 2.10. *For the graph given in figure 2.5, the laplacian matrix:*

$$\Delta = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 2 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

Derived from remark 2.3, we yield the following remark:

Remark 2.4. *Every row sum of Laplacian matrix is 0.*

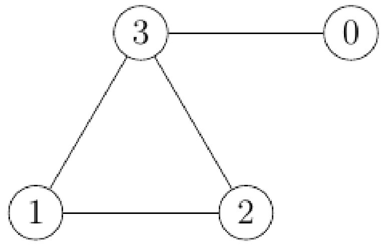


Figure 2.6: Example configuration of graph in Figure 2.5

2.2 Chip firing game

First of all, let us borrow some words from Biggs to define the Chip Firing Game:

Definition 2.7 (**Chip Firing Games [2]**). *A chip-firing game on a graph G starts with a pile of tokens (chips) at each vertex. At each step of the game a vertex v is 'fired', that is, chips move from v to the adjacent vertices, one chip going along each edge incident with v. A vertex v can be fired if and only if the number of chips currently held at v is at least $deg(v)$, the degree of v.*

From the definition of Biggs, Chip firing game can be defined under a pair of graph states C and the firing rule $\sim$, denoted: $CFG = (C, \sim)$. We will define the graph state in advance:

Definition 2.8 (**Graph state**). *Given the graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, a state of graph G, usually known as a configuration, is the ordered n-vector which contains the number of chips of each vertex.*

Let us use c as the configuration representation, then c is the vector $(c_{v_1}, c_{v_2}, ..., c_{v_n})$ where c_{v_i} is the number of chips in vertex v_i.

Example 2.11. *In Figure 2.6, the configuration is $c = (3, 1, 2, 0)$.*

Since the number of chips is a non-negative integer, the set of all possible configurations of graph G can be presented as $C : V \longrightarrow \mathbb{N}^n$. From this, we define the firing rule:

Definition 2.9 (**The firing rule**). *The firing rule can be described as an unary operator on configuration set in which one configuration can be transform into another if it is fire-able.*

It leads us to define the fire-able predicate:

Definition 2.10 (Firing condition). *A configuration is fire-able when there exists at least one vertex that has the number of chips is greater than or equal to its degree.*

$$c \text{ is fire-able} \iff \exists v_i \in V : c_{v_i} \geq d_i$$

From the work of Björner and Lovász, we know that if the game is finite then there exists at least one vertex which is not fired at all [3]. From this proposition, we can easily derive that if the graph has at least one vertex that never fires then starting from any configuration, the game will always stop. Since we want our game to be always finite, we assign one vertex to be the vertex mentioned above. Therefore, we define the next definition:

Definition 2.11 (Sink of a graph). *In a simple connected graph, a vertex s is defined as a sink if it never gives out any chips to its neighbors.*

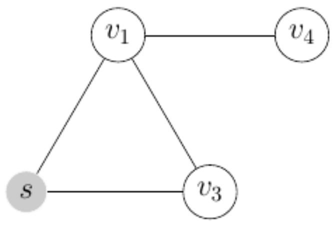


Figure 2.7: Example graph of Figure 2.5 with v_2 as the sink

Since this vertex only takes in but never gives out (see Figure 2.8), we ignore it in the matrix representation as well as the configuration.

Remark 2.5. *From now on, we will exclude the sink out of some representations, namely:*

- *The configuration c and its set $C : V \setminus s \longrightarrow \mathbb{N}^{n-1}$.*
- *Exclude its row and column out of the laplacian matrix Δ to produce the reduced laplacian matrix Δ'.*

Example 2.12. *Let us try to fire the configuration from Example 2.7:*

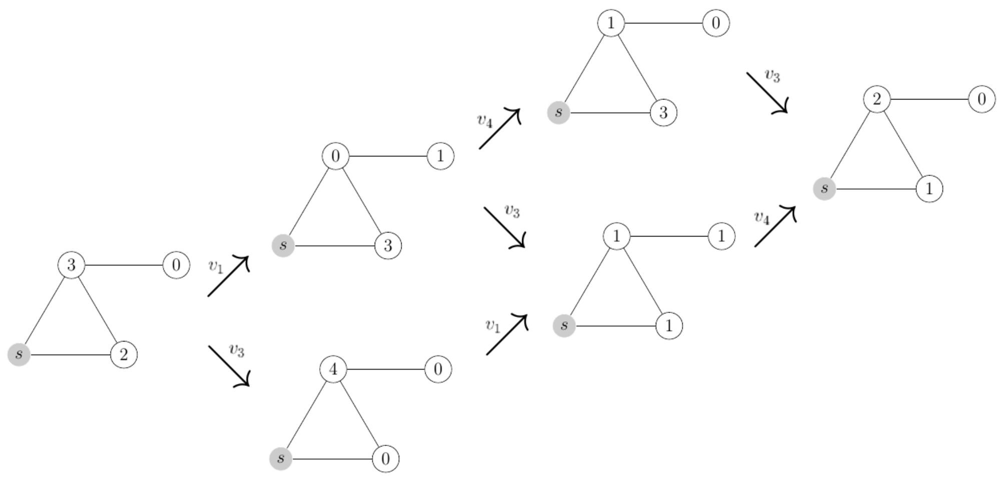


Figure 2.8: Firing procedure of 2.7 under vertex-edge representation

Example 2.13. *We exclude the sink from the configuration and the laplacian matrix:*

- *The reduced Laplacian Matrix:*

$$\Delta = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 2 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \longrightarrow \Delta' = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

- *The configuration vector after the exclusion:*

$$\begin{tikzcd}
 & & (1,3,0) \arrow[dr, "v_3"] & \\
 & (0,3,1) \arrow[ur, "v_4"] \arrow[dr, "v_3"] & & (2,1,0) \\
(3,2,0) \arrow[ur, "v_1"] \arrow[dr, "v_3"] & & (1,1,1) \arrow[ur, "v_4"] & \\
 & (4,0,0) \arrow[ur, "v_1"] & &
\end{tikzcd}$$

Figure 2.9: Firing procedure of 2.7 under configuration representation

From the sink properties, we know that the sink will never give out any chip. However, in some cases, it is useful if we intentionally fire the sink, forcing it to act like a normal vertex, sending its chips to its neighbor. This operation will be defined below:

Definition 2.12 (**Back-fire operation**). *The back-fire operation is the operation where we force the sink to fire once, sending one chip to each of its neighbors. The backfire operation is denoted by* $\rightsquigarrow$ *and can be used as* $u \rightsquigarrow \overline{u}$*, where back-fire the sink in* u *will yield* $\overline{u}$ *as the result.*

From now on, we do not care about the number of chips in the sink anymore. Notice that on configuration u whenever you firing the sink, sending chips to its neighbors; the result configuration is actually the difference after subtract u to the laplacian vector of the sink and ignore the results of the chips at the sink. Therefore, we want to introduce one more vector that will be helpful for us when we perform the back-fire operation: the reduced laplacian vector of the sink:

Definition 2.13 (**The reduced laplacian vector of the sink**). *Given the simple connected graph* G *with the sink* s*, the reduced laplacian vector of the sink is the laplacian vector of itself* Δ_s *but exclude the element at the sink position. The reduced laplacian of the sink is denoted by* Δ'_s

Example 2.14. *For graph in figure 2.6 with the sink* v_2*, the laplacian vector of the sink:*

$$\Delta_s = (-1, 2, -1, 0)$$

Since the sink is at the 2^{nd} *position, we exclude the* 2^{nd} *element of the laplacian vector of the sink. The result is the reduced laplacian vector of the sink:*

$$\Delta'_s = (-1, -1, 0)$$

We have a small remark about this reduced laplacian vector of the sink with the reduced laplacian matrix of the graph:

Remark 2.6. *Total row sum of the reduced laplacian matrix of the graph with the reduced laplacian vector of the sink is zero.*

$$\Delta'_1 + \Delta'_2 + ... + \Delta'_{n-1} + \Delta'_s = 0$$

We have defined the definition of CFG, including the configuration and the firing rule. In order to examine further on special configurations and the game properties, we need to introduce some operations that we will use during the thesis. First of all, let us introduce four basic operations:

Remark 2.7. *Given two configurations $u = (u_1, u_2, ..., u_{n-1})$ and $v = (v_1, v_2, ..., v_{n-1})$, here are some of the basic operations:*

- *Scalar multiplication: $k \in \mathbb{N}, ku = (ku_1, ku_2, ..., ku_{n-1})$.*
- *Scalar addition: For convention, we use the notation $k = (k, k, ..., k)$ for $k \in \mathbb{N}$ such that: $k + u = (k + u_1, k + u_2, ..., k + u_{n-1})$.*
- *Addition: $u + v = (u_1 + v_1, u_2 + v_2, ..., u_{n-1} + v_{n-1})$.*
- *Subtraction: $u - v = (u_1 - v_1, u_2 - v_2, ..., u_{n-1} - v_{n-1})$.*

Based on the basic operations, we can define some of important and more complex operations:

Definition 2.14 (Firing operation). *Firing operation is an unary operation of configuration. A configuration u fires the vertex v_i will result another configuration v, denote: $u \xrightarrow{v_i} v$.*

Notice that whenever a vertex is fired, it gives out the number of chips that is equals to its degree while each of the vertex neighbors will receive one chip.

Remark 2.8. *Firing the vertex v_i, we can calculate the result configuration by subtracting the current configuration with the i-th row of the Reduced Laplacian Matrix:*

$$u \xrightarrow{v_i} v : v = u - \Delta'_i$$

Example 2.15. *For the graph in figure 2.6 with the sink v_2, notice that from initial configuration $c_1 = (3, 2, 0)$, if we fire v_1 we obtain the new configuration c_2:*

$$\begin{aligned} c_1 \xrightarrow{v_1} c_2 : c_2 &= c_1 - \Delta'_1 \\ &= (3, 2, 0) - (3, -1, -1) \\ &= (0, 3, 1). \end{aligned}$$

Since we are always considering the graphs that has the closure under the firing operation, sometimes we only care about the initial configuration and the final configuration of the game. For that reason, let us denote $\xrightarrow{*}$ as the stabilizing operator. For example, $c \xrightarrow{*} c^o$ means that from the configuration c, by applying the firing rule until the game is end, one will achieve the final configuration c^o. From the remark 2.8, we derive the following remark:

Remark 2.9. *Given an initial configuration u, by applying the stabilizing operation, one obtain the final stable configuration, denoted by u^o. There will always exist at least one radix (n-1)-vector r that:*

$$u \xrightarrow{*} u^o : u^o = u - \sum_{i=1}^{n-1} r_i \Delta'_i$$

If we combine the above remark with the corollary from Dhar, where he stated that from any initial configuration, the result of stabilizing, if exists, can only be unique and every vertex will fire exactly the same time in any procedure [5]. From that, we extend our remark:

Remark 2.10. *Given an initial configuration u, by applying the stabilizing operation, one obtain the final stable configuration u^o. There will always exists exactly one unique radix (n-1)-vector r such that:*

$$u \xrightarrow{*} u^o : u^o = u - \sum_{i=1}^{n-1} r_i \Delta'_i$$

Example 2.16. *For the graph given in figure 2.6 with the sink v_2, from c_1 to $(c_1)^o$ the vertices v_1, v_3 and v_4 have fired exactly once thus radix vector $r = (1,1,1)$:*

$$\begin{aligned} c_1 \xrightarrow{*} (c_1)^o : (c_1)^o &= c_1 - (\Delta'_1 + \Delta'_3 + \Delta'_4) \\ &= (3,2,0) - ((3,-1,-1) + (-1,2,0) + (-1,0,1)) \\ &= (3,2,0) - (1,1,0) \\ &= (2,1,0) \end{aligned}$$

2.3 Recurrent and identity configuration

Recall from the introduction, let us define recurrent configuration:

Definition 2.15 (Recurrent Configuration). *Let G be a simple graph with a global sink s, a recurrent configuration σ on G must be both stable and accessible.*

Indeed, since stabilization always results an unique stable configuration, it is worth notice that the selective firing sequence will not be taken under consideration anymore. Taken from the work of Levine and the others, recurrent configuration also has this interesting property [6]:

Remark 2.11. *Let G be a simple graph with a global sink s and σ be the recurrent configuration on G, for any configuration ω, there exists a configuration β that $(\omega + \beta) \overset{*}{\longrightarrow} \sigma$.*

It is not difficult to know whether a configuration c is recurrent or not. From the work of Dhar, he provided an algorithm to test for this property.

Definition 2.16 (**Dhar burning algorithm**). *Given a configuration c in graph G, in order to test the recurrency properties of this, one can fire the sink of G one time, then stabilize it. If and only if the final configuration is indeed c, then c is recurrent.*

Example 2.17. *For the graph given in figure 2.6 with the sink v_2, notice that v_1, v_3 and v_4 cannot fire if the number of chips are $\{0, 1, 2\}$, $\{0, 1\}$ and $\{0\}$. From that, we can find the set of stable configurations, namely*

$$\{(2, 1, 0), (2, 0, 0), (1, 1, 0), (1, 0, 0), (0, 1, 0), (0, 0, 0)\}$$

We can find the sandpile group of G by applying the Dhar's burning algorithm to all the stable configurations:

- *Applying back-fire operation to $c_1 = (2, 1, 0)$:*

  $$c_1 \rightsquigarrow \overline{c_1} = (3, 2, 0)$$

  As Figure 2.8 shown, after the stabilization, the final configuration is indeed $(2, 1, 0)$. E.g.

  $$(3, 2, 0) \xrightarrow{v_3} (4, 0, 0) \xrightarrow{v_1} (1, 1, 1) \xrightarrow{v_4} (2, 1, 0)$$

  Since the stabilization yields c_1 as the final configuration, it is a recurrent configuration.

- *Applying back-fire operation to $c_2 = (2, 0, 0)$:*

  $$c_2 \rightsquigarrow \overline{c_2} = (3, 1, 0)$$

  We can stabilize configuration $(3, 1, 0)$ by firing sequentially v_1 then v_3 and v_4. E.g.

  $$(3, 1, 0) \xrightarrow{v_1} (0, 2, 1) \xrightarrow{v_3} (1, 0, 1) \xrightarrow{v_4} (2, 0, 0)$$

  Since the stabilization yields c_2 as the final configuration, it is a recurrent configuration.

- *Applying back-fire operation to* $c_3 = (1,1,0)$*:*

$$c_3 \rightsquigarrow \overline{c_3} = (2,2,0)$$

We stabilize configuration $(2,2,0)$ *by firing* v_3*,* v_1 *and then* v_4*. E.g.*

$$(2,2,0) \xrightarrow{v_3} (3,0,0) \xrightarrow{v_1} (0,1,1) \xrightarrow{v_4} (1,1,0)$$

Since the stabilization yields c_3 *as the final configuration, it is a recurrent configuration.*

- *Applying back-fire operation to* $c_4 = (1,0,0)$*:*

$$c_4 \rightsquigarrow \overline{c_4} = (2,1,0)$$

Since $(2,1,0)$ *is still a stable configuration, we cannot stabilize it. Thus,* $(1,0,0)$ *is not recurrent.*

- *Applying back-fire operation to* $c_5 = (0,1,0)$*:*

$$c_5 \rightsquigarrow \overline{c_5} = (1,2,0)$$

We stabilize configuration $(1,2,0)$ *by firing* v_3 *once. E.g.*

$$(1,2,0) \xrightarrow{v_3} (2,0,0)$$

Since $(2,0,0)$ *is stable but different from the configuration* c_5*. Thus,* $(0,1,0)$ *is not recurrent.*

- *Applying back-fire operation to* $c_6 = (0,0,0)$*:*

$$c_6 \rightsquigarrow \overline{c_6} = (1,1,0)$$

Since $(1,1,0)$ *is still a stable configuration, we cannot stabilize it. Thus,* $(0,0,0)$ *is not recurrent.*

Afterward, we obtain the set of recurrent configurations of C*:*

$$\{(2,1,0),(2,0,0),(1,1,0)\}$$

.

As the example shown, more often than not, there exists more than one recurrent configuration in C. Since we want to work on the whole set. Let us define:

Definition 2.17 (**Sandpile Group**). *Given the graph G, the set of all recurrent configurations is called the sand-pile group of G, denote as $S(G)$.*

There exists some interesting properties under this sand-pile group. For example, Levine et. al. have provided an interesting corollary stated that $S(G)$ is an abelian group under the combination of adding then stabilizing operation:

Remark 2.12. *The closure property of the sand-pile group:*

$$\forall u, v \in S(G) : (u+v)^o \in S(G)$$

Since the sand-pile group is abelian, it is worth to consider the identity element of this group. Thus, we have given the definition for the identity configuration:

Definition 2.18 (**Identity Configuration**). *A configuration I of the sand-pile group $S(G)$ is the identity if and only if it is recurrent and for any recurrent configuration $u \in S(G)$ we have:*

$$(I+u) \xrightarrow{*} u$$

Example 2.18. *For the graph given in figure 2.6 with the sink v_2, since we only have three elements in $S(G)$. We calculated and noticed that the configuration $(1,1,0)$ is indeed the identity configuration. Apply the adding-then-stabilize operation yields:*

- *Adding:*

  $$c = s_1 + I = (2,1,0) + (1,1,0) = (3,2,0)$$

  Stabilizing:

  $$(3,2,0) \xrightarrow{v_3} (4,0,0) \xrightarrow{v_1} (1,1,1) \xrightarrow{v_4} (2,1,0)$$

  Thus, $s_1 + I \xrightarrow{*} s_1$.

- *Adding:*

  $$c = s_2 + I = (2,0,0) + (1,1,0) = (3,1,0)$$

  Stabilizing:

  $$(3,1,0) \xrightarrow{v_1} (0,2,1) \xrightarrow{v_3} (1,0,1) \xrightarrow{v_4} (2,0,0)$$

  Thus, $s_2 + I \xrightarrow{*} s_2$.

- *Adding:*

$$c = s_3 + I = 2I = (1,1,0) + (1,1,0) = (2,2,0)$$

Stabilizing:

$$(2,2,0) \xrightarrow{v_3} (3,0,0) \xrightarrow{v_1} (0,1,1) \xrightarrow{v_4} (1,1,0)$$

Thus, $s_3 + I \overset{*}{\longrightarrow} s_3$*.*

Therefore $s_3 = (1,1,0)$ *is the identity configuration.*

However, it is not feasible if one try to find an identity element manually since the configuration set can become quite big given more complex graphs. Hence, we present two methods to find the identity element:

Definition 2.19 (**Extended Dhar's burning algorithm** [**5**])**.** *Starting from the zero configuration as an initial input configuration, one applies the burning algorithm and result an output stable configuration. One continues to apply the burning algorithm until the output is the same with the input. That repeated configuration is indeed the identity configuration.*

It will be further explained as we will talk about the algorithm under the pseudo-code in chapter 3. At the moment, let us take the example above and calculate its identity:

Example 2.19. *For the graph given in figure 2.6 with the sink* v_2 *with zero configuration* $c_0 = (0,0,0)$*. After the first burning algorithm:*

$$c_0 \rightsquigarrow \overline{c_0} = (1,1,0)$$

Since $\overline{c_0}$ *is stable, there is no need for a stabilization. Since* $\overline{c_0} \neq c_0$*, let us apply the algorithm again:*

$$\overline{c_0} \rightsquigarrow \overline{\overline{c_0}} = (2,2,0)$$

Since this yields an unstable configuration, we try to stabilize it:

$$(2,2,0) \xrightarrow{v_3} (3,0,0) \xrightarrow{v_1} (0,1,1) \xrightarrow{v_4} (1,1,0)$$

Since we receive $\overline{c_0}$ *again, we conclude that* $\overline{c_0} = (1,1,0)$ *is the identity configuration.*

While the Dhar method can be seen at one bottom-up approach, Levine and the others have an algorithm that can be stated as top-down:

Definition 2.20 (**Levine et. al. algorithm [6]**). *Denote δ be the vector taken from the main diagonal of the degree matrix, exclude the sink. Let $\sigma = 2\delta - 2$ be the initial configuration. The stable configuration after stabilize the initial one is denoted by σ^o. Then, the identity configuration is the stable configuration after stabilizing the difference configuration between σ and σ^o.*

$$I = (\sigma - \sigma^o)^o$$

Example 2.20. *We calculate $\delta = (3, 2, 1)$, then*

$$\sigma = 2\delta - 2 = 2(3, 2, 1) - 2 = (4, 2, 0)$$

Stabilizing σ:

$$(4, 2, 0) \xrightarrow{v_1} (1, 3, 1) \xrightarrow{v_3} (2, 1, 1) \xrightarrow{v_4} (3, 1, 0)$$

$$\xrightarrow{v_1} (0, 2, 1) \xrightarrow{v_3} (1, 0, 1) \xrightarrow{v_4} (2, 0, 0)$$

We receive $\sigma^o = (2, 0, 0)$. Then the difference between σ and σ^o is equal to:

$$\sigma - \sigma^o = (4, 2, 0) - (2, 0, 0) = (2, 2, 0)$$

From this, we calculate the identity configuration by stabilizing this difference:

$$(2, 2, 0) \xrightarrow{v_3} (3, 0, 0) \xrightarrow{v_1} (0, 1, 1) \xrightarrow{v_4} (1, 1, 0)$$

We conclude with $(1, 1, 0)$ as the identity configuration.

Chapter 3

Results on identity configuration computing

The last chapter captures important definitions and remarks about graph theory and chip firing game. Also, we introduce some new terminologies that we will use later on. In this chapter, we aim to explain our application in greater detail but not too technical-related. After that, we present our example results on identity configuration computing on flower, wheel, cycle and complete graphs and provide a general remark that we will prove in the end.

3.1 Firing simulation and finding the identity configuration

In this section, we will briefly introduce the program that we have written in parallel with researching activities. It is used mainly to calculate the identity configuration of specific graphs. Besides, one can use this application as a tool to generate LaTeX-based graph or to check the configuration stability or recurrency.

With the given time, we were trying to focus only on the kind of graphs mentioned in the scope of this thesis. In short, this program can read and write under the scope of one simple graph with the allowance of maximum one sink. Specifically, it takes the set of vertices, their edges and an optional initial configuration from one input text file as an ideal input. In return, it exports one file (either under plain text or LaTeX format) contains the result based on the option the user chose, including visualizing the graph under the Tikz LaTeX package, checking the configuration whether it is recurrent or stable or calculating its identity configuration.

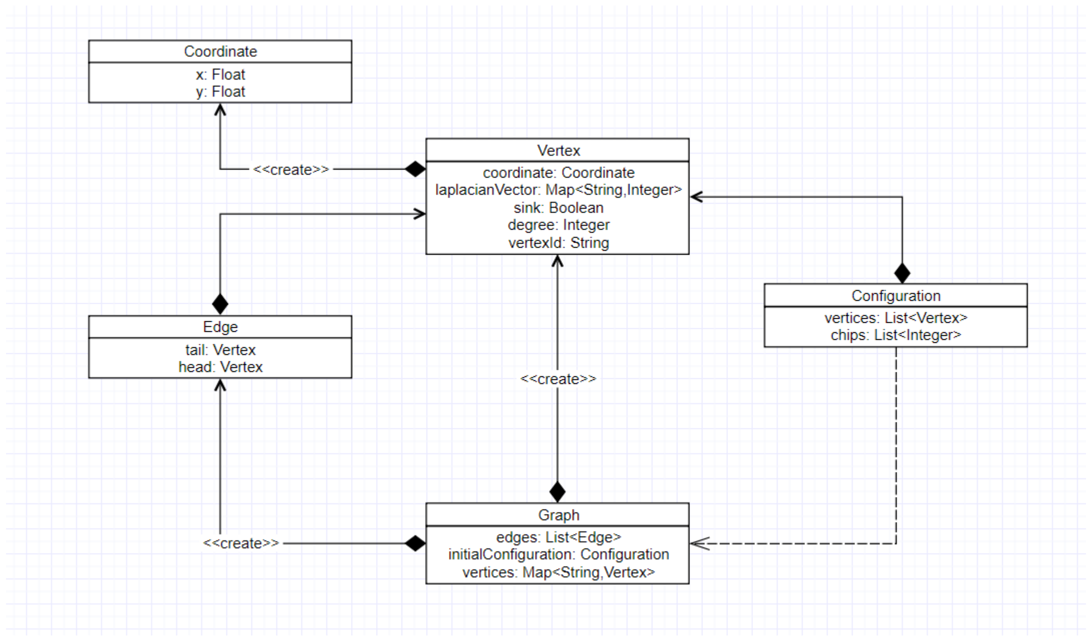


Figure 3.1: Application model diagram

The application was written in Java Standard Edition and it is object-oriented. Aware the fact that some others programming languages have a strong support in graph theory, or chip firing game simulation in specific (e.g. MATLAB); it will be a risk if we put more time on learning this new language consider the time constraints we had. Moreover, the application that we planned to create only served for some specific reasons and it is running under some specific conditions (i.e. it works under the simple graphs where there is maximal allowance of one sink); it is best to work on the language that we are familiar the most.

Since we felt the need of presenting our results not only under the text format but also under the format that can be visualized and also the programming language currently did not have a great support to graph representation, we decided to implement the LaTeX generator in the application on our own. The application, therefore, can produce a source file under LaTeX format and produce a graph.

For the sake of simplicity, only the simple dependency diagram is shown in the paper (see Figure 3.1). The program stores our graph under the set of vertices and edges and configurations. I introduced an interface class named Algorithm, leaving an open potential for implementing later specific algorithm (e.g. Dhar's back firing algorithm, Dhar's burning algorithm or Levine's identity finding algorithm).

Algorithm 1 Extended Dhar's burning algorithm

```
procedure FIND IDENTITY(Dhar)
    graph ← the given graph
begin:
    $C_{former}$ ← all-0 configuration
    $C_{latter}$ ← backfire($C_{former}$)
loop:
    $C_{latter}$ ← stabilize($C_{latter}$)
    if $C_{former} \neq C_{latter}$ then
        $C_{former}$ ← $C_{latter}$
        $C_{latter}$ ← backfire($C_{former}$)
        goto loop
    return $c_{former}$
    close;
```

Table 3.1: Extended Dhar's burning algorithm

Algorithm 2 Levine et. al.' finding identity configuration algorithm

```
procedure FIND IDENTITY(Levine)
    graph ← the given graph
begin:
    $C_{levine}$ ← levine configuration
    $C_{levineStable}$ ← stabilize($C_{levine}$)
    $C_{difference}$ ← $C_{levine} - C_{levineStable}$
    return stabilize($C_{difference}$)
    close;
```

Table 3.2: Levine et. al.' finding identity configuration algorithm

In addition, the algorithms to find identity are included in the program (See pseudo-code in table 3.1 and 3.2). Both of the algorithms need only one input, the graph, and return the identity configuration. We also include the mechanism for back firing and stabilizing configuration to help the calculation. The source code of this algorithm can be found with in the attached materials with this thesis paper or can be accessed from an open source under GitHub[1] for further references.

In conclusion, the contribution of this application is rather small to the field that we are currently working on but it is the best fit for this thesis purpose consider the given time and effort. In fact, during the implementation had I developed one complex but scale-able structure that can easily be enhanced for the future purpose, namely enhancement of input or graphical user interface.

3.2 Some results on specific classes of graph

In this section, some specific graphs will be investigated. Their identity configuration will be calculated. In some problems, it is easier to find the pattern if one start to study the relationship between the special cases and then from that try to predict and prove for more general ones. I started with the specific graphs because they have a better regularity than the random ones, thus better chance to recognize for the special patterns.

Recall on the earlier chapter, we have already mentioned some of the special kinds of graph. Let us continue to create some experiments on these classes:

Flower graphs

To keep the regularity of the graph, we decided to choose the common vertex as the sink. First of all, here are some results that can be naturally derived:

Remark 3.1. *Given a flower graph F_n with the common vertex v^*,*

- $|V| = 2n + 1$
- $|E| = 3n$
- $deg(v/v*) = 2$
- $deg(v*) = 2n$

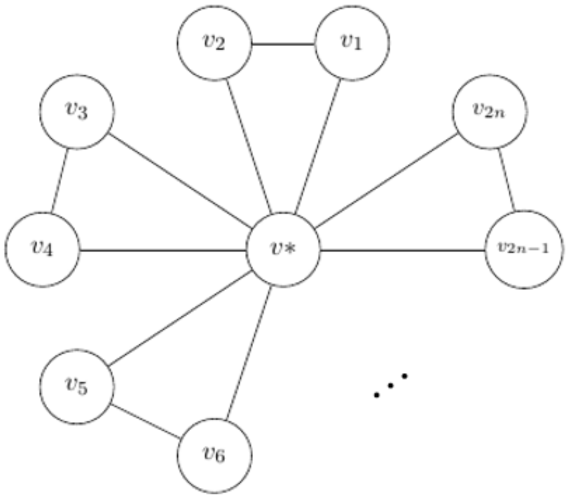


Figure 3.2: Flower graph with n petals

For convention, every petal should have an index. Starting from the top and in the counterclockwise direction, let us index the petal from 1 to n. Given a petal with index i, it contains two vertices, the left-most vertex(from the common vertex perspective) will be denoted as v_{2i} and the other will be v_{2i-1} (see Figure 3.2). Now, we calculate identity configuration of this graph:

Example 3.1. *Identity configuration of F_3 is all-1 configuration.*

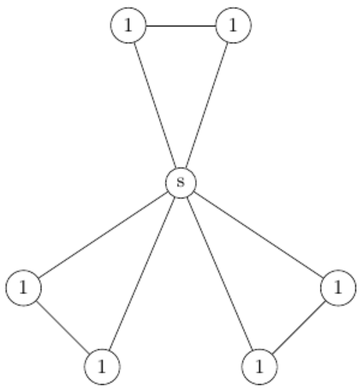


Figure 3.3: Identity configuration of F_3

[1]https://github.com/Nguyenhoang146/CFG-ConfigurationFinality

Let us calculate the identity configuration of F_3 using the extended Dhar algorithm. Denote c is the initial zero configuration of F_3, then:

$$c = (0, 0, 0, 0, 0, 0)$$

Since this is a flower graph with the sink is a common vertex, it is connected to all other vertices, therefore when applying the back-fire, we add a chip into every non-sink vertex:

$$\begin{aligned} c \rightsquigarrow \overline{c} : \overline{c} &= c + 1 \\ &= (0+1, 0+1, 0+1, 0+1, 0+1, 0+1) \\ &= (1, 1, 1, 1, 1, 1). \end{aligned}$$

Because $\overline{c}$ is still a stable configuration, we continue to back fire it a second time:

$$\begin{aligned} \overline{c} \rightsquigarrow \overline{\overline{c}} : \overline{\overline{c}} &= \overline{c} + 1 \\ &= (1+1, 1+1, 1+1, 1+1, 1+1, 1+1) \\ &= (2, 2, 2, 2, 2, 2). \end{aligned}$$

Now $\overline{\overline{c}}$ is an unstable configuration. We will try to stabilize it. Recall the matrix representation:
From the degree matrix D:

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

and adjacency matrix A:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

We calculate the laplacian matrix Δ:

$$\Delta = D - A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 2 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 2 & -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 6 \end{bmatrix}$$

Then, we derived the reduced laplacian matrix Δ':

$$\Delta' = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Notice that $\sum_{i=1}^{6} \Delta'_i = (1,1,1,1,1,1)$. From this, we can stabilize $\overline{\overline{c}}$ by sequentially fire each non-common vertices once:

$$(2,2,2,2,2,2) \xrightarrow{v_1} (0,3,2,2,2,2) \xrightarrow{v_2} (1,1,2,2,2,2) \xrightarrow{v_3} (1,1,0,3,2,2)$$
$$\xrightarrow{v_4} (1,1,1,1,2,2) \xrightarrow{v_5} (1,1,1,1,0,3) \xrightarrow{v_6} (1,1,1,1,1,1)$$

In other way, we can express the stabilizing process like this:

$$\begin{aligned} c \xrightarrow{*} c^o : c^o &= c - \sum_{i=1}^{6} \Delta'_i \\ &= c - (1,1,1,1,1,1) \\ &= (2,2,2,2,2,2) - (1,1,1,1,1,1) \\ &= (1,1,1,1,1,1). \end{aligned}$$

Since $\overline{c} \rightsquigarrow \overline{\overline{c}}$ and $\overline{\overline{c}} \xrightarrow{*} \overline{c}$, then $\overline{c} = (1,1,1,1,1,1)$ is the identity configuration of F_3.

Further graph of this kind have been examined using the simulation program above, we tried to find the identity configuration of the flower graph with 2,4 and 5 petals. Indeed, it is the all-1 configuration. From the results we gather above, the following proposition is yielded and its proof will be delayed until the next section.

Proposition 3.1. *The identity configuration of the flower graph is the all-1-configuration.*

Cycle graph

To begin with, let us present some properties of cycle graphs:

Remark 3.2. *Given a cycle C_n,*

- $|V| = n$ *where* $n \geq 3$
- $|E| = n$
- $deg(v) = 2$

For convention, let us choose a vertex to be v_1, the left neighbor of it will be v_2 and it goes on until it reach the final vertex v_n, which will have its left neighbor is v_1. Since every vertex is equivalent, one of them can be the sink. Assume vertex v_1 is chosen as a sink. We hereby calculate the identity configuration of C_5:

Example 3.2. *The identity configuration of C_5 is all-1 configuration.*

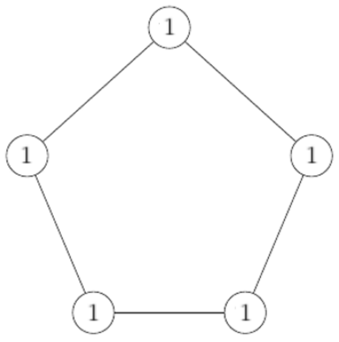


Figure 3.4: Identity configuration of C_5

Using the same approach with the flower graphs, let us calculate some matrices that we would use firstly:
From the degree matrix D:

$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

and adjacency matrix A:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

We calculate the laplacian matrix Δ:

$$\Delta = D - A = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Then, we derived the reduced laplacian matrix Δ':

$$\Delta' = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

Let us start with the zero configuration $c = (0, 0, 0, 0)$. Every time the sink fires, it sends out two chips, one for each neighbors (in this context v_2 and v_5). After the first back-fire, the new configuration will still be stable, therefore we will do it twice:

$$c \rightsquigarrow \overline{c} : \overline{c} = (1, 0, 0, 1)$$

$$\overline{c} \rightsquigarrow \overline{\overline{c}} : \overline{\overline{c}} = (2, 0, 0, 2)$$

Stabilize $\overline{\overline{c}}$, we can only fire two vertices v_2 and v_5:

$$(2, 0, 0, 2) \xrightarrow{v_2} (0, 1, 0, 2) \xrightarrow{v_5} (0, 1, 1, 0)$$

Since $\overline{\overline{c}} \xrightarrow{*} (0, 1, 1, 0) \neq \overline{c}$, let us continue to apply the algorithm, denote the stabilization of $\overline{\overline{c}}$ is c_1:

$$c_1 \rightsquigarrow \overline{c_1} : \overline{c_1} = (1, 1, 1, 1)$$

$$\overline{c_1} \rightsquigarrow \overline{\overline{c_1}} : \overline{\overline{c_1}} = (2, 1, 1, 2)$$

Let us stabilize $\overline{\overline{c_1}}$:

$$(2, 0, 0, 2) \xrightarrow{v_2} (2, 1, 1, 2) \xrightarrow{v_3} (1, 0, 2, 2) \xrightarrow{v_4} (1, 1, 0, 3) \xrightarrow{v_5} (1, 1, 1, 1)$$

This time, we receive the stable configuration $(1,1,1,1) = \overline{c_1}$ which implies $\overline{\overline{c_1}} \xrightarrow{*} \overline{c_1}$. Thus, $\overline{c_1} = (1,1,1,1)$ is the identity configuration of C_5.

However, we did some further simulation on different kind of cycles and found out that this is not always the case. As the cycles of 4 and 6 shown, the identity configuration were consecutively (1,0,1) and (1,1,0,1,1). Thus, we have a remark:

Proposition 3.2. *The identity configuration of the odd cycles will be all-1.*

Since we are trying to find the pattern, we ignore some oddities. However later on, we will also try to find the identity configuration of the even cycles as well.

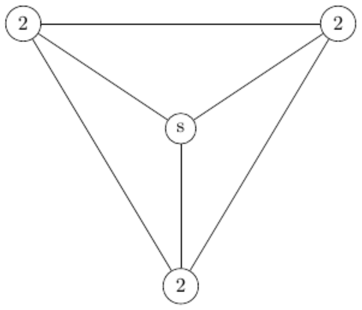
Wheel graph

Similar to the flower graph, let us choose common vertex as the sink. Here are some properties of the wheel graph:

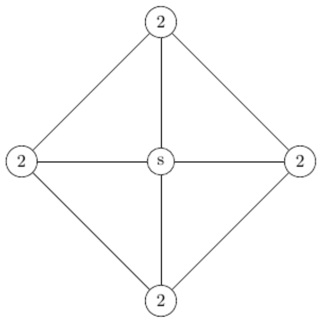
Remark 3.3. *Given a wheel graph W_n:*

- $|V| = n$ *where* $n \geq 4$
- $|E| = 2n - 2$
- $deg(v/v*) = 3$
- $deg(v*) = n - 1$

Having used the finding identity configuration program and chose the common vertex as the sink, I conducted the following examples:



(a) Wheel graph W_4



(b) Wheel graph W_5

Figure 3.5: Identity configuration of W_4 and W_5

Example 3.3. *Identity configuration of W_4 and W_5 is all-2 configuration (see Figure 3.5).*

In this example, let us find the identity element using the approach from Levine and the others. Firstly, we present the matrix representation of W_5, we put the sink at the last vertex:
From the degree matrix D:

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

and adjacency matrix A:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

We calculate the laplacian matrix Δ:

$$\Delta = D - A = \begin{bmatrix} 3 & -1 & 0 & -1 & -1 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & -1 & 3 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{bmatrix}$$

Then, we derive the reduced laplacian matrix Δ':

$$\Delta' = \begin{bmatrix} 3 & -1 & 0 & -1 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ -1 & 0 & -1 & 3 \end{bmatrix}$$

From definition, we have $\delta = (3, 3, 3, 3)$. Then σ is calculated from δ:

$$\sigma = 2\delta - 2 = 2(3, 3, 3, 3) - 2 = (4, 4, 4, 4)$$

Stabilize the σ, we have:

$$(4, 4, 4, 4) \xrightarrow{v_1} (1, 5, 4, 5) \xrightarrow{v_2} (2, 2, 5, 5) \xrightarrow{v_3} (2, 3, 2, 6) \xrightarrow{v_4} (3, 3, 3, 3)$$

$$\xrightarrow{v_1} (0,4,3,4) \xrightarrow{v_2} (1,1,4,4) \xrightarrow{v_3} (2,1,1,5) \xrightarrow{v_4} (2,2,2,2)$$

Then $\sigma \xrightarrow{*} \sigma^o = (2,2,2,2)$. Since $\sigma - \sigma^o = \sigma^o = (2,2,2,2)$ then identity configuration will be the result of the stabilization of σ^o, which is itself (because it is already stable).
Thus, (2,2,2,2) is the identity configuration of W_5.
From this, we arrive at the proposition.

Proposition 3.3. *The identity configuration of the wheel graph is the all-2-configuration.*

Complete graph

Let recall some basic properties of the complete graph:

Remark 3.4. *Given a complete graph K_n:*

- $|V| = n$
- $|E| = n(n-1)/2$
- $deg(v) = n - 1$

Similar to the case of cycles, since all of the vertices are equivalent, we chose one vertex as a sink of the complete graph. Assume we choose v_n to be the sink. Let us take the K_4 as an example to find the identity configuration:

Example 3.4. *The identity element of K_4 is all-2 configuration.*

To calculate the identity element, we must first present the graph under some matrix representation:
From the degree matrix D:

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

and adjacency matrix A:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

We calculate the laplacian matrix Δ:

$$\Delta = D - A = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Then, we derived the reduced laplacian matrix Δ':

$$\Delta' = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix}$$

Applying the Levine algorithm in this situation, we yield the $\delta = (3, 3, 3)$. From this, we calculate our initial configuration:

$$\sigma = 2\delta - 2 = 2(3, 3, 3) - 2 = (4, 4, 4)$$

Stabilize the σ, we have:

$$(4,4,4) \xrightarrow{v_1} (1,5,5) \xrightarrow{v_2} (2,2,6) \xrightarrow{v_3} (3,3,3) \xrightarrow{v_1} (0,4,4)$$
$$\xrightarrow{v_2} (1,1,5) \xrightarrow{v_3} (2,2,2)$$

Then $\sigma \xrightarrow{*} \sigma^o = (2, 2, 2)$. Since $\sigma - \sigma^o = \sigma^o = (2, 2, 2)$ then identity configuration will be the result of the stabilization of σ^o, which is itself (because it is already stable).
Thus, (2,2,2) is the identity configuration of K_4.

We take one further step to calculate the identity element of complete graph of n where n is larger than 4. Here are the results:

Table 3.3: Identity configuration under the complete graph of n

Number of vertices n	**Identity configuration** I
5	$(3,3,3,3)$
6	$(4,4,4,4,4)$
7	$(5,5,5,5,5,5)$
$\vdots$	$\vdots$
20	a 19-vector of all-18

From this, a proposition has been proposed:

Proposition 3.4. *The identity configuration of the complete graph is its maximal stable configuration.*

3.3 Derived results for general case

More often than not, we noticed the maximal stable configuration coincides with the identity configuration in most of our considered cases. Inspired by the results, one might wonder: "given what type of graphs will result the identity as the maximal stable configuration?".

First of all, let us establish the relationship between the identity configuration with the reduced laplacian matrix.

Lemma 3.5. *Given a simple connected graph G with a sink s. The identity configuration I of this graph can be expressed under the matrix equation:*

$$\Delta'(G)x = I$$

in which exists exactly one non-trivial natural solution.

Proof. Recall the circumstance we defined, given any initial accessible configuration for graph G with the sink s, since we always play a finite game, by applying the stabilize operation, we receive one stable configuration. In other way, we obtain final stable configuration by firing the sequence of vertices in $G \setminus s$. Earlier in this thesis, we defined the operation for firing one vertex as:

$$u \xrightarrow{v_i} v : v = u - \Delta'_i$$

Based on the work of Dhar, we know that the firing order does not effect the final configuration since there can only be one stable configuration from one given initial one [5]. Therefore we can calculate the final stable configuration as:

$$u \xrightarrow{*} u^o : u^o = u - x_1\Delta'_1 - x_2\Delta'_2 ... - x_{n-1}\Delta'_{n-1}$$

where $x_1, x_2, ..., x_{n-1} \in \mathbb{N}$.

Let us denote $x = (x_1, x_2, ..., x_{n-1})$ is the solution vector for the above calculation, we can rewrite it under the form of matrix equation:

$$\Delta'(G)x = u - u^o$$

In other way, we can consider x to be the vector which contains the information of the times each vertex fires to transform from configuration u to u^o. As Levine stated, there exists an unique vector like that [6]. Thus, the equation above contains only one non-trivial natural solution.

From the definition of identity configuration (see Definition 2.19), we know that:

$$\forall u \in S(G) : (I + u) \xrightarrow{*} u$$

Combine with the matrix equation above, we derived:

$$\Delta'(G)x = (I + u) - u = I$$

□

Based on the lemma we have just proved, denoted the maximal configuration by c_{max}. Let us propose our next lemma:

Lemma 3.6. *The maximal stable configuration is always recurrent.*

Proof. In order to test the recurrent validity of the maximal configuration, let us apply the burning algorithm from Dhar:

$$\begin{aligned} c_{max} \rightsquigarrow v : v &= c_{max} - \Delta'_s \\ &= c_{max} + \Delta'_1 + \Delta'_2 + ... + \Delta'_{n-1} \qquad \text{(from Remark 2.7)} \end{aligned}$$

Since we are firing the sink to send some chips back to the non-sink vertices on the maximal stable configuration, the vertices that takes the sink as its neighbor will be fire-able. After firing these vertices, the vertices that are the neighbors of these will also be fire-able... Since this is the connected graph, we are sure that in order to stabilize v, we have to fire each of the non-sink vertex once:

$$v \xrightarrow{N(s)} \ldots \xrightarrow{N(N(s)) \setminus N(s)} \ldots \xrightarrow{N(N(N(s))) \setminus (N(s) \cup N(N(s)))} v^o$$

Every time we fire a vertex v_i, we know that the new configuration will be the difference between the old one with the reduced laplacian vector of v_i. Then stabilizing by firing each of the vertex once means:

$$\begin{aligned} v \xrightarrow{*} v^o : v^o &= v - (\Delta'_1 + \Delta'_2 + ... + \Delta'_{n-1}) \\ &= (c_{max} + \Delta'_1 + \Delta'_2 + ... + \Delta'_{n-1}) - (\Delta'_1 + \Delta'_2 + ... + \Delta'_{n-1}) \\ &= c_{max} \end{aligned}$$

Since c_{max} is already stable, then we are sure that c_{max} is indeed the recurrent configuration. □

Finally, let us provide our theorem:

Theorem 3.7. *Given a simple connected graph G with the sink s, the identity configuration coincides with the maximal stable configuration iff there exists one non-trivial natural solution for $\Delta'(G)x = c_{max}$.*

Proof. Derived by the lemma 3.5, we have already prove whether if the maximal stable configuration is the identity one then the equation $\Delta'(G)x = c_{max}$ will have one non-trivial natural solution of x.

Now we are going to prove the other direction. From lemma 3.6, we know that c_{max} is already a recurrent configuration. Let us assume that c is the identity configuration and it is different than c_{max}, then from lemma 3.5, we have a vector r of $n-1$ natural numbers such that:

$$c = \Delta'(G)r = \Delta'_1 r_1 + \Delta'_2 r_2 + ... + \Delta'_{n-1} r_{n-1}$$

Since c_{max} is recurrent, then by the abelian group property, we have $(c + c_{max}) \overset{*}{\longrightarrow} c_{max}$. In particular, we can add the two configurations to create the new one, then stabilize it by selective firing each vertex such that the total number of times a vertex fire equals to the vector r value at its position. The final result will be indeed the maximal configuration.

However, since

$$c_{max} = \Delta'(G)x = \Delta'_1 x_1 + \Delta'_2 x_2 + ... + \Delta'_{n-1} x_{n-1}$$

there also exists a sequence of firing where we selectively fire each of the vertex once. The final result will be the identity configuration.

This contradicts to the fact that starting from any configuration (e.g. the sum of c and c_{max}) we can only receive one final stable configuration after the stabilization and the number of times each vertex fires is exactly the same regardless of the firing sequence [5].

Thus, c_{max} must be the identity configuration. □

Given a simple graph $G = (V, E)$, the rank of laplacian matrix of G can be calculated by the difference between the number of vertices and the number of connected components in G [8]. The graph that we are considering in our thesis is simple and connected, therefore the rank of the laplacian matrix will be $|V| - 1$. From this, the rank of the reduced laplacian matrix will also be $|V| - 1$ which leads to the fact that the matrix equation $\Delta' x = u$, for any configuration u, will always has a solution. However, the solution has to be non-trivial natural in order to make u an identity configuration of G. In fact, the table below shows some examples we have conducted to enhance our proof:

Graph G	**Rank of**Δ'	**Solution for** $\Delta' x = c_{max}$	**Is** $c_{max} = I$**?**
F_3	6	$x = (1,1,1,1,1,1)$	Yes
F_4	8	x is a 8-vector of all-1	Yes
F_{10}	20	x is a 20-vector of all-1	Yes
F_{20}	40	x is a 40-vector of all-1	Yes
C_4	3	$x = (\frac{3}{2}, 2, \frac{3}{2})$	No
C_5	4	$x = (2,3,3,2)$	Yes
C_6	5	$x = (\frac{5}{2}, 4, \frac{9}{2}, 4, \frac{5}{2})$	No
C_9	8	$x = (4,7,9,10,10,9,4,7)$	Yes
C_{10}	9	$x = (\frac{9}{2}, 8, \frac{21}{2}, 12, \frac{25}{2}, 12, \frac{21}{2}, 8, \frac{9}{2})$	No
$W_4(K_4)$	3	$x = (2,2,2)$	Yes
W_5	4	$x = (2,2,2,2)$	Yes
W_{10}	9	x is a 9-vector of all-2	Yes
W_{20}	19	x is a 19-vector of all-1	Yes
K_5	4	$x = (3,3,3,3)$	Yes
K_{10}	9	x is a 9-vector of all-8	Yes
K_{20}	19	x is a 19-vector of all-18	Yes

Table 3.4: Experimental results under various graphs

From the important theorem above, let us reconsider the special graphs that we have been working on and try to apply the new theorem. Consider the complete graph of n:

Corollary 3.8. *In complete graph of n* $(n \geq 3)$ *with one random vertex as a sink, maximal configuration is the identity.*

Proof. First of all, let us define some components of the proof. Given a complete graph $K_n = (V, E), |V| = n$ with the last vertex v_n is the sink, the following representations can be calculated:

- Reduced laplacian matrix is a $(n-1) \times (n-1)$ matrix:

$$\Delta' = \begin{bmatrix} (n-1) & \dots & -1 & -1 \\ \vdots & \ddots & \vdots & \vdots \\ -1 & \dots & (n-1) & -1 \\ -1 & \dots & -1 & (n-1) \end{bmatrix}$$

- Maximal stable configuration is a $(n-1)-$vector:

$$c_{max} = (n-2, n-2, ..., n-2)$$

- Reduced laplacian vector of the sink is a $(n-1)-$vector:

$$\Delta'_s = (-1, -1, ..., -1)$$

From the preliminaries, we have the relationship between the reduced laplacian vector of the sink with the reduced laplacian matrix, we can establish a equation:

$$\Delta'_s = (-1, -1, ..., -1) = -\sum_{i=1}^{n-1} \Delta'_i$$

Since $c_{max} = -(n-2)\Delta_s$, therefore we have the solution the equation $\Delta' x = c_{max}$ is a $(n-1)-$vector:

$$x = (n-2, n-2, ..., n-2)$$

Since K_1 is just a single vertex and K_2 is an association between two vertices, there exists no stable configuration to consider. From $n = 3$, it is clear that x is non-trivial and natural. Thus, the solution for matrix equation $\Delta' x = c_{max}$ is non-trivial natural.

From theorem 3.7, we derived that the identity configuration of complete graphs coincide with the maximal stable configuration. □

Corollary 3.9. *In wheel graph of n $(n \geq 4)$ with the common vertex as a sink, maximal stable configuration is the identity.*

Proof. Similar to the previous proof, given a wheel graph $W_n = (V, E), |V| = n$ with the last vertex v_n is the sink, the following representations can be calculated:

- Reduced laplacian matrix:

$$\Delta'_{ij} = \begin{cases} deg(v_i) = 3 \text{ for j = i} \\ -a_{ij} \text{ for j} \neq \text{i} \end{cases}$$

  It is difficult to visualize the reduced laplacian matrix. Nevertheless, we can describe it by row representation. In the i-th row of the reduced laplacian matrix, the element at the i-th position is equal to the degree of the vertex, which is 3, while the element at the (i+1)-th and (i-1)-th position[2] is equal to -1; the remaining remains zero.

[2]There exists a case of the first and the last element where the its (i+1)-th or (i-1)-th position can be the other.

- Maximal stable configuration is a $(n-1)-$vector:

$$c_{max} = (2, 2, ..., 2)$$

We notice that since this is a simple graph, the reduced laplacian matrix here is symmetric. Therefore, it includes an element of value 3, two elements of value -1 and zeros in every column of the matrix. From here, if we add all of the rows together, we will receive a vector $c = (1, 1, 1, ...)$.

Since $c_{max} = 2c$, therefore we have the solution the equation $\Delta' x = c_{max}$:

$$x = (2, 2, ..., 2)$$

From theorem 3.7, we derived that the identity configuration of wheel graphs coincide with the maximal stable configuration. □

About the cycles, since it is not true for all of the graphs. For instance, C_4 takes $(1, 0, 1)$ as an identity configuration while $(1, 1, 1, 1)$ is the maximal stable configuration in C_5. We will firstly prove the easier part:

Corollary 3.10. *In odd cycle graph of n ($n \geq 3$) with the vertex v_n as a sink, maximal stable configuration is the identity.*

Proof. For this proof, given a cycle graph $C_n = (V, E), |V| = n$ with the last vertex v_n is the sink, the following representations can be calculated:

- Reduced laplacian matrix:

$$\Delta'_{ij} = \begin{cases} deg(v_i) = 2 \text{ for j = i} \\ -a_{ij} \text{ for j} \neq \text{i} \end{cases}$$

  Since v_n is the sink, it will only effect on v_1 and v_{n-1} when create the reduced laplacian matrix from the laplacian matrix. As the example in section 3.2 shown, all of the rows will contains one element of value 2 - the vertex degree and two elements of value -1 (which are allocated on the neighbor position) except two special rows (v_1 and v_{n-1}) where there exists only one element of value -1 since v_n has been removed.

- Maximal stable configuration is a $(n-1)-$vector

$$c_{max} = (1, 1, ..., 1)$$

It is more difficult to find the solution for the equation $\Delta' x = c_{max}$ than the other graphs. Therefore, we conduct a chain of experiments on this kind of graphs. We examined the solution of the equation above:

Table 3.5: Cycle graphs and the equation's solution

k	**Graph** C_{2k+1}	**Solution of** $\Delta' x = c_{max}$
1	C_3	$(1, 1)$
2	C_5	$(2, 3, 3, 2)$
3	C_7	$(3, 5, 6, 6, 5, 3)$
4	C_9	$(4, 7, 9, 10, 10, 9, 7, 4)$

From the table, we create an algorithm to calculate the solution for the equation:

Algorithm 3 Solution of $\Delta' x = c_{max}$ in odd cycles

```
procedure SOLVE($C_n$)
begin:
    $d \leftarrow n - 1$ (dimension of $\Delta'$)
    $x \leftarrow (1, 1)$ (initial solution vector)
    $k \leftarrow \frac{d}{2}$
    $i \leftarrow 1$
loop:
    if $i \neq k$ then
        $x \leftarrow (0, x, 0)$ (include zeros to both ends of x)
        $i \leftarrow i + 1$
        $x \leftarrow x + i$ (scalar addition)
        goto loop
    return $x$
    close;
```

Table 3.6: Solution of $\Delta' x = c_{max}$ in odd cycles

The proof for the validity of this algorithm can be proven using induction method. Since the output of this algorithm always be natural, starting from $(1, 1)$, thus the equation always has non-trivial natural solution.

From theorem 3.7, we derived that the identity configuration of odd cycle graphs coincide with the maximal stable configuration. □

For the even cycles, we also conduct a remark:

Remark 3.5. *In even cycle graph of n ($n \geq 4$) with the vertex v as a sink, the identity configuration of C_n is $I = (1, 1, ..., 0, ..., 1, 1)$ where zero is the number of chips at the far-most vertex compared to the sink v.*

In fact, this remark is true for C_4 and C_6, where the identity configuration is $(1, 0, 1)$ and $(1, 1, 0, 1, 1)$. By applying the algorithm above and adjust some initial value, we can calculate the non-trivial solution for the equation $\Delta' x = I$ also.

Algorithm 4 Solution of $\Delta' x = I$ in even cycles

```
 1: procedure SOLVE($C_n$)
 2: begin:
 3:     $d \leftarrow n - 1$ (dimension of $\Delta'$)
 4:     $x \leftarrow (1, 1, 1)$ (initial solution vector)
 5:     $k \leftarrow \frac{d-1}{2}$
 6:     $i \leftarrow 1$
 7: loop:
 8:     if $i \neq k$ then
 9:         $x \leftarrow (0, x, 0)$ (include zeros to both ends of x)
10:         $i \leftarrow i + 1$
11:         $x \leftarrow x + i$ (scalar addition)
12:         goto loop
13:     return $x$
14:     close;
```

Table 3.7: Solution of $\Delta' x = I$ in even cycles

Last but not the least, let us conclude this chapter with the corollary for flower graphs:

Corollary 3.11. *In the flower graph of n with the common vertex as the sink, maximal stable configuration is the identity.*

Proof. Similar to others, we first list out the initial components for our calculation:

- Reduced laplacian matrix: Since it is hard to visualize the matrix, we break it down to each row. At the i-th row of reduced laplacian matrix:

$$\Delta'_{ij} = \begin{cases} deg(v_i) = 2 \text{ for j = i} \\ -a_{ij} \text{ for j} \neq \text{i} \end{cases}$$

- Maximal stable configuration:

$$c_{max} = (1, 1, 1, ...)$$

Each row of the reduced laplacian matrix, there exists only one element that contains the degree of the vertex and one element that contains the value of -1 (each vertex only connects to one neighbor within the petal and the sink). From here, if we add all of the rows together, we will receive a vector $c = (1, 1, 1, ...)$.

Since $c_{max} = c$, therefore we have the solution the equation $\Delta' x = c_{max}$:

$$x = (1, 1, 1, ...)$$

From theorem 3.7, we derived that the identity configuration of flower graphs coincide with the maximal stable configuration. $\square$

Chapter 4

Conclusions and Future work

To begin with, we began the work with some fundamental background studies of the graph theory and some algebraic matrix calculations. We continued to explore further the chip firing game throughout some of the papers, especially by Dhar [5] and Levine et. al. [6].

Secondly, we created an application that can simulate the chip firing game and receive some positive results. We then used the program to run on some of the specific classes of graph and discover some patterns among them. One of them was the frequency of the identity and the maximal stable configuration coincidence.

From that, we conducted a predication under which graph properties this pattern exists. The result was not beautiful or can be seen as derivative but it is still significant. Last but not the least, we tried to prove the specific graphs above share the same properties.

For the future work, we noticed that the flower graph is, in fact, the combination of many 3-cycle graphs. They are connected together through one special vertex, also known as the sink. Under the property of the flower graph - that is connecting petals through one common vertex, we have conducted some experiments where we try to combine different classes of graph on each petal and merged their sink into the common vertex. The results shown that it is the case where the configuration, whether it is stable or recurrent, is the combination of the corresponding configuration in the sub-graphs. Moreover, the identity configuration of these graphs is also the combination from the identity configuration of its components. As a consequence, we would like to provide a proof for that property in the future.

Furthermore, we also wonder what would happen if two cycles did not only share the same single common vertex like above, but share the same common path. In other ways, there are two vertices that are shared between a pair of graphs (see Figure 4.1). Furthermore, we can extend this work to another class of graph, which is known as the ear decomposition. These are very interesting topics that we would also like to have a look.

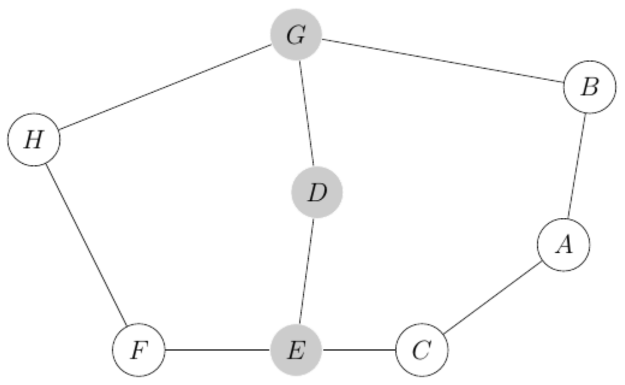


Figure 4.1: A simple graph contains two sub-graphs each is cycle of 5. These two graphs share the same path between E and G

It is also worth to consider the relationship between the sink and the identity configuration. We, hereby, propose one problem statement:

"Given a simple connected graph G with the sink s, the identity configuration can be presented under the relation $(G, s) = I$. The question is whether one will be able to conduct the new identity configuration from the same graph G and the old identity configuration if we decide to change the sink, meaning $(G, s) \longrightarrow (G, s')$".

Bibliography

[1] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality, Jul 1988.

[2] N. L. Biggs. Chip-firing and the critical group of a graph, January 1999.

[3] Anders Björner and László Lovász. Chip-firing games on directed graphs. *J. Algebraic Comb.*, 1(4):305–328, December 1992.

[4] Robert Cori and Dominique Rossin. On the sandpile group of dual graphs. *European Journal of Combinatorics*, 21(4):447 – 459, 2000.

[5] Deepak Dhar. Self-organized critical state of sandpile automaton models, Apr 1990.

[6] Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuyal Peres, James Propp, and David B. Wilson. Chip-firing and rotor-routing on directed graphs, 2008.

[7] Kenneth H. Rosen. *Discrete Mathematics and Its Applications.* McGraw-Hill Higher Education, 5th edition, 2002.

[8] Richard P. Stanley. *Enumerative Combinatorics: Volume 1.* Cambridge University Press, New York, NY, USA, 2nd edition, 2011.