2005

Investigations of a chip-firing game

David Perkins
The University of Montana

INVESTIGATIONS OF A CHIP-FIRING GAME

by

David Perkins

B.A. Houghton College, 1988

M.S. South Dakota State University, 1996

presented in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

The University of Montana

May 2005

Approved by:

P. Mark Kayll

Chairperson

[illegible]

Dean, Graduate School

5-17-05

Date

Perkins, David Ph.D., May 2005 Mathematical Sciences

Investigations of a Chip-firing Game

Chairperson: Professor Mark Kayll P.M.K

We investigate a variation on a chip-firing game in which a node v may fire only if it is possible to transfer a chip to each neighbor of v and discard a chip from v. The discarded chip provides the variation, so we call this kind of chip-firing game a *burn-off game*. Chip-firing games have been used by physicists to study a property of complex systems called *self-organized criticality*. The main goal of this dissertation is to study burn-off games using strictly mathematical methods.

The chip distributions that occur during a sequence of burn-off games are called *legal configurations*. We characterize legal configurations, and then show that each legal configuration is equally likely, in the long run, to arise during a sequence of burn-off games played on any connected graph. In the case of complete graphs, this allows us to find a closed formula for the probability of experiencing a burn-off game of any desired length. For connected graphs, we describe a method for calculating the same probabilities.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

My sincere thanks go to the members of my committee: William Derrick, Rudy Gideon, Jennifer McNulty, and Steven Sheriff. The longer I teach, the more I value being part of an academic community that stretches across this country and around the world.

I am grateful to the administration of Houghton College, whose president and academic dean trusted me to finish this degree while I taught. Houghton has also supported me financially during this endeavor.

My colleagues in the math department at Houghton College deserve my heartfelt thanks as well. They have encouraged me at every step, and have taken extra work onto themselves so that I could focus on this dissertation.

Finally, my thanks to my wife Anjuli and my advisor Mark Kayll. Anjuli travelled across the country, hung out by the Blackfoot River, rode bikes and scooters, planned trips to Glacier, and did many other wonderful things with me during the days I worked on this document. Mark agreed to remain my advisor even when I sidestepped his advice and took the job at Houghton College as my dissertation was getting under way. Working on this document during the past six summers, sometimes long distance, was difficult, and only possible thanks to Mark's tireless efforts. Without the support of these two people, I cannot imagine I would be writing these words today. Thank you.

LIST OF FIGURES

Chapter 1

Introduction

1.1 Models

Over the past twenty years, physicists have modeled complex physical systems with simple computer models that display some of the important characteristics of their real-life counterparts. One example of such a complex system is the earth's crust. Sudden transfers of energy within the earth's crust create seismic events, such as earthquakes, that are felt on the surface. Most of these events are brief and minor, while a few cause great damage.

On August 14, 2003, a power blackout affected over 50 million people in Canada and the northeastern United States. In [17], Sara Robinson relates that engineers who are studying the history of power blackouts consider the electrical grid to be a complex system in which blackouts are events just like earthquakes are events in the earth's crust. Other complex systems that have been modeled include snowfields, traffic, and the human brain (see Section 1.4). All of these examples qualify as *systems* in the sense that they are composed of many small interacting parts. The earth's crust may be viewed as a collection of small platelets of earth that press against each other and occasionally shift. The electrical grid is a web of interacting wires and power stations. A snowfield, at its finest level, is a collection of snowflakes that put weight on each other until a group of them slide downhill.

These systems are *complex* in that their behaviors, while predictable in one sense, are entirely unpredictable in another. If we consider these systems from a broad perspective, we can be fairly secure in our predictions about them. For example, small earthquakes occur with great frequency, while large earthquakes are uncommon. While geologists are able to make relatively accurate predictions about when and where the next large earthquake will strike, they rely on macro-level observations, like fault lines and historical data, to make their claims as opposed to closely observing the tiny interacting chunks of earth. The system they study is far too complex for such a micro-level analysis to be helpful; the same is true for power blackouts and avalanches.

The models for these complex systems have one feature in common: they simulate a network of individuals that, when stimulated, may interact with their neighbors by distributing some sort of stress. In fact, all of the models mentioned thus far are special cases of "chip-firing games", studied in a recent paper by Bjorner *et al.* [6]. We turn now to a discussion of these games.

1.2 Chip-firing games

A *chip-firing game* takes place on a connected graph; all graphs in this thesis are simple. We assume throughout this dissertation that the reader is familiar with both basic graph theory, as introduced, e.g., in [19], and basic probability theory, as introduced, e.g., in [10]. (A glossary of notation and terminology appears in Appendix 1.) The nodes of the graph take the role of the individuals in the network. Integer-valued stresses upon each individual are represented by chips contained on the nodes. When a node fires, chips are moved to its neighbors in accordance with a *firing rule*. A node cannot fire until the number of chips it contains meets or exceeds a threshold, called the node's *critical number*.

A generalized chip-firing game that includes as special cases the models discussed in Section 1.1 has three features:

(a) the game takes place on a connected graph;

(b) each node v has a critical number k_v;

(c) if the number of chips on v is at least k_v, then v can fire, sending a chip to k_v of its neighbors, according to some rule.

If no node is able to fire, we say that the game is *relaxed*. A node v containing exactly k_v chips is *critical*. If v contains more than k_v chips, we say v is *supercritical*.

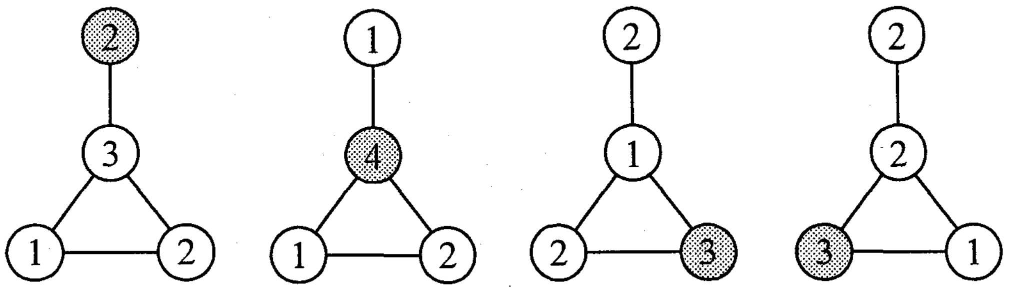

Figure 1.1. A chip-firing game

In Figure 1.1, the node numbers enumerate the chips resting on their respective nodes. The game operates as follows: each node v has critical number k_v = deg(v); when a node fires, it sends one chip to each neighbor. The figure shows a typical sequence of configurations under these stipulations. In each configuration, the shaded node is the one about to fire, which leads to the next configuration.

Note that in the first configuration, all but the lower-left node may fire. Bjorner *et al.* [6] showed that when more than one node may fire, the order in which the nodes are fired does not affect either the number of firings until the game is relaxed or the final configuration of the chips. Note also that the final configuration would lead to a configuration identical to the initial one if the shaded node fired. This suggests that this particular chip-firing game will be of infinite length. Bjorner *et al.* proved that if the number of chips on a graph exceeds a certain value (that depends on the numbers of nodes and edges in the graph), the chip-firing game will indeed never terminate.

The chip-firing game studied in this dissertation is similar to that of [6], with one small difference that will force all games to be of finite length. Before we turn our attention to this new game, we formally define "chip-firing game" and a few related terms.

1.3 Definitions involving chip-firing games and burn-off games

Let $G = (V,E)$ be a connected graph. For each node v, we place a nonnegative number $C(v)$ of chips on v. A particular distribution $C : V \rightarrow \mathbb{N}$ of chips on V is called a *configuration*. Any node v that contains at least as many chips as its *critical number* k_v—that is, $C(v) \geq k_v$—may be *fired*; we call such a node v *fire-able*. A node that fires sends a chip to each of its neighbors.

With these definitions in mind, we consider the following *chip-firing game*: Begin with any configuration on G. If there exists a node that may be fired, we fire it. This constitutes a *turn* of the game. As the turns of a game progress, we say that the configuration is *relaxing*. If no node may be fired, the game ends; otherwise, a new turn begins in which any fire-able node is fired. The *length* of the game is the number of turns taken from the initial configuration until the game ends. If at the start of the game no node may be fired, the game has length zero. If no node is able to fire, we say that the configuration is *relaxed*. To play a new game, a chip is placed on a randomly selected node. This node is called the *seed*. We also use *seed* as a verb to refer to the process of making this selection.

In most models, including the Bjorner *et al.* model [6], we take k_v to be the degree deg(v) of v. A theorem in [6] states that in such a chip-firing game, the nodes may be fired in any allowable order, and

(a) the length of the game will not be affected;

(b) the final configuration will not be affected.

This theorem implies that if the graph contains more than one fire-able node, the choice of which node to fire has no bearing on the length of the game or the final configuration of the chips. The same paper [6] points out that some chip-firing games may be of infinite length. For example, if the number of chips on the graph exceeds twice the number of edges, then, by the pigeonhole principle, we may always find at least one node that contains enough chips to fire. The paper also establishes that if the number of chips on the graph is less than the number of edges, then every chip-firing game is of finite length.

In our consideration of models for real-world phenomena, it is helpful to study a slightly different chip-firing game in which all games are of finite length. Our goal is to enumerate games of each possible length, and infinite games obstruct this analysis. Further, since many models are linked to phenomena such as earthquakes and avalanches that involve friction and kinetics, it is reasonable to build into those models some recognition that the energy in the system may be lost through escaping heat. Indeed, Kauffmann [12, pp. 71-92] argues that complex systems spontaneously exhibit both order and complexity at the cost of thermodynamic energy loss.

Therefore, we consider the following variation of the chip-firing game above: a node v may only be fired when $C(v) \geq \deg(v) + 1$, and when v is fired, one of its chips is lost from the system. The chip that escapes at each firing models the loss of energy from the system. We call such a game a *burn-off game*. Such a system is referred to as "dissipative" by physicists, in contrast with "conservative" systems like classical chip-firing games, that do not lose energy as they are processed. We also note that *critical* takes on a new meaning in the context of burn-off games: a critical node is one that will fire when a chip is added to it.

The study of modified chip-firing games is not uncommon. In [9], Eriksson summarizes and extends the results of [6] by considering games on directed graphs and games where the edges are weighted. In the latter case, the values placed on the nodes are not limited to integers but allow any real numbers. A node fires when its value is negative, and this value is added to the values on its neighbors. Both modified games exhibit properties of the original described in [6], although in some ways they differ drastically.

Eriksson continues to explore modified chip-firing games in [8], where he considers graphs that mutate between games. After a node fires, the edges incident with the node may be erased and then new edges may be added. The mutations occur in a predetermined order; this order may or may not have a finite period. As in the other papers mentioned so far, close attention is given to discovering the conditions under which the games are finite or infinite.

The variant studied by Biggs in [4] includes a vertex that can always fire, but will only do so if no other node in the graph can fire. Biggs confirms the results in [6] for his variant and demonstrates that the set of "stable", "recurrent" configurations of a graph has the structure of an abelian group.

A final example of recent interest in this topic is [5]. Bitar and Goles consider the periodic nature of chip-firing games in which all nodes that can fire are fired simultaneously. These "parallel" chip-firing games are shown to simulate logical functions, like OR and NOT, making chip-firing games of nontrivial interest to computer scientists.

1.4 Self-organized criticality

Every firing in a burn-off game removes a chip from the system; clearly, every burn-off game is of finite length. Later (see Lemma 4.8) we will see that the length of a burn-off game is bounded from above by the number of nodes. For example, burn-off games played on the graph in Figure 1.1 will be of lengths zero through four. Figure 1.2 shows the results of 10,000 simulated burn-off games played on the graph in Figure 1.1; the code, written in Smallbasic, appears in

Appendix 2. The simulation proceeds as discussed in Section 1.3 (p. 5). Note that longer games occur with lower frequency.

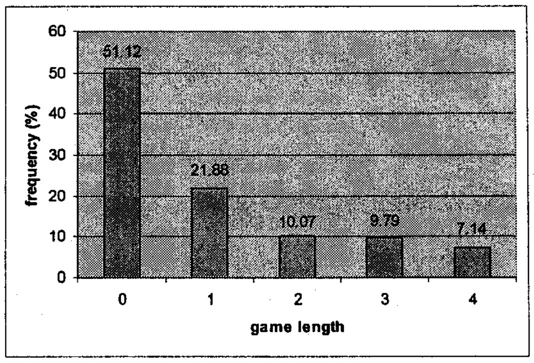

Figure 1.2. Results of 10,000 burn-off games

Empirical results like these are common in papers that investigate simple models of complex systems. In [7], for example, Dhar models a pile of sand by assigning numerical values to cells in a lattice. Each value represents the steepness of the sandpile in that region. Stresses are added at random until a cell's steepness exceeds some threshold—at which point the sand slips. In the model, the simulated stress is distributed to the cell's neighbors, and again the cells are checked for instability. The simulated avalanche continues until all cells of the grid have steepnesses within the threshold, at which point the number of cells that slipped during the avalanche is recorded. This process continues through many computer-driven iterations, and when the data is compiled, it compares favorably with measurements taken from real sandpiles.

Complex systems like this sandpile model potentially possess what is called *self-organized criticality* (hereafter abbreviated SOC). The important features of SOC are listed in [1], a nontechnical introduction to the subject that appeared in *Scientific American*. The primary characteristic of SOC is this: the system, although complex, displays predictable behavior when viewed at a macroscopic level. Such systems are simulated using computer software that collects data relating the size and frequency of events. In a sandpile model, the events are avalanches (as measured by the number of sand particles that slipped during the event), and they can occur every time a bit of sand is added to the system. In a burn-off game, the events are games (as measured by the number of cells that fire), and they can occur every time a chip is added to a cell.

In some systems, the relationship between the size and frequency of events in the system follows a *power law*; that is, if S is the size of an event, and F_S is the frequency of events of size S, then $F_S \approx C\gamma^S$ for some real constants C and γ. A power law relationship is a hallmark of a system that is in a self-organized critical state; on page 1 of [11], for example, the author states, "Although the dynamical response of the systems is complex, the simplifying aspect is that the statistical properties are described by simple power laws." The results in Figure 1.2 suggest this relationship for some $\gamma < 1$. We estimate γ using regression after considering an earthquake model due to Bak and Tang [3].

Their simple model works on a 50 × 50 lattice. The neighbors of each cell are defined to be those horizontally and vertically adjacent. A cell fires if its value is at least four, increasing the value of each neighbor by one while itself decreasing in value by four. Cells on the edge of the lattice operate as if an invisible border of cells surrounds the lattice: if a cell on the edge fires, one stress (or two stresses, in the case of corner cells) is (are) lost from the system. Figure 1.3 shows the results of 10,000 such games simulated with a Smallbasic program (the code appears in Appendix 2).

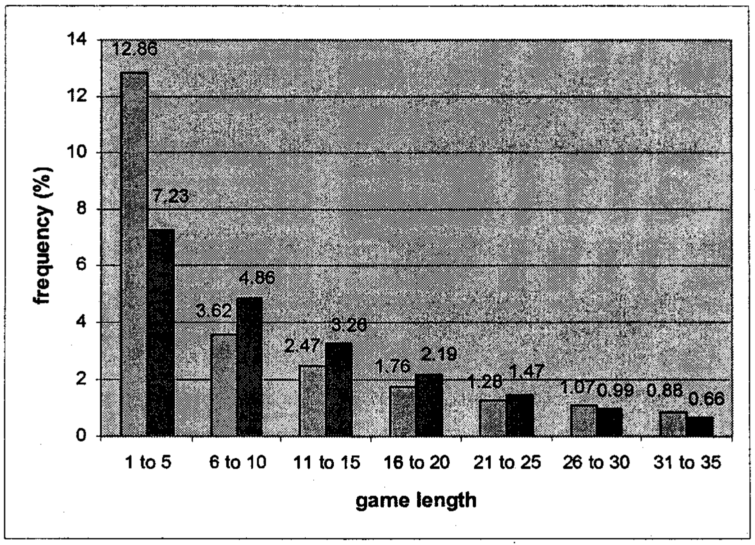

Figure 1.3. Results of 10,000 Bak games

Before the first game, every cell in the lattice receives between one and four stresses with uniform probability. In Section 2.3, we discuss an alternative method of initializing a game. Note that results for games of length zero and of length greater than 35 are omitted, and that the results are compressed into groups of five; both of these decisions were made simply to neaten the presentation. The lefthand bar in each pair shows the results from the simulation, while the righthand bar depicts the standard (least squares) exponential regression. For this data, the regression gives $C = 7.834$ and $\gamma = 0.924$, so $F_S \approx 7.834(0.924)^S$. The results in [3] exhibit the same behavior. The agreement between the simulated data and the regression is at best marginally satisfying. Applying the same analysis to the data in Figure 1.2, we find that $C = 39$ and $\gamma = 0.622$, so $F_S \approx 39(0.622)^S$ (see Figure 1.4). The results in this dissertation provide a way to predict the simulated data mathematically, and Figure 4.6 (p. 96) demonstrates the close agreement we can find using our methods.

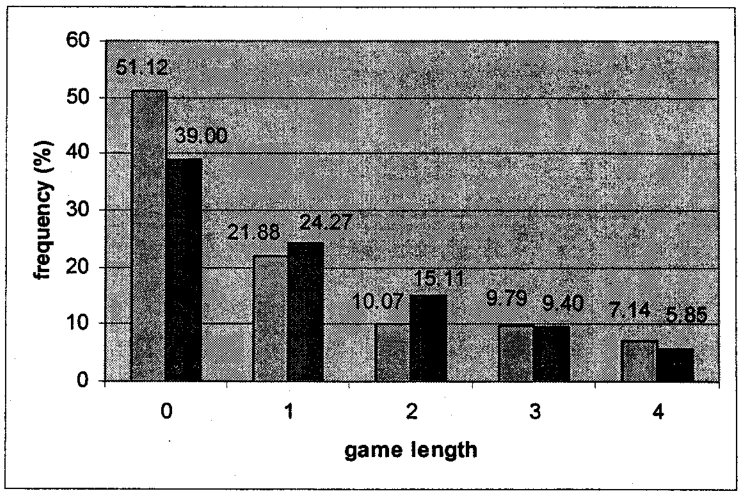

Figure 1.4. Results of 10,000 burn-off games, revisited

Since SOC was originally identified [3], its possible links with real-world phenomena have been widely explored. While earthquakes [3] and sandpile avalanches [7] are the events most commonly connected with SOC, investigation has also proceeded along less obvious lines. Nagel and Paczuski [15] created a simple traffic model and studied its self-organized critical behavior, using traffic jams as the "events" in their model. The model considers a single lane of traffic in which all cars move at or below the maximum allowable speed, responding to the car ahead according to preset rules of acceleration and deceleration. The model then introduces a small random element that interferes with the deterministic motion of the cars. Resultant traffic jams are noted, and the system is allowed to relax into its original deterministic state. The authors show that the frequency of the jams has a power law relationship with the size of the jams.

Stassinopolous and Bak [18] turned their attention to the human brain, modeling the brain as a graph in which neurons are nodes and their connections are edges. Each neuron possesses a firing threshold: when its neighbors fire, they add a charge to the neuron; if the charge exceeds the threshold, the neuron fires. An "event" in this model is a *thought*—if the proper neurons fire, the thought is a successful one, and the neurons that were involved in the thought are "rewarded" by having their thresholds decreased.

Liu *et al.* [14] investigate the fractal nature of the sandpile model studied in [3] when the model is allowed to relax from uniform initial conditions—in other words, when every cell in the sandpile begins at the same slope. The authors experiment with different boundary shapes (e.g., square, triangular) and different initial conditions. Despite the uniformity of the initial configurations, the sand exhibits behavior typical of systems in a SOC state.

As an example of how the theory of SOC has reached a non-mathematical audience, we note a recent article (2002) in *Atlantic Monthly* [16] that investigates a wide variety of systems demonstrating elements of SOC behavior, from persons of different ethnic backgrounds tending to live in ethnic neighborhoods, to rival religious groups with tendencies toward inter-group violence, to a possible explanation for the disappearance of the Anasazi culture of the southwestern United States in 1300 A.D.

1.5 SOC and burn-off games

As discussed in Section 1.4, a system that exhibits SOC should demonstrate a power law relationship between the frequency and size of the events in the system [1]. These systems can be modeled by chip-firing games, and the particular kind of chip-firing game in question in this work has the burn-off feature that not only more realistically models real-world phenomena, but also allows us to investigate mathematically whether such chip-firing games exhibit SOC. Later, we discover the closed formula (5.15) relating burn-off game length to frequency on complete graphs; restricting the model to complete graphs, however, produces analytic results that do not indicate the presence of SOC. Nevertheless, the model exhibits interesting properties of its own, which are the focus of this dissertation.

Jensen [11] devotes a chapter to a discussion of computer models of SOC. On p. 29, his list of features that characterize the models he studies are all characteristics of burn-off games. These features are: "the dynamical variable or field is updated in every time step according to some algorithm"; "the choice of the updating algorithm is, to some degree, arbitrary"; and "the criteria for choosing the relevant definitions are, for the most part, simplicity and intuition". A burn-off game exemplifies these features. Of attempts to formalize the study of SOC, such as [7], Jensen writes, "Despite their undeniable beauty, the exact solutions have one drawback: the specific mathematics tends to be tailored to the details of the solved model." In this dissertation, we find analytical results that apply to burn-off games played on any connected graph; although these results are not closed formulas, they are certainly not "tailored to the details" of the model. Our strictly mathematical approach adds a level of rigor to a field typically centering more on empirical results.

1.6 Outline

In Chapter 2, we show that burn-off games possess an important feature of other chip-firing games and discover an algorithm that helps determine all configurations of chips that can occur during a sequence of burn-off games on a connected graph. These results are crucial in subsequent chapters.

In Chapter 3, we investigate complete graphs and find a relationship between chip configurations that arise during burn-off games and spanning trees of related graphs. As an aside, we demonstrate how this result produces a new proof of Cayley's Theorem for enumerating the spanning trees of a complete graph.

Before we use the results of Chapter 3 to find a closed formula relating burn-off game lengths to frequency, we extend the results on complete graphs to connected graphs in Chapter 4. The chapter concludes with a method for determining the probability that a burn-off game on a connected graph will be of any given length. This method is used to generate analytically results like those in Figure 1.1, which were generated by computer simulation.

Finally, Chapter 5 contains the remaining results for burn-off games on complete graphs. The methods of Chapter 4 are shown to confirm what we discover about complete graphs in Chapter 3. We conclude with ideas for further research.

Chapter 2

Preliminary results

2.1 Introduction

This chapter establishes results that are fundamental to subsequent chapters. We first show that the order in which nodes are fired in a burn-off game is irrelevant to the final configuration of chips when the game ends. After discussing how we may characterize those relaxed configurations that can occur at the beginning of a sequence of burn-off games, we present an algorithm that recognizes all relaxed chip configurations that can occur during the sequence of games.

2.2 Burn-off games exhibit strong convergence

Burn-off games enjoy the property that all are of finite length. This is clear: after every turn, the total number of chips decreases by one, so eventually no node will be able to fire. We now show that burn-off games possess an additional important property called *strong convergence,* as defined in [9], namely, that nodes may be fired in any order without affecting the length or final configuration of a game.

Proposition 2.1 *In a burn-off game on a connected graph $G = (V,E)$, the nodes may be fired in any order without affecting the length or final configuration of the game.*

In our proof below, we follow the argument in [6], establishing the same conclusion for the chip-firing game considered there. The authors show that the "language" of such games possesses three properties that together imply their version of the assertion. In order to discuss the essential ingredients of their argument, we need to introduce some terminology and notation.

Label the nodes of G. It may be the case that a particular configuration on G can be played in more than one way. For each firing sequence, write down the labels that correspond to the nodes in the order that they are fired. The resulting sequence of labels is a *word,* and the set of all possible words, over all possible initial configurations, is a *language* $\mathcal{L}$. The *empty word* λ is a member of every language, corresponding to a game of length zero. A word need not bring a configuration to its eventual relaxed state.

Let $n = |V|$ be the number of vertices. If α is a word, we define its score vector $[\alpha] \in \mathbb{N}^n$ as follows:

$$[\alpha]_i = k \text{ if the node } i \text{ occurs } k \text{ times in } \alpha.$$

For example, suppose that a four-node graph has labels $\{1,2,3,4\}$. If $23424 \in \mathcal{L}$ (indicating that node 2 fires first, followed by node 3, and so on), then $[23424] = (0,2,1,2)$, because node 1 fires 0 times, node 2 fires 2 times, and so on. The issue here is whether a firing sequence different from 23424 could result in the same final configuration. Now we are ready to state the three aforementioned properties sufficient for Proposition 2.1.

Definition 2.2

(1) $\mathcal{L}$ is *left-hereditary* if, whenever a word belongs to $\mathcal{L}$, every initial segment of the word also belongs to $\mathcal{L}$. For example, if $23424 \in \mathcal{L}$, then so must λ, 2, 23, 234, and 2342 be elements of $\mathcal{L}$.

(2) $\mathcal{L}$ is *locally-free* if, for any $\alpha \in \mathcal{L}$ and any two different nodes x and y with $\alpha x \in \mathcal{L}$ and $\alpha y \in \mathcal{L}$, we also have $\alpha xy \in \mathcal{L}$. For example, if $2342 \in \mathcal{L}$, and both $23421 \in \mathcal{L}$ and $23424 \in \mathcal{L}$, then so also are 234214 and 234241 elements of $\mathcal{L}$.

(3) $\mathcal{L}$ is *permutable* if whenever $\alpha, \beta \in \mathcal{L}$, with score vectors $[\alpha] = [\beta]$, and $\alpha x \in \mathcal{L}$ for some node x, then $\beta x \in \mathcal{L}$. In other words, if one word is just a permutation of another, then both may be extended by the same symbol to obtain a new word of $\mathcal{L}$. For example, if $23424 \in \mathcal{L}$ and $22344 \in \mathcal{L}$, and 23424 may be extended to $234241 \in \mathcal{L}$, then 223441 is also an element of $\mathcal{L}$.

Proof of Proposition 2.1. We follow the strategy of [6], where the authors show that any two words (in a language possessing the properties of Definition 2.2) that have the same score will describe two different firing sequences, yet will result in the same final configuration. Thus, we verify that if $\mathcal{L}$ is the collection of all chip-firing words in a burn-off game, then $\mathcal{L}$ possesses the properties of Definition 2.2. Recall that in this collection we include all configurations that have not yet relaxed, but are merely on their way to a relaxed configuration.

That $\mathcal{L}$ is left-hereditary is clear: for a game to have progressed from one configuration to another necessarily means that all intermediate configurations must have also belonged to $\mathcal{L}$.

To see that $\mathcal{L}$ is locally free, consider a configuration that allows for two different nodes, x and y, to fire. Firing one of them (say, x) sends one chip to each neighbor of x (and burns one chip from the system). Thus, a node y that could have fired *before* x fired can certainly still do so, since the number of chips on y has either remained unchanged or increased by one.

Finally, to see that $\mathcal{L}$ is permutable, we argue that any two partial games corresponding to words α and β with the same score must lead to the same configuration. There are only two ways that the number of chips on a node x can change: if x fires, the number of chips on x decreases by $\deg(x) + 1$; if a neighbor of x fires, the number of chips on x increases by one. If two partial games α and β have the same score, then x fires the same number of times in each game, as do the neighbors of x. Thus, the configurations at the end of either partial game are identical. ■

The property that nodes may be fired in any order without affecting the length or final configuration of a burn-off game will be essential to almost every argument presented in this dissertation.

2.3 Definition of reverse-firing game

In the literature (e.g., [3], [14]), computer models of complex systems are initialized into a state that the authors assume will exhibit SOC as soon as the model operates on the system. For example, in Bak's paper [3], which studied SOC on a checkerboard-grid, the system was initialized by randomly assigning an integer larger than four (and smaller than some pre-determined upper bound) to each cell of the grid. Recall that in Bak's model a cell fires if its value exceeds four. Thus, this initialization makes it possible for *all* cells to fire. This grid is then relaxed until no cells can fire. Bak assumes that the resulting graph will immediately begin to exhibit SOC as the values in the cells begin to be randomly perturbed.

Define a configuration to be *supercritical* if every node is supercritical. In our analysis, we shall play burn-off games only on those configurations that can result from relaxing a supercritical configuration. In fact, our first task in the following analysis is to establish, for any given connected graph, how to determine if a given configuration of chips can indeed be such a result.

To determine those configurations that are the result of relaxed supercritical configurations, it is instructive to consider what happens when a chip-firing game is played in reverse. A *reverse-firing game* is defined so as to undo the firing rule of the chip-firing game under consideration. For example, recall the firing rule for the burn-off model: a node v may be fired only when $C(v) \geq \deg(v) + 1$, and when a node is fired, one chip from v is lost from the system. In a reverse-firing game, chips are *added* to the graph at each turn. The *reverse-firing rule* for the burn-off model is: select any node v such that all neighbors of v contain at least one chip; from each neighbor, move one chip onto v, then add a chip to v from outside the system.

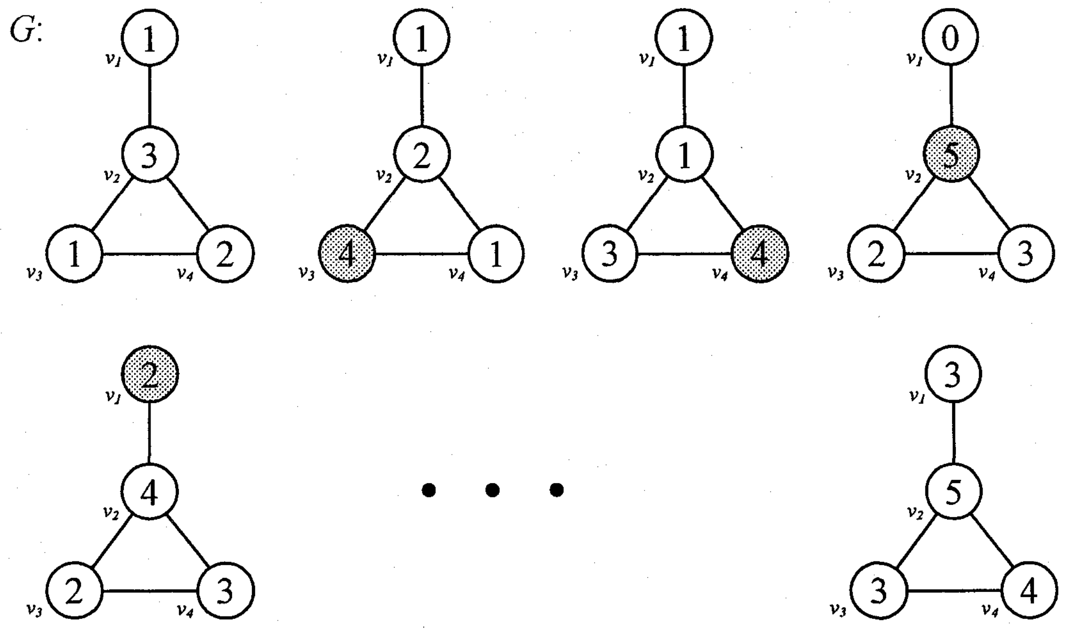

Figure 2.1. An instance of a reverse-firing game. The ellipsis indicates that the fifth configuration (lower left) continues to reverse-fire until the final supercritical configuration (lower right) is reached.

Example 2.3 Figure 2.1 shows this in action on a graph G. Suppose we begin a reverse-firing game on G by reverse-firing v_3. Nodes v_2 and v_4, the neighbors of v_3, both donate one chip to v_3, and another is added from outside the system, for a total of 4 chips on v_3. Of course, in the second configuration, v_3 is able to fire; if it did, the first configuration would be the result. The figure also displays the results if the nodes are reverse-fired in the order v_3, v_4, v_2, and v_1. In the fifth configuration, the nodes v_1, v_2, and v_4 are all able to fire. However, v_3 does not have enough chips to fire in a burn-off game. If we continued the reverse-firing game by reverse-firing the nodes again in the same order, the final configuration would be as shown in the figure. In this configuration, all nodes are supercritical. If this configuration were used to initialize G, then the *first* configuration in the figure would result from playing the burn-off game to relaxation.

While it is certainly possible that this reverse-firing game can continue indefinitely, we will only care to play such games until every node is able to fire. Any configuration, like the first one in Figure 2.1, that can be reverse-fired to a supercritical state shall be called *legal*. A *relaxed legal configuration* is a legal configuration in which no nodes may fire. Because the rest of this chapter is concerned with enumerating relaxed legal configurations, we let $L(G)$ denote the number of relaxed legal configurations on a graph G. Note that as we define L we drop the stipulation that G be connected. The flexibility allowed by dropping this stipulation will be useful in Theorem 4.9, which considers relaxed legal configurations on disconnected graphs. None of the subsequent results in this chapter require that G be connected.

2.4 Characterizing relaxed legal configurations

Not all relaxed configurations are legal; for example, a graph containing no chips is clearly relaxed, but just as clearly cannot be reverse-fired into a supercritical state. We are now ready to characterize the relaxed legal configurations on any given graph $G = (V,E)$. Our characterization uses the notation $1_A : V \to \{0,1\}$ to denote the *indicator function* of a subset $A \subseteq V$.

Proposition 2.4 *A relaxed configuration $C : V \to \mathbb{N}$ is legal if and only if there exists an ordered partition (I_k) of independent sets I_k of V so that, for each $v \in V$, if $v \in I_j$, then*

$$C(v) \geq \sum_{x \sim v} 1_{(x \in I_r \text{ and } r<j)}.$$

For convenience later, we call the property of $(C, (I_k))$ that every node v contains at least as many chips as it has neighbors that are members of earlier independent sets *Property P*.

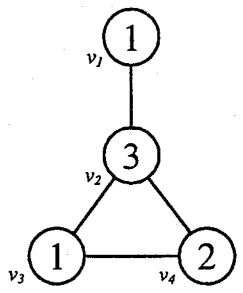

Figure 2.2. A legal configuration

Example 2.5 Let C be the configuration shown in Figure 2.2. We may assign the nodes to independent sets as follows: $I_1 = \{v_1, v_3\}$; $I_2 = \{v_4\}$; $I_3 = \{v_2\}$. Nodes v_1 and v_3 have no neighbors in earlier independent sets because they are members of I_1, the earliest independent set of all. Node v_4 has one neighbor, v_3, in an earlier independent set; since v_4 contains at least one chip, Property P is not violated at v_4. Node v_2 has three neighbors in earlier independent sets, and it contains three chips. Thus, Property P holds for this choice of $(C, (I_k))$.

Proposition 2.4 implies that since this configuration is relaxed, it is legal; that is, it may be reverse-fired to a supercritical configuration. As we saw in Example 2.3, this configuration is indeed legal. Now we verify (in general) that this legality is equivalent to the existence of a partition of V with Property P.

Proof of Proposition 2.4. Suppose that we have a relaxed legal configuration C. By definition, a legal configuration is one that can be reverse-fired into a configuration where all nodes are supercritical. Further, C is relaxed, so none of the nodes in G are supercritical. Thus, in any reverse-firing game that reveals a relaxed configuration to be legal, all nodes must reverse-fire (because reverse-firing is the only way for a node to gain chips during a reverse-firing game).

Consider any reverse-firing sequence that shows C to be legal. List only the first time each node reverse-fires during the game; suppose that the nodes are reverse-fired in the order $v_1, v_2, \ldots, v_n$. Put each node v_j into its own set I_j. Since each I_j contains only one node, each member of the ordered family (I_j) is independent. We now need to show that $(C, (I_j))$ possesses Property P.

Any given node v_j will reverse-fire only after each node $v_1, v_2, \ldots, v_{j-1}$ reverse-fires at least once. Each of these nodes that is a neighbor of v_j takes a chip from v_j when it reverse-fires. Since each earlier node is a member of an earlier independent set, v_j must contain at least as many chips in C as it has neighbors that are members of earlier independent sets.

To establish the converse, suppose that the nodes can be partitioned into r independent sets so that $(C,(I_k))$ possesses Property P. Reverse fire the nodes of I_k, for $k = 1,2,\ldots,r$, in that order (although within sets, the nodes may be fired in any order). We claim that this reverse-firing results in each node increasing its number of chips by one.

Consider a node $v \in I_j$, and let $C(v)$ be the number of chips on v before the reverse-firing process starts. Because of Property P, the node v contains at least as many chips as it has neighbors in all I_k with $k < j$. Suppose that there are s such neighbors. Each neighbor in these sets I_k with $k < j$ reverse-fires before v does, and each reverse-firing will pull one chip from v. This leaves $C(v) - s \geq 0$ chips on v. If any nodes that are in I_j reverse-fire before v, the number of chips on v is unaffected, since I_j is independent. When v reverse-fires, it pulls a chip from each of its $\deg(v)$ neighbors and receives one extra chip for the reverse burn-off. Finally, all nodes in sets I_m with $m > j$ reverse-fire, and each neighbor of v in these sets (say there are l of these) pulls a chip from v. Therefore, after each node has been reverse-fired once, the number of chips on v is decreased by $s + l = \deg(v)$ and increased by $\deg(v) + 1$ (while never becoming negative), for a net increase of one, as claimed.

Notice that reverse-firing each node once, as described in the preceding paragraph, preserves Property P. Thus, this process may be repeated until all nodes become supercritical. That is, if $t = \max_v\{\deg(v) - C(v)\}$, then repeating the process $t+1$ times will result in every node v containing at least $\deg(v) + 1$ chips. Therefore, the original configuration was legal. ■

2.5 Checking legality of any configuration on any graph

Given a configuration C (not necessarily relaxed) on a graph G, we may be interested in knowing if C is legal. Though we will not use it immediately, we take a short detour here to present an algorithm to answer this question. The algorithm will often be useful in Chapter 4 as we consider burn-off games on a connected graph. Our proof of the algorithm's efficacy leans on Proposition 2.4.

Algorithm 2.6 INPUT: a graph $G = (V,E)$ and a chip configuration $C : V \to \mathbb{N}$ on G;

OUTPUT: answer to question "Is C legal?"

```
(1) Find $v \in V$ such that $C(v) \geq \deg(v)$. If this cannot be done,
    then stop: $C$ is not legal. Otherwise, let $G^* = G$.

(2) Delete $v$ from $G^*$. If all nodes are now deleted, then stop:
    $C$ is legal. Otherwise, let $G^* = (V^*, E^*)$ be the new graph.

(3) Find $v \in V^*$ such that $C|_{V^*}(v) \geq \deg_{G^*}(v)$. If this cannot be
    done, then stop: $C$ is not legal. Otherwise, go to step (2).
```

Example 2.7 Figure 2.3 shows two ways in which Algorithm 2.6 might operate on the configuration depicted. In both passes from left to right, appropriate nodes are deleted until none remain. In the first step, the algorithm may delete either v_2 or v_4, as they both contain at least as many chips as their degree. At the end of either execution sequence, all nodes are deleted, so, by Proposition 2.7 below, the starting configuration is legal.

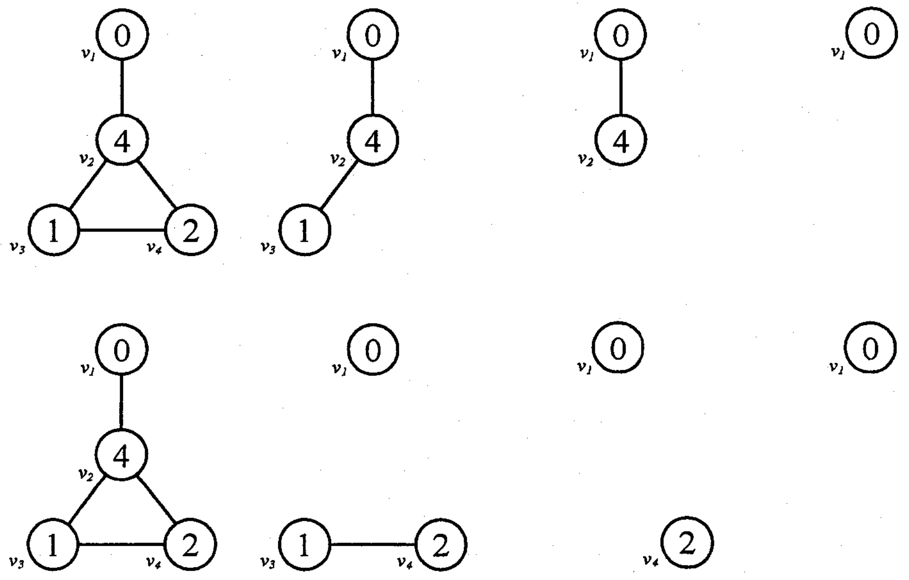

Figure 2.3. Two execution sequences of Algorithm 2.6 on the same initial configuration

In the proof of Proposition 2.9, we need the following result.

Lemma 2.8 *Let $C : V(G) \rightarrow \mathbb{N}$ be a configuration and G' a subgraph of G. If $C|_{V(G')}$ is not legal on G', then C is not legal on G.*

Proof. We will argue the contrapositive: if C is legal on G, then so must $C|_{V(G')}$ be legal on G'. By Proposition 2.4, the legality of C on G implies that it is possible to partition $V(G)$ into independent sets (I_k) enjoying Property P (see p. 25).

For each k, consider the partition $J_k = I_k \cap V(G')$ of $V(G')$. Collect all nonempty sets J_k into the ordered family (J_r) of independent sets, and focus on a node $v \in V(G')$. Since $(C,(I_k))$ has Property P, the node v (considered now in G) contains at least as many chips as it has neighbors (in G) that are members of earlier independent sets. In G', the node v may have fewer such neighbors, but it obviously cannot have more. Thus, $(C|_{V(G')},(J_k))$ has Property P, so $C|_{V(G')}$ is legal on G'. ■

Let $\mathcal{L}$ be the set of legal configurations on G. Now we are ready to establish the correctness of Algorithm 2.6.

Proposition 2.9 *Given a graph $G = (V,E)$ and a configuration $C : V \to \mathbb{N}$, Algorithm 2.6 correctly determines whether $C \in \mathcal{L}$.*

Proof. First, we show that if at any point during the operation of Algorithm 2.6 (say, when we have arrived at a subgraph G^*) every node v contains fewer than $\deg_{G^*}(v)$ chips, then the original configuration is not legal. Suppose, by way of contradiction, that an original configuration C^* leading to this situation on G^* is legal. By Lemma 2.8, $C^*|_{V(G^*)}$ is legal; then the nodes of G^* may be partitioned into independent sets (J_r) so that $(C^*|_{V(G^*)}, (J_r))$ has Property P. Consider a node v that is a member of the last set J_s. Since all of its neighbors are members of earlier independent sets, v contains at least $\deg_{G^*}(v)$ chips, contradicting our assumption in the first sentence. Thus, if every $v \in V(G^*)$ contains fewer than $\deg_{G^*}(v)$ chips, then C^* is not legal.

Second, we show that if the algorithm proceeds until all nodes are deleted, then C is legal. Suppose that the algorithm's deletion order is $v_n, v_{n-1}, \dots, v_1$. Place each node v_j into its own set I_j. For v_j to be deleted from G^* by the algorithm, it must contain at least as many chips as it has neighbors in G^*. Since every one of these neighbors is in an earlier independent set, our choice of $(C, (I_k))$ has Property P, and so C is legal. ■

2.6 The poset of configurations

Now that we have characterized legal configurations, we turn our attention to a result that allows us (in Chapter 4) to enumerate the relaxed legal configurations on any given connected graph. We assume the reader is familiar with the topic of posets as discussed, e.g., in [13].

Chip-firing games proceed in cycles beginning with the random placement of a stress (one chip) that may trigger a game of nontrivial length. We therefore consider configurations that arise from adding a single chip to a legal configuration. To this end, it is useful to consider the set $\mathcal{B}$ of all configurations on a fixed graph G as a poset $(\mathcal{B},\preceq)$ whose ordering relates to the numbers of chips on the nodes of G as follows: for $P, Q \in \mathcal{B}$ and $\leq$ the usual (total) ordering on $\mathbb{N}$, let

$$P \preceq Q \text{ iff each } v \in V(G) \text{ satisfies } P(v) \leq Q(v).$$

Recall that a *legal* configuration is one that can be reverse-fired to a supercritical configuration (see Section 2.3).

Proposition 2.10 *If P is a legal configuration, then any Q with $P \preceq Q$ is also legal.*

Proof. We clearly need only consider those configurations Q with $P \prec Q$. Such a Q has at least as many chips on any given node v as does P, and since $P \prec Q$, there exists a node x with $P(x) < Q(x)$. Starting from the configuration P, add one chip to x to create a new configuration P'.

Since P is legal, there exists a reverse-firing sequence that results in a supercritical configuration. "Freeze" the new chip on x, and carry out the same reverse-firing sequence starting with P'. The frozen chip will not affect the reverse-firing game (since it is frozen), and once P is reverse-fired to a supercritical configuration, the chip may be "thawed". The resulting supercritical configuration shows that P' is legal.

If $P' = Q$, the assertion is proved; if not, the argument above can be repeated with P' in the role of P. ■

Now suppose that a relaxed legal configuration Q has a chip added to a randomly selected node v, creating the configuration Q^+. By Proposition 2.10, we know that Q^+ is legal. Suppose that v may fire in Q^+, and let R be the relaxed configuration that results. Since R can be reverse-fired back to the legal configuration Q^+, we know that R is also legal. Thus, if we initialize a sequence of burn-off games by relaxing an arbitrary supercritical configuration, then all relaxed configurations that occur during the sequence of burn-off games will be legal.

We have demonstrated in this chapter how we may begin playing a series of burn-off games on a connected graph and how to recognize when a relaxed configuration is legal. This knowledge will be important in Chapter 4, where we analyze burn-off games on any given connected graph. First, we consider the special case of complete graphs in Chapter 3.

Chapter 3

Investigations on complete graphs

Before we investigate how the material in Chapter 2 leads to analytical results for general connected graphs, we pause to study complete graphs in this chapter. We find a pair of algorithms that give a one-to-one correspondence between relaxed legal configurations on K_n and spanning trees of K_{n+1}. These algorithms not only provide us with a proof of Cayley's Formula (see [19]), but also prepare us to discuss a similar, but more difficult, pair of algorithms in Chapter 4.

3.1 Statements equivalent to Property *P*

Proposition 2.4 established that a configuration C on a graph is legal if and only if the nodes can be partitioned into independent sets (I_r) so that $(C,(I_r))$

possesses Property P. Here we show that Property P, for *complete* graphs, may be reformulated in two ways that will be helpful later. We continue to let C denote a chip configuration, here on $V(K_n)$, and now define $q = (q_1, q_2, \dots, q_n)$ by $q_i = C(v_i)$.

Proposition 3.1 *Let $C : V \to \mathbb{N}$ be a configuration on K_n. The following statements are equivalent:*

(1) *For some partition of V into independent sets (I_r), the pair $(C, (I_r))$ satisfies Property P;*

(2) *For each $j = 1, 2, \dots, n$, at most j of the q_i are at most $j - 1$;*

(3) *For each $k = 1, 2, \dots, n$, at least k of the q_i are at least $n - k$.*

Proof of (1) $\Rightarrow$ (2). Suppose that $(C, (I_r))$ has Property P. Then for $s = 1, \dots, r$, every node in I_s contains at least as many chips as it has neighbors that are members of earlier independent sets. In a complete graph, the only way to partition the nodes into independent sets is to assign each node to its own set. Thus, each independent set contains just one node; label that node v_j for each $j = 1, 2, \dots, n$.

By Property P, each v_{j+1} contains at least j chips. The only nodes that might contain $j - 1$ chips or fewer are $v_1, v_2, \dots, v_j$; this implies that at most j nodes contain at most $j - 1$ chips, which establishes statement (2).

Proof of (2) $\Rightarrow$ (3). If at most j of the q_i are $j-1$ or less, then at least $n-j=k$ of the q_i are $j=n-k$ or more.

Proof of (3) $\Rightarrow$ (1). It will be convenient to relabel the nodes of K_n so that the index runs from from 0 to $n-1$ and $C(v_i) \geq C(v_{i+1})$ for $i = 0,1,\dots,n-2$. Define $p = (p_0,p_1,\dots,p_{n-1})$ by $p_i = C(v_i)$; observe that p is a permutation of q. By statement (3), at least k of the p_i are at least $n-k$, for $k = 1,2,\dots,n$. Because $(C(v_i))$ is nonincreasing, we have $C(v_j) = p_j \geq n-j-1$ for $j = 0,1,\dots,n-1$.

Let $I_j = \{v_{n-j-1}\}$ for $j = 0,1,\dots,n-1$; equivalently, $I_{n-j-1} = \{v_j\}$ for $j = 0,1,\dots,n-1$. Since each I_j contains a single node, it is independent. Since each $v_j \in I_{n-j-1}$ and the graph is complete, we see that v_j has $n-j-1$ neighbors in earlier independent sets. We have $C(v_j) \geq n-j-1$ from the preceding paragraph, so $(C,(I_j))$ has Property P. ■

Example 3.2 Consider K_6, and let $q = (5,2,1,5,0,2)$. Statement (2) shows this configuration to be *not* legal: let $j = 3$, and observe that the statement "at most 3 entries of q are at most 2" is false. Statement (3), with $k = 3$, is "at least 3 entries of q are at least 3," which is also false.

3.2 Enumerating relaxed legal configurations on a complete graph

Here we establish a formula for $L(K_n)$, the number of relaxed legal configurations C on a complete labeled graph. Our determination includes a parameter to bound the maximum value of $C(v)$ for $v \in V = V(K_n)$. For $n \geq 1$ and $m \geq n - 1$, let $L_{n,m}$ be the number of legal configurations satisfying $C(v) \leq m$ for each $v \in V$. For convenience, we also define $L_{0,m} = 1$ for all $m \geq 0$. In the proof of the next result, we shall find it convenient to use characterizations (2) and (3) in Proposition 3.1.

Theorem 3.3 *For all $n \geq 1$ and $m \geq n - 1$, we have* $L_{n,m} = (m - n + 2)(m + 2)^{n-1}$. (3.1)

Example 3.4 Suppose that we wish to find the number of legal configurations on K_4, where no node contains more than 5 chips, as in Figure 3.1. Note that the nodes v_1 and v_3 have enough chips to fire, so this is not a *relaxed* legal configuration. To count *all* relaxed legal configurations, we would fix m equal to the degree of each node in K_n, which is $n - 1$. Theorem 3.3 asserts that the number of legal configurations like the one in Figure 3.1 is $L_{4,5} = (5 - 4 + 2)(5 + 2)^{4-1} = 1029$.

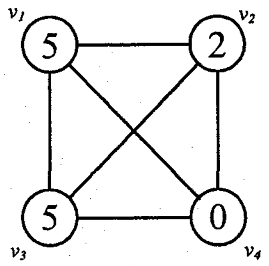

Figure 3.1. One of the 1029 legal configurations on K_4 for which no node contains more than 5 chips

Proof of Theorem 3.3. We use induction. Since $L_{0,m} := 1$ for all $m \geq 0$, this satisfies (3.1). For each $m \geq 0$, we extend our base case to include $L_{1,m}$, which counts the number of legal configurations on a single node. By condition (3) in Proposition 3.1, at least one node must contain at least zero chips. Thus, the number of chips occupying our single node lies in the set $\{0,1,\ldots,m\}$; so $L_{1,m} = m+1$, which satisfies (3.1). Finally, we observe that for $n \geq 2$, $L_{n,n-2}$ enumerates the legal configurations in which all n nodes contain at most $n-2$ chips; since condition (2) in Proposition 3.1 requires that at most $n-1$ nodes may contain at most $n-2$ chips in a legal configuration, we have $L_{n,n-2} = 0$ for all $n \geq 2$. This result satisfies (3.1), so we include it in our base case.

Now fix $n \geq 2$ and $m \geq n-1$ and assume that (3.1) is valid for each $L_{n-k,m-1}$ for $k = 0,1,\ldots n$. To determine $L_{n,m}$, we let $k \in [0,n]$ count the number of nodes containing exactly m chips; there are $\binom{n}{k}$ ways to choose these k nodes. Consider the configuration of chips on the remaining $n-k$ nodes of K_n. Focusing on these nodes, we see from condition (3) of Proposition 3.1, that for $l = 1, \ldots, n-k$, at least l of them contain at least $(n-k)-l$ chips. Thus, the configuration on the remaining $n-k$ nodes is a legal configuration on K_{n-k} with at most $m-1$ chips on each node. Since the number of such configurations is $L_{n-k,m-1}$, we have

$$L_{n,m} = \sum_{k=0}^{n} \binom{n}{k} L_{n-k,m-1}.$$

We may now apply our inductive hypothesis to simplify the sum:

$$\begin{aligned}
L_{n,m} &= \sum_{k=0}^{n} \binom{n}{k} L_{n-k,m-1} \\
&= \sum_{k=0}^{n} \binom{n}{k} ((m-1)-(n-k)+2)((m-1)+2)^{(n-k)-1} \\
&= \sum_{k=0}^{n} \binom{n}{k} \left(\tfrac{k}{n}[(m+1)-(m-n+1)] + (m-n+1)\right)(m+1)^{n-k-1} \\
&= \sum_{k=0}^{n} \left[\tfrac{k}{n}\binom{n}{k}(m+1) + \left(\tfrac{n-k}{n}\right)\binom{n}{k}(m-n+1) \right](m+1)^{n-k-1} \\
&= \sum_{k=1}^{n} \binom{n-1}{k-1}(m+1)^{n-k} + \sum_{k=0}^{n-1} \binom{n-1}{k}(m-n+1)(m+1)^{n-k-1} \\
&= (m-n+2)\sum_{k=0}^{n-1} \binom{n-1}{k}(m+1)^{(n-1)-k} \\
&= (m-n+2)((m+1)+1)^{n-1}.
\end{aligned}$$

■

As suggested in Example 3.4, $L_{n,n-1} = (n+1)^{n-1}$ gives the number of relaxed legal configurations for a burn-off chip-firing game on K_n. We consider this familiar expression again in the following section.

3.3 A connection between $L_{n,n-1}$ and Cayley's Formula

In the preceding section, we determined that $L(K_n) = (n+1)^{n-1}$. Of course, this is Cayley's Formula (see, e.g., [19]) for the number of spanning trees of K_{n+1}. Our next result provides a new proof of this formula.

Theorem 3.5 *The number of relaxed legal configurations on K_n equals the number of spanning trees of K_{n+1}.*

Proof. We establish injections between the set $\mathcal{R}$ of relaxed legal configurations on K_n and the set $\mathcal{S}$ of spanning trees of K_{n+1} by giving algorithms that, given a member of one set, generate a unique member of the other set. Define $A : \mathcal{R} \to \mathcal{S}$ via Algorithm 3.6 below and $B : \mathcal{S} \to \mathcal{R}$ via Algorithm 3.7. In both algorithms, *score* refers simply to a numeric label assigned to a node. Define $F : V(K_n) \to \mathbb{N}$ as the score function that makes this assignment.

Our first algorithm injectively maps $\mathcal{R}$ to $\mathcal{S}$.

Algorithm 3.6 INPUT: a complete graph K_n and relaxed legal configuration $q = (q_1, q_2, \ldots, q_n)$;
OUTPUT: a spanning tree of K_{n+1}.

```
(1) Fix a node $v_0$ in $K_{n+1}$.
(2) Label the remaining nodes $v_1, v_2, \ldots, v_n$.
(3) Let $M_0 = (v_0)$ and $\overline{M}_0 = \{v_0\}$.
(4) Let $F(v_0) = n - 1$.
(5) Let $Q_0 = 0$ and $i = 0$.
(6) Until all $v_k$ have been included in some sequence $M_i$, do the
    following:
    (a) $i + 1 \mapsto i$.
    (b) Let $Q_i = Q_{i-1} + |\overline{M}_{i-1}|$.
    (c) Let $M_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_t})$, for some $t \geq 1$, be the
        sequence (in increasing subscript order) of all nodes
        $v_k$ for which $q_k = F(u)$ for some $u \in \overline{M}_{i-1}$. Let
        $\overline{M}_i = \{x : x \text{ is an entry of } M_i\}$.
    (d) Add an edge from each $v_k$ in $\overline{M}_i$ to the node $w \in \overline{M}_{i-1}$
        for which $q_k = F(w)$.
    (e) For each $j = 1, 2, \ldots, |\overline{M}_i|$, let $F(v_{i_j}) = n - Q_i - j$.
```

Note: The purpose of the variables Q_i is to record the number of nodes that have been included in an earlier M_j. In fact, $Q_i = \left|\bigcup_{j=0}^{i-1} \overline{M}_j\right|$. Thus, in step (6e), each score is given to exactly one node. Therefore, "the node" selected in step (6d) is indeed unique. For an example trace of this algorithm, please see Example 3.8 starting on p. 47.

Proof that A is well-defined. Algorithm 3.6 will fail if, at any step, $\overline{M}_i$ is empty, because no further edges can then be added in step (6d). We demonstrate below that no $\overline{M}_i$ is ever empty, but assume for now that this is true. Step (6d) adds an edge from each $v_k \in \overline{M}_i$ to a node that is already part of a single growing component of the subgraph of K_{n+1} being constructed. The algorithm continues until all nodes of K_{n+1} are members of some $\overline{M}_i$, so all nodes of K_{n+1} are eventually connected to the growing component. Note also that exactly $\boldsymbol{n}$ edges are created by step (6d), one for each node except v_0. A spanning connected subgraph (of an $(n+1)$-node graph G) with n edges is necessarily a spanning tree of G; thus, the algorithm indeed constructs a spanning tree of K_{n+1}.

It remains to prove that no $\overline{M}_i$ is empty. We proceed by induction. It is clear in step (3) that $\overline{M}_0$ is not empty; suppose that $\overline{M}_{i-1}$ is not empty for some fixed $i > 0$. (By definition of $\overline{M}_{i-1}$, it is clear that $\overline{M}_{i-1}$ is also not empty.) As Algorithm 3.6 proceeds, the scores assigned to the nodes in step (6e) descend from $\boldsymbol{n} - 1$. We must show that at least one entry of q is large enough to equal the score of one of the nodes in $\overline{M}_{i-1}$. This will guarantee that $\overline{M}_i$ contains at least one element.

When, in step 6(c), the nodes are checked to see if they will be members of M_i, the algorithm inspects the scores assigned to the nodes in M_{i-1} (our induction hypothesis ensures that there are nodes in M_{i-1} to inspect). These scores were assigned (during the preceding iteration) in the following order: $n - Q_{i-1} - 1$, $n - Q_{i-1} - 2, \ldots, n - Q_{i-1} - |\overline{M}_{i-1}| = n - Q_i$. Thus, the lowest score assigned to a node of M_{i-1} is $n - Q_i$.

Since q is a legal configuration, at least Q_i entries of q are at least $n - Q_i$ (see Proposition 3.1). Since $Q_i - 1$ nodes (corresponding to entries of q) have been assigned scores (we subtract 1 because v_0 does not correspond to an entry of q), there is at least one unassigned node v_k containing at least $n - Q_i$ chips (i.e., $q_k \geq n - Q_i$); this number is at least as big as the lowest score found in M_{i-1}. We also know that q_k equals one of the scores assigned to a node in M_{i-1}, for if q_k exceeded all of those scores, then v_k would have already been assigned to an earlier sequence. It follows by induction that no $\overline{M}_i$ is empty and, by our earlier remarks, that A is indeed well-defined.

Proof that A is an injection. Let $q = (q_1, q_2, \ldots, q_n)$ and $q^* = (q_1^*, q_2^*, \ldots, q_n^*)$ be two distinct relaxed legal configurations on K_n. Let $V(K_{n+1}) = \{v_0, v_1, \ldots, v_n\}$. Let $A(q) = T$ and $A(q^*) = T^*$. We will prove that A is an injection by showing that T and T^* must be distinct.

As $q \neq q^*$, Algorithm 3.6 must encounter $q_i \neq q_i^*$ for some $i \in \{1,\ldots,n\}$. Since the scores are assigned in order of decreasing value, in the first such encounter either q_i or q_i^* will be the largest entry of its corresponding sequence. Without loss of generality, suppose that $q_i^* > q_i$; specifically, find $\max\{\max_{i\in\{1,\ldots,n\}}\{q_i, q_i^* : q_i \neq q_i^*\}\}$, and (if necessary) interchange the labels q and q^* so that q_i^* has this value. We will show that v_i has different neighbors in T and T^*, so that $T \neq T^*$. Suppose that as Algorithm 3.6 operates on q, the sequences constructed in step (6c) are $M_1,\ldots,M_j$, and suppose that as it operates on q^*, the sequences constructed in step (6c) are $M_1^*,\ldots,M_k^*$. Now suppose that $v_i \in \overline{M}_s$ (for some $s \in \{1,\ldots,j\}$) and $v_i \in \overline{M}_t^*$ (for some $t \in \{1,\ldots,k\}$). We consider two cases.

Case 1. $s = t$

Since q_i,q_i^* are the earliest unequal entries encountered, we have $M_{s-1} = M_{t-1}^*$. Since $q_i \neq q_i^*$, the node v_i must be assigned different neighbors, say x,y, from among the nodes of M_{s-1} and M_{t-1}^*, respectively. Then the edge $\{v_i,x\} \in E(T)\backslash E(T^*)$, implying that $T \neq T^*$.

Case 2. $s \neq t$

Algorithm 3.6 adds an edge between v_i and some node w^* in M_{t-1}^*. Since q_i,q_i^* are the earliest unequal entries encountered, we know that $M_{t-1} = M_{t-1}^*$. But we have assumed that $q_i^* > q_i$, so $t < s$; this implies that $M_{s-1} \neq M_{t-1}^*$. Thus, the edge $\{v_i,w^*\} \in E(T^*)\backslash E(T)$, again implying that $T \neq T^*$. ■

Our second algorithm injectively maps $\mathcal{S}$ to $\mathcal{R}$. Recall that $F : V(K_n) \to \mathbb{N}$ assigns a numeric score to each node of the complete graph on which the chip-firing games are taking place.

Algorithm 3.7 INPUT: a spanning tree of K_{n+1};
OUTPUT: a relaxed legal configuration $q = (q_1, q_2, \dots, q_n)$ on K_n.

```
(1) Fix a node $v_0$ in $K_{n+1}$.
(2) Label the remaining nodes $v_1, v_2, \dots, v_n$.
(3) Let $N_0 = (v_0)$, $\overline{N}_0 = \{v_0\}$, and $i = 0$.
Repeat:
    (4) $i + 1 \mapsto i$
    (5) Define $N_i = (v_{i_1}, v_{i_2}, \dots, v_{i_t})$, for $t \geq 1$, as the
        sequence (in increasing subscript order) of all
        unassigned nodes that are neighbors in $T$ of a node
        in $N_{i-1}$. Let $\overline{N}_i = \{x : x$ is an entry of $N_i\}$.
Until: all nodes have been assigned.
(6) Define $N = (u_k)_{k=1}^{n+1}$ as the concatenation of all the $N_i$'s, in
    their natural order.
(7) For $k = 1, 2, \dots, n$, set $F(u_k) = n - k$ (note that the $(n + 1)$st
    entry is not assigned a score).
(8) Let $q_i$ represent the number of chips on node $v_i$ of $K_n$.
    Determine the vector $q = (q_1, q_2, \dots, q_n)$ as follows:
    (a) For each $i = 1, 2, \dots, n$, $v_i$ is an entry of some $N_j$, and
        is thus the neighbor in $T$ of some $v_k \in \overline{N}_{j-1}$.
    (b) Let $q_i = F(v_k)$.
```

Note: In step (5) we are performing a breadth-first search (see [19]) from v_0 to determine the N_i's. For an example trace of this algorithm, see Example 3.9 starting on p. 50.

Proof that B is well-defined. Step (7) makes it clear that q is relaxed, since the highest score assigned to a node is $n-1$. In order to show that q is legal, we demonstrate that, for $l = 0, 1, \ldots, n-1$, at most $l+1$ entries of q could be assigned a value at most l (refer to Proposition 3.1). In step (7), a node v_k receives the score l after $(n-1)-l$ other nodes have been labeled. As v_k has n neighbors in K_{n+1}, it will have $n-((n-1)-l) = l+1$ unlabeled neighbors in T. These are the only nodes that can be assigned q-values at most $F(v_k) = l$ in step (8b). Thus, for $l = 0, 1, \ldots, n-1$, at most $l+1$ entries of q are assigned a value at most l. Proposition 3.1 implies that q is indeed legal. ■

Proof that B is an injection. Let T and T^* be two distinct spanning trees of K_{n+1}. Let $V(K_{n+1}) = \{v_0, v_1, \ldots, v_n\}$. Let $B(T) = q$ and $B(T^*) = q^*$. We will prove that B is an injection by showing that the configurations q and q^* must be distinct.

For $v \in V(T)$, let $\Gamma_T(v)$ denote the set of neighbors of v that are assigned the value $F(v)$ in step (8b); in other words, $\Gamma_T(v)$ is the set of nodes adjacent to, and one edge further from, v_0 in T. Define $\Gamma_{T^*}(v)$ analogously.

Since $T \neq T^*$, we know that $\Gamma_T(v) \neq \Gamma_{T^*}(v)$ for some $v \in V(K_{n+1})$. Let $r = \min\{i : \exists\, v \in \overline{N}_i \cap \overline{N}_i^* \text{ with } \Gamma_T(v) \neq \Gamma_{T^*}(v)\}$. Choose any $v \in \overline{N}_r \cap \overline{N}_r^*$ with $\Gamma_T(v) \neq \Gamma_{T^*}(v)$. Step (8b) assigns the value $F(v)$ to all $x \in \Gamma_T(v)$ and the value $F^*(v)$ to all $y \in \Gamma_{T^*}(v)$. But since r is a minimum, we have $F(v) = F^*(v)$; by step (7), we know that this score is assigned exclusively to v. To receive this score in step (8b), a node must be a member of either $\Gamma_T(v)$ or $\Gamma_{T^*}(v)$. Since $\Gamma_T(v) \neq \Gamma_{T^*}(v)$, we have $q \neq q^*$; thus, B is injective. ■

Since we have demonstrated injections between the set $\mathcal{R}$ of relaxed legal configurations on K_n and the set S of spanning trees of K_{n+1}, we have $|\mathcal{R}| = |S|$, which finally completes the proof of Theorem 3.5. ■

Notice that Theorems 3.3 and 3.5 together yield a new proof of Cayley's Formula. Indeed, the first of these implies that $|\mathcal{R}| = (n+1)^{n-1}$ (see the remark at the beginning of this section), while the second yields $|\mathcal{R}| = |S|$. Thus, $|S| = (n+1)^{n-1}$, or, as the result is more typically presented, the number of spanning trees of K_n is n^{n-2}.

Example 3.8 We illustrate Algorithm 3.6. Consider the configuration on K_5 shown in Figure 3.2. We know it is legal by Algorithm 2.6 (the verification algorithm could, e.g., successfully delete the nodes in the order x_4, x_5, x_2, x_1, x_3). For this configuration, $q = (1,3,1,4,3)$, and $n = 5$.

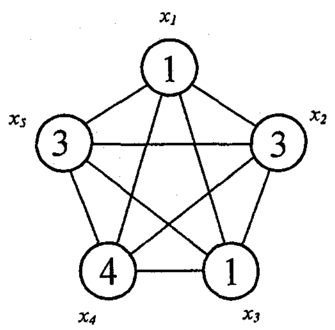

Figure 3.2. A legal configuration on K_5

Now we trace the execution of Algorithm 3.6 for this configuration as the input. Steps (1) through (5) result in the node labels $v_0,\dots,v_5$ (with v_0 fixed), $\overline{M}_0 = \{v_0\}$, $F(v_0) = n-1 = 4$, $i = 0$, and $Q_0 = 0$. Now we perform the steps in (6) until every v_k is a member of some $\overline{M}_i$ (see Figure 3.3). In the first iteration of step (6), we let $i = 1$, $Q_1 = Q_0 + |\overline{M}_{1-1}| = 0+1 = 1$, and $\overline{M}_1 = \{v_4\}$ (because $q_4 = 4$, the score assigned to $v_0 \in \overline{M}_0$). We therefore connect v_4 to v_0 with an edge, and v_4 is assigned the score $n - Q_1 - 1 = 5-1-1 = 3$. (See the leftmost graph in Figure 3.3.) In the second iteration of step (6), we let $i = 2$, $Q_2 = Q_1 + |\overline{M}_{2-1}| = 1+1 = 2$, and $\overline{M}_2 = \{v_2, v_5\}$, because $q_2 = q_5 = 3$, the score assigned to $v_4 \in \overline{M}_1$. We therefore connect both v_2 and v_5 to v_4, and v_2 is assigned the score $n - Q_2 - 1 = 5-2-1 = 2$, whereas v_5 is assigned the score $n - Q_2 - 2 = 5-2-2 = 1$. (See the rightmost graph in Figure 3.3.)

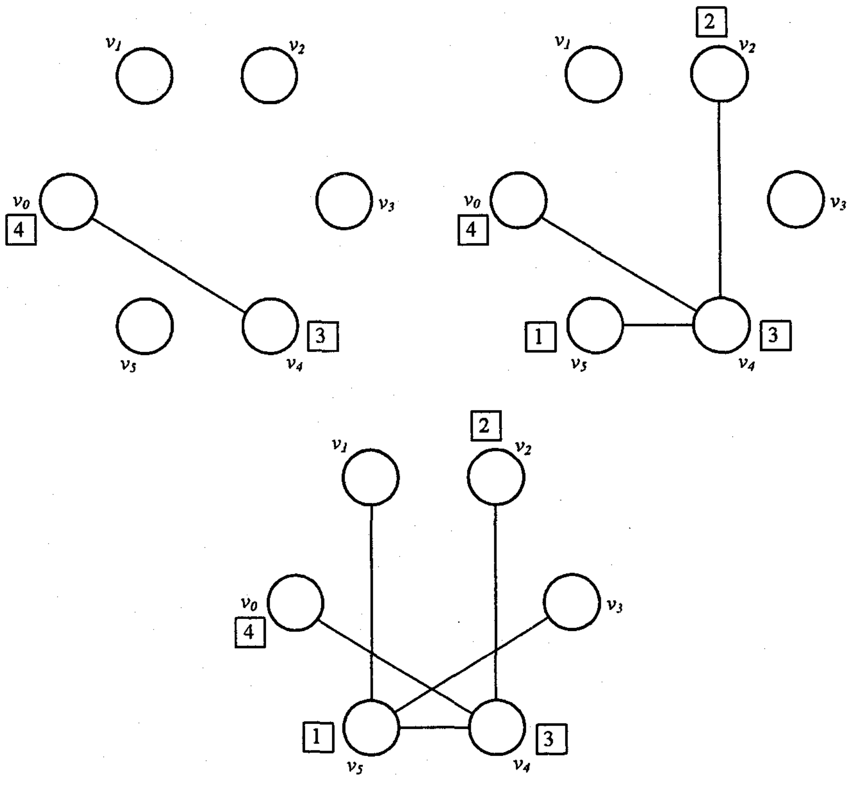

Figure 3.3. Three iterations on K_6 (the scores appear in the boxes)

In the final iteration, we let $i = 3$, $Q_3 = Q_2 + |\overline{M}_{3-1}| = 2 + 2 = 4$, and $\overline{M}_3 = \{v_1, v_3\}$, because $q_1 = q_3 = 1$, the score assigned to $v_5 \in \overline{M}_2$. We therefore connect both v_1 and v_3 to v_5, completing the spanning tree. We may assign scores to v_1 and v_3, but all nodes are now members of some M_i, so the algorithm terminates. The lower graph in Figure 3.3 shows the spanning tree T of K_6 associated with the legal configuration $q = (1,3,1,4,3)$ on K_5. The interested reader will find that if Algorithm 3.7 is given T, the output will be q, but we illustrate Algorithm 3.7 with a new example.

Example 3.9 Consider the spanning tree of K_7 shown in Figure 3.4. We trace the execution of Algorithm 3.7 when this tree is the input.

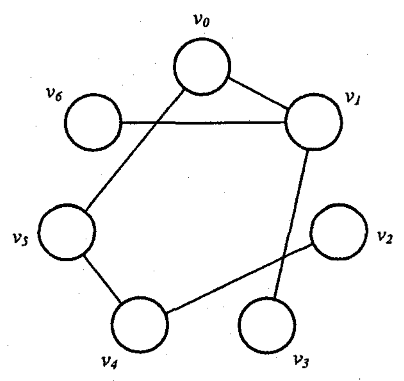

Figure 3.4. A spanning tree of K_7

Three iterations of steps (4) and (5) give us $N_0 = (v_0)$, $N_1 = (v_1, v_5)$, $N_2 = (v_3, v_4, v_6)$, and $N_3 = (v_2)$. Thus, $N = (v_0, v_1, v_5, v_3, v_4, v_6, v_2)$. Step (7) assigns scores to the nodes as shown in Figure 3.5.

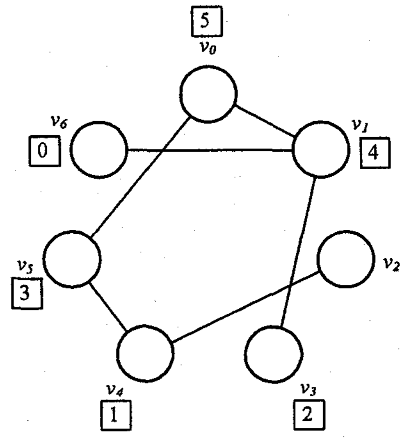

Figure 3.5. The same spanning tree with scores displayed

Step (8) assigns the value 5 to q_1, since $v_1 \in \overline{N}_1$, and its neighbor in $\overline{N}_0$ is v_0. Next, step (8) assigns the value 1 to q_2, since $v_2 \in \overline{N}_3$, and its neighbor in $\overline{N}_2$ is v_4. Continuing in this fashion, we find that the relaxed legal configuration on K_6 associated with the spanning tree of K_7 in Figure 3.4 is $q = (5, 1, 4, 3, 5, 4)$.

In Chapter 5, we will use Proposition 3.1, Theorem 3.3, and Theorem 3.5 to find the probabilities associated with each possible burn-off game length on a complete graph. First, however, we extend the results of this chapter to the more general case of a connected graph.

Chapter 4

Results for connected graphs

We will return to the preceding chapter's investigation of complete graphs in Chapter 5. In this chapter, we generalize the results of Chapter 3 to connected graphs. We begin by generalizing Theorem 3.5, and then discuss a method for enumerating the pairs of relaxed legal configurations and seeds that result in burn-off games of any desired length.

In Section 2.6, we saw that the relaxed configurations that occur in a sequence of burn-off games are legal. Theorem 4.13, which concludes the present chapter, determines the (stationary) probability distribution of these relaxed legal configurations. Thus, we consider a burn-off game as an experiment starting with a configuration C on a connected graph $G = (V,E)$ in which each node has uniform probability $1/|V|$ of being the seed; the resulting relaxed legal configuration is the outcome.

As we will see in Theorem 4.1, the set $\mathcal{R}$ of relaxed legal configurations on G is finite; therefore, we may consider the elements of $\mathcal{R}$ as states in a Markov chain (see, e.g., [10]). We employ this idea in Section 4.4, where we finally answer the question of whether there is a connection between burn-off games and SOC introduced in Section 1.4.

4.1 Enumerating relaxed legal configurations on any connected graph

A burn-off game played on a connected graph $G = (V,E)$ with a relaxed legal configuration consists of two phases: first, we choose a seed at random and place a chip on it; second, we allow the game to play until we obtain a relaxed legal configuration. We continue by choosing a seed in this configuration, and so on. In this section, we enumerate such configurations.

We create the graph $G^* = (V^*,E^*)$ by adding a new node x adjacent to every node in G. Specifically, $V^* = V \cup \{x\}$ and $E^* = E \cup (\bigcup_{v \in V} \{x, v\})$. Perhaps surprisingly, relaxed legal configurations on G are related to spanning trees of G^* just as in Theorem 3.5 for complete graphs.

Theorem 4.1 *The number of relaxed legal configurations on G is the number of spanning trees of G^*.*

We may use the Matrix Tree Theorem (as discussed, e.g., in [19]) to count the spanning trees of G^*.

Proof. We establish algorithmically injections back-and-forth between the set of relaxed legal configurations on G and the set of spanning trees of G^*. Let $\mathcal{R}$ be the set of relaxed legal configurations on G and $\mathcal{S}$ the set of spanning trees of G^*. Define $A : \mathcal{R} \to \mathcal{S}$ via Algorithm 4.2 below, and $B : \mathcal{S} \to \mathcal{R}$ via Algorithm 4.3 below.

Algorithm 4.2 INPUT: a connected graph $G = (V,E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, and a relaxed legal configuration $C : V \to \mathbb{N}$;

OUTPUT: $A(C)$, a spanning tree T^* of G^*.

```
(0) Let $T^*$ be the subgraph of $G^*$ with $V(T^*) = \{x\}$, $E(T^*) = \phi$.
(1) Let $i = 1$.
(2) Let $M_1$ be the sequence (in increasing subscript order) of
    nodes $v_k$ such that $C(v_k) = \deg_G(v_k)$; let
    $\overline{M}_1 = \{x : x$ is an entry of $M_1\}$.
(3) For each $v_k \in \overline{M}_1$, add $v_k$ to $V(T^*)$ and $\{x, v_k\}$ to $E(T^*)$.
    (If $V(T^*) = V$, then stop.)
(4) $i + 1 \mapsto i$.
(5) Let $M_i$ be the sequence (in increasing subscript order) of
    the nodes not yet included in $V(T^*)$ that are neighbors of
    nodes in $M_{i-1}$. Let $\overline{M}_i = \{x : x$ is an entry in $M_i\}$.

    For each $u \in \overline{M}_i$,
        execute steps (6) through (9):
        (6) For $r = 1, 2, \ldots, i-1$, let
            $N_r = (v_{r,1}, v_{r,2}, \ldots, v_{r,k_r})$ be the sequence (in
            increasing subscript order) of the $k_r$ G-neighbors
            of $u$ that appear in $\overline{M}_r$. Let
            $\overline{N}_i = \{x : x$ is an entry in $N_i\}$
        (7) Let $s = \left|\bigcup_{r=1}^{i-1} \overline{N}_r\right|$ and $N = (v_{1_1}, v_{1_2}, \ldots, v_{1_s})$ be the
            sequence determined by concatenating the
            sequences $N_1, N_2, \ldots, N_{i-1}$.
        (8) If $C(u) < \deg_G(u) - s$, then delete $u$ from $M_i$ and $\overline{M}_i$.
        (9) Otherwise, $C(u) = \deg_G(u) - j$ for some $j$ with
            $1 \le j \le s$. Add $u$ to $V(T^*)$ and $\{u, v_{1_j}\}$ to $E(T^*)$.
    (10) If $V(T^*) = V$, then stop. Otherwise, go to step (4).
```

Note the similarity between Algorithm 4.2 and Algorithm 3.6; for a complete graph K_n, adding a special node x adjacent to every other node simply creates K_{n+1}, and the output of Algorithm 3.6 is a spanning tree of K_{n+1}. However, the two algorithms will not necessarily produce the same outputs. In step 6(d) of Algorithm 3.6, our choice of edge depends on the sequence formed in step 6(c); this sequence is created by arranging nodes simply in increasing subscript order. Compare this approach to that of steps (6) and (7) of Algorithm 4.2, which impose greater restrictions on this sequence.

For an example trace of Algorithm 4.2, see Example 4.5 starting on p. 67.

Proof that A is well-defined. Not only must we be sure that Algorithm 4.2 outputs a spanning tree T^*, but also we must check that it does not halt before doing so. To establish both of these results, we look at each step in turn.

Step (2). We have already seen in Algorithm 2.6 that in a legal configuration, at least one node contains at least as many chips as its degree. Thus $\overline{M}_1$ is nonempty.

Step (3). It is clear that T^* is thus far a tree; in fact, it is a star.

Step (5). We must establish that $\overline{M}_i$ is nonempty so that the "for each $u \in \overline{M}_i$" step is not quantifying over an empty set. We proceed by induction. In the discussion of Step (2) above, we observed that $\overline{M}_1$ is nonempty. By construction, all nodes in $\overline{M}_1$ are critical. Because C is a legal configuration, we may apply Algorithm 2.6 to G and delete all of the nodes (in any order) in $\overline{M}_1$.

With these statements as our base case, our induction hypothesis is in two parts: for fixed $i > 1$, suppose that (a) $\overline{M}_1, \overline{M}_2, \ldots, \overline{M}_{i-1}$ are nonempty; and (b) we may apply Algorithm 2.6 to G and delete the nodes in $\overline{M}_1, \overline{M}_2, \ldots, \overline{M}_{i-1}$ without halting.

Let $M = \bigcup_{j=1}^{i-1} \overline{M}_j$. Lemma 2.8 states that the configuration on any subgraph of a graph (on which we have a legal configuration) must itself be legal. So, if our application of Algorithm 2.6 has deleted exactly the nodes of M, then at least one of the remaining nodes u of $G - M$ must be critical in $G - M$. Suppose that u is not a neighbor of any node in M. Because u is critical in $G - M$, and none of its neighbors have been deleted in our application of Algorithm 2.6, we see that u is also critical in G. But this places u in $\overline{M}_1$, which contradicts the choice of u in $G - M$.

Thus, we know that u is a neighbor of some node in M. Now if u is not a neighbor of a node in $\overline{M}_{i-1}$, it must be adjacent to, say, $s \geq 1$ nodes in $\overline{M}_1, \overline{M}_2, \ldots, \overline{M}_{i-2}$. Thus, u has been considered previously by step (8) and has been deleted each time. Therefore, $C(u) < \deg_G(u) - s$. This shows—back in our application of Algorithm 2.6—that if we have deleted all of the nodes in $\overline{M}_1, \ldots, \overline{M}_{i-1}$, including the s neighbors of u, then u will not be critical in $G-M$. This contradicts the fact that u is critical in $G-M$, so u must be a neighbor of a node in $\overline{M}_{i-1}$.

Because u is critical in $G-M$, step (8) will not delete u from $\overline{M}_i$. Thus, $\overline{M}_i$ is nonempty; this fulfills part (a) of the induction hypothesis. We claim that any node w placed in $\overline{M}_i$ by step (5) will survive past step (8) only if it, too, is critical in $G-M$. For w to survive step (8), we require $C(w) \geq \deg_G(w) - s$, where s is the number of G-neighbors of w that appear in M. Since $\deg_G(w) - s$ simply equals $\deg_{G-M}(w)$, we know w is critical in $G-M$. Thus, all nodes in $\overline{M}_i$ may be deleted as we apply Algorithm 2.6. This fulfills part (b) of the induction hypothesis.

Step (6). Step (5) assures us that these neighbors exist.

Step (8). The argument given above for step (5) assures us that $\overline{M}_i$ remains nonempty after all nodes of $\overline{M}_i$ have been processed in step (8).

Step (9). It is impossible to create a cycle in this step because step (5) only considers those nodes that are not yet part of T^*.

Step (10). This step assures us that T^* will be a spanning tree of G^*.

Observe that step (9) adds at least one edge to T^* since $\overline{M}_i$ remains nonempty. Once $n-1$ edges have been added to T^*, step (10) will halt the algorithm. Since A does not halt until it outputs a spanning tree T^*, A is well-defined. ■

Proof that A is an injection. Let $C \in \mathcal{R}$ and $C' \in \mathcal{R}$ be two distinct relaxed legal configurations on G. We prove that A is an injection by showing that the spanning trees $A(C)$ and $A(C')$ must be distinct. As Algorithm 4.2 operates on C and C', it must encounter a node v for which $C(v) \neq C'(v)$. Let us call such a v *special.* Step (8) might remove v from consideration; if this occurs for both inputs C and C', then we consider a future pass of the algorithm. Because Algorithm 4.2 includes every node in the output before it halts, we know that eventually we will find a special node that is not removed by step (8) concurrently for both inputs C and C'.

Now if v is removed by step (8) for one input but not the other, then step (9) will connect v to a different neighbor for the two inputs. On the other hand, suppose v is not removed by step (8) for either input; because $C(v) \neq C'(v)$, step (9) will connect v to a different neighbor for the two inputs. In either case, $A(C)$ and $A(C')$ must be distinct, and A is an injection. ■

Before we turn to Algorithm 4.3, we recall one definition. Let $G = (V,E)$

be a connected graph, and $u, v \in V$. As in [19], we define the distance from u to v (denoted $d_G(u,v)$) as the least length of a u,v-path.

Algorithm 4.3 INPUT: a spanning tree T^* of G^* and an ordering $V = (v_1, v_2, \ldots, v_n)$ of the nodes of G;
OUTPUT: $B(T^*)$, a relaxed legal configuration $C : V \to \mathbb{N}$ on G.

```
(1) Let $M_0 = (x)$.

(2) Let $m = \max_{v \in V}\{d_{T^*}(x, v)\}$. For $j = 1, 2, \ldots, m$, let $M_j$ be
    the sequence (in breadth-first order, breaking ties
    lexicographically by subscript) of nodes $v$ for which
    $d_{T^*}(x, v) = j$. Let $\overline{M_j} = \{x : x$ is an entry of $M_j\}$.

(3) For each $u \in \overline{M_1}$, let $C(u) = \deg_G(u)$.

For $i = 2, 3, \ldots, m$,

    For each $u \in \overline{M_i}$, following the ordering in $M_i$,

        execute steps (4) through (7):

        (4) For $r = 1, 2, \ldots, i-1$, let
            $N_r = (v_{r,1}, v_{r,2}, \ldots, v_{r,k_r})$ be the sequence (in their
            $M_r$-ordering) of the $k_r \geq 0$ $G$-neighbors of $u$ that
            appear in $M_r$. Let $\overline{N_r} = \{x : x$ is an entry of $N_r\}$.

        (5) Let $s = \left|\bigcup_{r=1}^{i-1} \overline{N_r}\right|$.

        (6) Let $N = (v_{h_1}, v_{h_2}, \ldots, v_{h_s})$ be the sequence
            determined by concatenating the sequences
            $N_1, N_2, \ldots, N_{i-1}$.

        (7) For some $t \in \{1, 2, \ldots, s\}$, we have $(v_{h_t}, u) \in E(T^*)$. Let
            $C(u) = \deg_G(u) - t$.
```

Note the similarity between Algorithm 4.3 and Algorithm 3.7, where a spanning tree of K_{n+1} is the input and a configuration on K_n is the output. However, the breadth-first order imposed by step (2) of Algorithm 4.3 is more restrictive than the search performed by steps (4) and (5) of Algorithm 3.7. Thus, given the same input, the two algorithms do not necessarily produce the same outputs.

For an example trace of Algorithm 4.3, see Example 4.6 starting on p. 69.

Proof that B is well-defined. In step (2), we partition V into sequences $M_1, M_2, \dots, M_m$. Step (3) assigns chips to the nodes in $\overline{M}_1$, while step (7) assigns chips to the nodes in $\overline{M}_2, \dots, \overline{M}_m$. Therefore, Algorithm 4.3 at least produces a function $C : V \rightarrow \mathbb{N}$.

Now we use Algorithm 2.6 to establish that C is legal. Since T^* is a spanning tree of G^*, we know that $\overline{M}_1$ is nonempty (see step (3)); hence, there is at least one node u such that $C(u) = \deg_G(u)$. Thus Algorithm 2.6, given C as input, may delete the nodes in $\overline{M}_1$. This fact is the base case in an induction argument that proves that in Algorithm 2.6, the nodes in $\overline{M}_1, \overline{M}_2, \dots, \overline{M}_m$ can be deleted in the order given by this list. Suppose that this is true for $\overline{M}_1, \overline{M}_2, \dots, \overline{M}_{k-1}$, where $2 \leq k < m$. For any $u \in \overline{M}_k$, step (7) assigns $C(u) = \deg_G(u) - t \geq \deg_G(u) - s$. Recall that s counts the neighbors in G of u that are in $\bigcup_{r=1}^{i-1} \overline{M}_r$; in our induction hypothesis, we have assumed that these neighbors have been deleted from G, resulting, say, in a subgraph G'. If other nodes in $\overline{M}_k$ have been deleted before we consider u, then

$\deg_{G'}(u)$ does not increase. Thus, we have $C|_{V(G')}(u) \geq \deg_{G'}(u)$, so u can be deleted in step (2) of Algorithm 2.6. ∎

Proof that B is an injection. Suppose that $T_1^*, T_2^* \in \mathcal{S}$ satisfy

$$C_1 := B(T_1^*) = B(T_2^*) =: C_2 \;(:= C);$$

we'll show that then $T_1^* = T_2^*$.

Let $V = V(G)$, $n = |V|$, and write the breadth-first orderings of V determined during the computation of $B(T_1^*)$ and $B(T_2^*)$ as $(u_i)_{i=1}^n$ and $(w_i)_{i=1}^n$, respectively. To complete the proof, we shall find it useful to establish the following lemma.

Lemma 4.4 *Under the hypothesis that $C_1 = C_2$, if there exists an integer $j \geq 1$ such that $u_i = w_i$ for all $i \in \{1, \ldots, j\}$, then the subtree H_1^* of T_1^* induced on $\{x, u_1, \ldots, u_j\}$ is identical to the subtree H_2^* of T_2^* induced on $\{x, w_1, \ldots, w_j\}$.*

Proof. We induct on j. First note that H_1^*, H_2^* are indeed subtrees of T_1^*, T_2^*, respectively, since the sequences (u_i), (w_i) are defined by breadth-first search on these trees. It is also clear from the definitions of (u_i), (w_i) that u_1, w_1 are adjacent to x in H_1^*, H_2^*, respectively. In the case when $j = 1$, these subtrees both consist of 2-vertex trees containing the edge $\{x, u_1\} = \{x, w_1\}$ and are therefore identical.

Now fix $j > 1$, assume that the lemma holds for smaller instances of j, and suppose that $u_i = w_i$ for all $i \in \{1,\dots,j\}$. Let G_0^* denote the subgraph of G^* induced on the common vertex set $U := \{x,u_1,\dots,u_j\}$ of H_1^*, H_2^*, and let $G_0 = G_0^* - x$. We consider four executions of Algorithm 4.3; in each case, the input vertex ordering is inherited from G.

The first pair of executions computes $D_1 := B(H_1^*)$ and $D_2 := B(H_2^*)$, two configurations on G_0. Since $(u_i)_{i=1}^j$, $(w_i)_{i=1}^j$ are initial segments of (u_i), (w_i), it is evident from Algorithm 4.3 that D_1, D_2 are obtained from C_1, C_2 by replacing $\deg_G$ in steps (3), (7) by $\deg_{G_0}$ and restricting the resulting functions to U. Since $C_1 = C_2$, we have $D_1 = D_2$. For $k = 1,2$ and for each node $u \in V(G_0)$, let $t_k(u)$ denote the value of t in step (7) as Algorithm 4.3 determines $D_k(u)$; if $D_k(u)$ is determined in step (3), we take $t_k(u) := 0$. Then

$$D_k(u) = \deg_{G_0}(u) - t_k(u) \qquad \text{for } k = 1,2 \text{ and each } u \in V(G_0). \tag{4.1}$$

The second pair of executions computes $D_1' := B(H_1^* - u_j)$ and $D_2' := B(H_2^* - w_j)$, two configurations on $G_0' := G_0 - u_j = G_0 - w_j$. For $k = 1,2$ and for each node $u \in V(G_0')$, define $t_k'(u)$ analogously with $t_k(u)$; now we have

$$D_k'(u) = \deg_{G_0'}(u) - t_k'(u) \qquad \text{for } k = 1,2 \text{ and each } u \in V(G_0'). \tag{4.2}$$

Since $(u_i)_{i=1}^j$, $(w_i)_{i=1}^j$ are respectively breadth-first orderings of $V(H_1^*)$, $V(H_2^*)$, the sequences $(u_i)_{i=1}^{j-1}$, $(w_i)_{i=1}^{j-1}$ are such orderings of $V(H_1^* - u_j)$, $V(H_2^* - w_j)$.

Thus, during the second pair of executions of Algorithm 4.3 described above, every sequence M_i (in the statement of the algorithm) is the same as during the first pair of respective executions, except, in passing from the first pair to the second, the final node of M_m (resp. u_j, w_j) has been deleted. Therefore

$$t'_k(u) = t_k(u) \qquad \text{for } k = 1,2 \text{ and each } u \in V(G'_0). \tag{4.3}$$

Since $D_1 = D_2$, the equations in (4.1) imply that

$$t_1(u) = t_2(u) \qquad \text{for each } u \in V(G_0). \tag{4.4}$$

Comparing (4.4) with (4.3), we see that

$$t'_1(u) = t'_2(u) \qquad \text{for each } u \in V(G'_0). \tag{4.5}$$

It follows from (4.2), (4.5) that $D'_1 = D'_2$. Since these are configurations on G'_0, whose vertex set is $U \setminus \{u_j\} = U \setminus \{w_j\}$, the induction hypothesis implies that $H_1^* - u_j = H_2^* - w_j$. Finally, from (4.4), we have $t_1(u_j) = t_2(u_j)$, and in Algorithm 4.3, this means that the node $u_j = w_j$ has the same neighbor in $H_1^* - u_j$ as in $H_2^* - w_j$. Therefore $H_1^* = H_2^*$. ■

It follows from Lemma 4.4, with $j = n$, that if (u_i) and (w_i) agree entirely, then $T_1^* = T_2^*$. Thus, it remains only to address the case when $u_i \neq w_i$ for some $i \in \{1,\dots,n\}$, and here we'll reach a contradiction.

First, notice that according to Algorithm 4.3, for any $u \in V$, we have

$C(u) = \deg_G(u)$ if and only if u is adjacent to x in both of T_1^*, T_2^*. Therefore, T_1^*, T_2^* do not differ in their adjacencies to x, and the sequences (u_i), (w_i) agree in their initial entries, corresponding to the (necessarily nonempty) neighbor sets of x in T_1^*, T_2^*. If there are L such neighbors, then $u_i = w_i$ for $i \in \{1, \ldots, L\}$, and we're assuming that $L < n$.

Let i_0 denote the least i such that $u_i \neq w_i$. Since $L < i_0 \leq n$, it is easy to see that Algorithm 4.3 reaches step (7) in defining $C_1(u_{i_0})$ and $C_2(w_{i_0})$. Let $j = i_0 - 1$, and define H_1^*, H_2^* as in the statement of Lemma 4.4. Since

$$u_i = w_i \qquad \text{for } i \in \{1, \ldots, j\}, \tag{4.6}$$

Lemma 4.4 shows that $H_1^* = H_2^*$. From (4.6), we also see that w_{i_0} does not appear in the subsequence $(u_i)_{i=1}^j$, and u_{i_0} does not appear in the subsequence $(w_i)_{i=1}^j$. Thus, in computing $B(T_1^*)$, Algorithm 4.3 processes u_{i_0} before w_{i_0}, while in computing $B(T_2^*)$, it processes u_{i_0} after w_{i_0}.

Now consider the instants during the two executions of Algorithm 4.3 when step (7) defines $C_1(u_{i_0})$ and $C_2(u_{i_0})$. In particular, for $k = 1, 2$, define t_k as in the proof of Lemma 4.4, so that

$$C_k(u_{i_0}) = \deg_G(u_{i_0}) - t_k(u_{i_0}) \qquad \text{for } k = 1, 2.$$

Since $C_1 = C_2$ by hypothesis, we have

$$t_1(u_{i_0}) = t_2(u_{i_0}) \tag{4.7}$$

As Algorithm 4.3 executes on T_1^* and is processing $u = u_{i_0}$, denote the sequence N in step (6) by N_1. Likewise, during execution on T_2^* and while processing the same node, denote the corresponding sequence by N_2. The entries of N_1 are the G-neighbors of u_{i_0} lying (strictly) closer to x in T_1^* than u_{i_0}. Similarly, the entries of N_2 are the G-neighbors of u_{i_0} lying closer to x in T_2^* than u_{i_0}. Since $H_1^* = H_2^*$, the sequence N_1 forms an initial segment of the sequence N_2. It follows from this and (4.7) that the T_1^*-neighbor of u_{i_0} closer to x (than u_{i_0}) in T_1^* and the T_2^*-neighbor of u_{i_0} closer to x in T_2^* are the same. A similar argument shows that the T_1^*- and T_2^*-neighbors of w_{i_0} closer to x (than w_{i_0}) in these trees are identical. Under these conditions, Algorithm 4.3 necessarily processes u_{i_0} and w_{i_0} in the same order during the computations of $B(T_1^*)$, $B(T_2^*)$. But we concluded two paragraphs earlier that this is not the case. This contradiction shows that the case when $u_i \neq w_i$ for some $i \in \{1, \dots, n\}$ is impossible and therefore completes the proof. ■

Example 4.5 Consider the legal configuration shown in Figure 4.1. We show how Algorithm 4.2 associates this configuration with a spanning tree T^* of G^*.

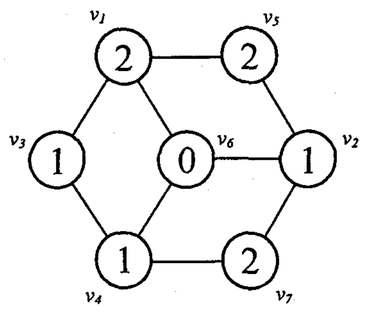

Figure 4.1. A legal configuration used to illustrate Algorithm 4.2

The two critical nodes are v_5 and v_7, so $M_1 = (v_5, v_7)$; the edges $\{x, v_5\}$ and $\{x, v_7\}$ are added to T^*. The nodes adjacent to those in M_1 comprise $M_2 = (v_1, v_2, v_4)$. Inspecting v_1 we find that $N = (v_5)$, so $s = 1$. Since $C(v_1) = 2 = 3 - 1 = \deg(v_1) - s$, step (9) adds the edge $\{v_1, v_5\}$ to T^*. Inspecting v_2 we find that $N = (v_5, v_7)$, so $s = 2$. Now $C(v_2) = 1 = \deg(v_2) - s$, so $C(v_2) = \deg(v_2) - 2$; step (9) adds the edge $\{v_2, v_7\}$ to T^*. (Note that if $C(v_2)$ were 2 rather than 1, step (9) instead would have added the edge $\{v_2, v_5\}$ to T^*.) Inspecting v_4 we find that $N = (v_7)$, so $s = 1$. However, $C(v_4) = 1$ and $\deg(v_4) - s = 3 - 1 = 2$; since $C(v_4) < 2$, step (8) deletes v_4 from M_2.

The (unused) nodes adjacent to those in $M_2 = (v_1, v_2)$ are $M_3 = (v_3, v_6)$. Inspecting v_3 we find that $N = (v_1)$, so $s = 1$. Since $C(v_3) = 1 = \deg(v_3) - s$, step (9) adds the edge $\{v_3, v_1\}$ to T^*. Inspecting v_6 we find that $N = (v_1, v_2)$, so $s = 2$. However, $C(v_6) = 0$ and $\deg(v_6) - s = 3 - 2 = 1$; since $C(v_6) < 1$, step (8) deletes v_6

from M_3. The (unused) node adjacent to the one in $M_3 = (v_3)$ is $M_4 = (v_4)$.

Inspecting v_4 we find that $N = (v_7, v_3)$. Note the order in which the two elements of N appear; this occurs because step (7) specifies that when $i = 4$, the order of the neighbors as they appear in M_1, M_2, M_3 must be preserved. Since each feature of the algorithm has now been illustrated in this example, we conclude simply by mentioning that the edges $\{v_3, v_4\}$ and $\{v_4, v_6\}$ are added to T^*. Figure 4.2 shows the output of Algorithm 4.2.

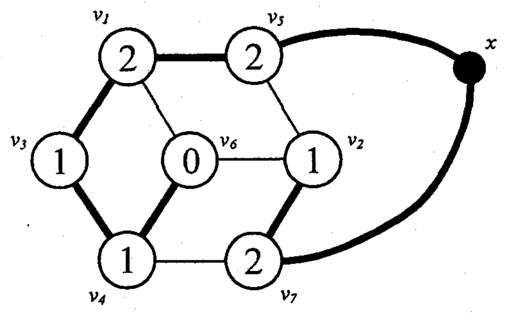

Figure 4.2. The spanning tree T^* associated (by Algorithm 4.2) with the legal configuration in Figure 4.1

Example 4.6 Consider the spanning tree T^* of G^* shown in Figure 4.3. We show how Algorithm 4.3 associates this spanning tree with a legal configuration C on G.

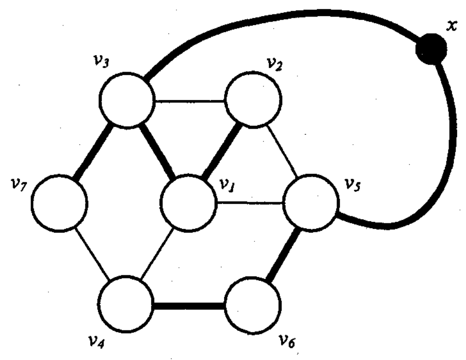

Figure 4.3. A spanning tree of G^* used to illustrate Algorithm 4.3

Step (2) gives $M_1 = (v_3, v_5)$, $M_2 = (v_1, v_7, v_6)$, and $M_3 = (v_2, v_4)$. Note how the requirements in step (2) affect the subscript order in M_2. Step (3) assigns $C(v_3) = \deg_G(v_3) = 3$ and $C(v_5) = \deg_G(v_5) = 3$. Inspecting v_1, we find $N = (v_3, v_5)$. Since the edge $\{v_1, v_3\}$ is in T^*, we have $t = 1$, so step (7) assigns $C(v_1) = \deg_G(v_1) - t = 4 - 1 = 3$. (Note that if the edge $\{v_1, v_5\}$ had been in T^* instead, step (7) would have assigned $C(v_1) = 2$.) For v_7 and v_6, step (7) assigns $C(v_7) = 1$ and $C(v_6) = 1$. At this point i becomes 3. Inspecting v_2, we find $N = (v_3, v_5, v_1)$. Note the order in which the elements of N appear; this occurs because step (6) specifies that the order of the neighbors as they appear in M_1, M_2 must be preserved. Since the edge $\{v_2, v_1\}$ is in T^*, we have $t = 3$, so step (7) assigns $C(v_2) = \deg_G(v_2) - t = 3 - 3 = 0$. Step (7) also assigns $C(v_4) = 1$, since $\{v_4, v_6\} \in E(T^*)$. Figure 4.4 shows the output of Algorithm 4.3.

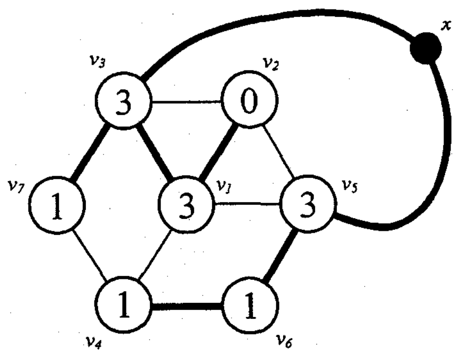

Figure 4.4. The legal configuration on G associated (by Algorithm 4.3) with the spanning tree of G^* shown in Figure 4.3

4.2 An extension of Theorem 4.1

By deleting certain edges from G^* (defined in Section 4.1), we may enumerate the relaxed legal configurations that correspond to games of length zero on G. This result will be useful in Chapter 5, but we mention it here because it requires Algorithm 4.2. For $v \in V$, let T_v denote the number of spanning trees of $G^* - xv$. Let $(C, v) \in \mathcal{R} \times V$ denote a choice of a relaxed legal configuration C on G and a seed $v \in V$.

Proposition 4.7 *The number of pairs (C, v) that result in a game of length zero is $\sum_{v \in V} T_v$.*

Proof. As shown in the discussion of Algorithm 4.2, an edge $\{x, v\}$ in T^* forces v to be critical in the corresponding relaxed legal configuration, whereas v will specifically *not* be critical when that edge is missing in T^*. So by removing the edge $\{x, v\}$ from G^* and enumerating the spanning trees, we will count the number of relaxed legal configurations in which v is not critical. Now if v is the seed, it will not fire, and the game length will be zero. We sum over all $v \in V$ to count all the pairs (C, v) that result in a game of length zero. ■

We will require Proposition 4.7 in Chapter 5 due to the unusual nature of games of length zero; in all other games, at least one node must fire, so the seed must be critical. A game of length zero requires a seed that is not critical, making games of length zero a special case.

4.3 Enumerating games of any length on any connected graph

Before we consider a theorem that allows us to enumerate the burn-off games of any length on any connected graph G, we prove that there are a finite number of game lengths to investigate.

Lemma 4.8 *During a burn-off game that starts with a relaxed legal configuration on* $G = (V,E)$*, no node may fire more than once.*

Proof. Let v be the seed. When v fires, it loses all of its chips. For v to fire a second time, it must gain at least $\deg(v)+1$ chips from its neighbors. By the pigeonhole principle, this will happen only if at least one neighbor of v fires at least twice. Thus, v cannot be the first node to fire more than once.

Suppose that $w \in V$, with $w \neq v$, is the first node to fire a second time. Say there are

$$c \leq \deg(w) \tag{4.8}$$

chips on w when the seed is chosen, and k neighbors of w have fired once before w fires for the first time. Just before w fires, it will contain $c+k > \deg(w)$ chips. After w fires, it will contain $c+k-(\deg(w)+1)$ chips. For w to fire again it must contain at least $\deg(w)+1$ chips, so it must gain at least $t := \deg(w)+1-(c+k-(\deg(w)+1))$ chips from its neighbors. But

$$t = (\deg(w)-k)+(\deg(w)-c)+2 > \deg(w)-k,$$

by (4.8). Now $\deg(w)-k$ is precisely the number of neighbors of w that did not fire before w fired. So for w to gain $t > \deg(w)-k$ chips, either one of these neighbors must fire at least twice before w fires a second time, or one of the k neighbors that fired before w fired must fire a second time. Either case contradicts our assumption that w was the first node to fire a second time. ■

So given any connected graph $G = (V,E)$, we know that the length g of any burn-off game must satisfy $0 \leq g \leq |V|$. While the relatively simple methods of

Section 4.2 do not extend to enumerating games of any length, the methods introduced in this chapter nevertheless give formulas for the number of such games.

First we need some new notation. For $v \in V$, let $\mathcal{T}_{v,g}$ denote the set of subtrees of G with g nodes and including v. For a graph X with subgraph X_1 and a node $u \in V(X)$, let $\Gamma_{X_1}(u)$ be the set of neighbors of u that lie in $V(X_1)$. For a connected graph $G = (V,E)$, let $(C,v) \in \mathcal{R} \times V$ denote a choice of relaxed legal configuration C and seed v. Finally, we note that this proof considers legal configurations on disconnected graphs and remind the reader that the function L enumerating relaxed legal configurations may operate on such graphs (see p. 24).

Theorem 4.9 *The number of pairs* (C,v) *resulting in a game of length* $g > 0$ *is* $\sum_{v \in V} \sum_{T \in \mathcal{T}_{v,g}} L(G - T)$.

Proof. For $v \in V$, let $\mathcal{R}_{v,g}$ denote the set of relaxed legal configurations on G such that if v is seeded, then the resulting burn-off game will be of length g. For $R_1, R_2 \in \mathcal{R}_{v,g}$, define the relation $\simeq$ as follows: suppose that when v is seeded in R_1 and R_2, the nodes that fire in each game induce the same subgraph H of G; suppose also that $R_1|_H = R_2|_H$. If both of these conditions hold, we say $R_1 \simeq R_2$. It is clear that $\simeq$ is an equivalence relation on $\mathcal{R}_{v,g}$. Let $Q_{v,g}$ be the set of equivalence classes of

$\mathcal{R}_{v,g}$. To prove Theorem 4.9, it will be helpful to establish injections $A : \mathcal{T}_{v,g} \to Q_{v,g}$ and $B : Q_{v,g} \to \mathcal{T}_{v,g}$.

Define $A : \mathcal{T}_{v,g} \to Q_{v,g}$ as follows. Let $T \in \mathcal{T}_{v,g}$, and let H be the subgraph of G induced on $V(T)$. Create H' as follows: to each $u \in V(T)$, append $\deg_G(u) - \deg_H(u)$ leaves to u. Let J be this set of leaves. Now let T' be the spanning tree of H' consisting of T and J. Create T^* by appending the node x and the edge $\{x, v\}$ to T'. Use T^* (with H' as the underlying graph) as the input in Algorithm 4.3; let C^* be the output configuration. Let Q be a configuration on G defined by $Q(v) = C^*(v)$ and $Q(u) = C^*(u) + 1$ for each $u \in V(H) - v$. Let Z be any relaxed legal configuration on $G - H$. Define $Q(w) = Z(w)$ for each $w \in V(G - H)$. Now Q is a configuration on G. We demonstrate below that $Q \in \mathcal{R}_{v,g}$; thus, we may let $\overline{Q}$ denote the equivalence class of Q. Finally, let $A(T) = \overline{Q}$.

Claim 1. A is well-defined.

Proof of claim. To show that $Q \in \mathcal{R}_{v,g}$, we will demonstrate that (a) Q is a relaxed legal configuration on G; and (b) seeding v in Q results in a burn-off game of length g.

(a) *Q is a relaxed legal configuration on G.*

Because v is the only neighbor of x in T^*, only v is critical in C^* (see step

(7) of Algorithm 4.3). As we define Q, then, adding a chip to each $u \in V(T) - v$ does not make any of these nodes supercritical. We choose Z to be any relaxed legal configuration on $G - T$, so none of the nodes in $V(G - T)$ are supercritical. Therefore, Q is relaxed.

We appeal to Algorithm 2.6 to demonstrate the legality of Q. We defined C^* using Algorithm 4.3, so C^* is a legal configuration on H'. Thus, if Algorithm 2.6 operates on C^*, it will provide a deletion sequence S of $V(H')$. Since every $w \in J$ is a leaf, each $\deg_{H'}(w) = 1$. Since only v is critical in C^*, we must have $C^*(w) = 0$. Without loss of generality, then, we may permute S so that $V(H)$ is processed before J and see that this new deletion sequence S' also satisfies the requirements of Algorithm 2.6. In passing from C^* to Q, we let $Q(v) = C^*(v)$ and $Q(u) = C^*(u) + 1$ for each $u \in V(H) - v$. Because $\deg_{H'}(x) = \deg_G(x)$ for every $x \in V(H)$, Algorithm 2.6 may begin to process Q on G in the same order found in the initial subsequence of S' containing the nodes of $V(H)$. Since we extended Q to $V(G - T)$ by choosing any legal configuration Z on the subgraph $G - T$, Algorithm 2.6 may finish processing Q, thereby confirming the legality of Q.

(b) *Seeding v in Q results in a game of length g.*

We first show that each node in T fires, and then show that none of the nodes in $G - T$ fire. Since T has g nodes, and by Lemma 4.8 no node may fire twice, the resulting game will be of length g.

Clearly, v can fire. For $u \in V(T) - v$, let

$$S_u = \{w \in \Gamma_H(u) : d_{H'}(w,x) < d_{H'}(u,x)\}$$

and $s_u = |S_u|$. By step (7) of Algorithm 4.3, we have

$$C^*(u) \geq \deg_{H'}(u) - s_u = \deg_G(u) - s_u.$$

We defined $Q(u) = C^*(u) + 1$, so once the nodes in S_u fire, the number of chips on u will be at least $\deg_G(u) + 1$, allowing u to fire as well.

For $w \in V(G - T)$, let $s_w = |\Gamma_T(w)|$. In each relaxed legal configuration Z on $G - T$, we must have $Z(w) \leq \deg_{G-T}(w) = \deg_G(w) - s_w$. Because the nodes in T contribute a total of s_w chips to w once they have all fired, the number of chips on w will never exceed $\deg_G(w)$. Since we define $Q(w) = Z(w)$, we know w will not fire when v is the seed.

We have shown that Q is a relaxed legal configuration on G such that if v is seeded, the resulting game will have length g; thus, we know that $Q \in \mathcal{R}_{v,g}$. Hence, A is well-defined. ■

Claim 2. *A is injective.*

Proof of claim. We will show that for distinct trees $T_1, T_2 \in \mathcal{T}_{v,g}$, we have $A(T_1) \neq A(T_2)$. For this argument, we let Q_{T_1}, Q_{T_2} denote one of the relaxed legal configurations on G that result as we find $A(T_1), A(T_2)$ respectively. (Note that $A(T_1)$

does not equal Q_{T_1} but rather $\overline{Q}_{T_1}$; similarly, $A(T_2) = \overline{Q}_{T_2}$.)

First suppose that T_1 and T_2 contain the same g nodes. Because T_1 and T_2 share the same vertex set, we know H_1 and H_2 will be identical. The creation of H'_1 (and H'_2) does not involve the structure of T_1 (and T_2), so H'_1 and H'_2 will be identical as well. Consequently, we know $J_1 = J_2$, which implies that what makes T_1^* and T_2^* distinct is the distinct structures of T_1 and T_2. When we use T_1^* and T_2^* as inputs to Algorithm 4.3, the injective nature of the algorithm implies that C_1^* and C_2^* will be distinct; thus, $Q_{T_1}|_{T_1}$ and $Q_{T_2}|_{T_2}$ will be distinct. Because $V(T_1) = V(T_2)$, we have $\overline{Q}_{T_1} \neq \overline{Q}_{T_2}$. Thus, $A(T_1) \neq A(T_2)$.

Now suppose that T_1 and T_2 do not contain the same g nodes, and that $A(T_1) = A(T_2) = \overline{Q}$ for some $\overline{Q} \in Q_{v,g}$. When we showed above that A is well-defined, we saw that seeding v in Q results in a game in which precisely the nodes in the underlying tree fire. But the original trees T_1, T_2 considered in this case are distinct. The deterministic nature of burn-off games, discussed in Proposition 2.1, prohibits this result; the same set of nodes must fire in any burn-off game played on a given configuration with seed v. Thus, $A(T_1) \neq A(T_2)$. ■

Having established that $A : \mathcal{T}_{v,g} \to Q_{v,g}$ is a well-defined injection, we turn our attention to showing the same is true of $B : Q_{v,g} \to \mathcal{T}_{v,g}$, defined as follows. Let $\overline{Q} \in Q_{v,g}$, so that $Q \in \overline{Q}$. Let H denote the subgraph induced on the nodes that fire if

v is seeded in Q.

Because $Q \in \overline{Q}$, seeding v in Q results in a burn-off game in which the nodes of H fire. With $h = |V(H) - v|$, let $F = (v, u_1, u_2, \ldots, u_h)$ be such a firing sequence of $V(H)$. For $m = 1, \ldots, h$, let l_m denote the number of H-neighbors of u_m that precede u_m in F. At the time u_m fires, it must contain at least $\deg_G(u_m) + 1$ chips, so

$$Q(u_m) \geq \deg_G(u_m) + 1 - l_m.$$

This inequality is clearly equivalent to

$$Q(u_m) \geq |\Gamma_{G-H}(u_m)| + \deg_H(u_m) + 1 - l_m,$$

and since $\deg_H(u_m) \geq l_m$, we may subtract $|\Gamma_{G-H}(u_m)|$ from the right-hand side without it becoming negative. On the left-hand side, subtracting $|\Gamma_{G-H}(u_m)|$ amounts to removing that many chips from u_m. Let Q_H^* denote the configuration on H that results if, for each $m = 1, \ldots, h$, we remove $|\Gamma_{G-H}(u_m)|$ chips from u_m. Thus, we have

$$Q_H^*(u_m) \geq \deg_H(u_m) + 1 - l_m, \qquad \text{for all } m = 1, \ldots, h.$$

Since $\deg_H(u_m) \geq l_m$, we may remove one additional chip from each $w \in V(H) - v$. Let Q_H denote the resulting configuration on H, so that

$$Q_H(u_m) \geq \deg_H(u_m) - l_m, \qquad \text{for all } m = 1, \ldots, h.$$

Note that v is the only node in $V(H)$ that is critical in Q_H.

Our intention is to input the graph H and the configuration Q_H into Algorithm 4.2. The algorithm requires that H is connected and Q_H is a relaxed legal configuration. Since H is a subgraph of G induced on the nodes that fire during a burn-off game, H is connected. Our choice of Q comes from an equivalence class of the relation $\simeq$ on $\mathcal{R}_{v,g}$, so Q is a relaxed configuration on G. For each $u \in V(H)$, we remove $|\Gamma_{G-H}(u)|$ chips from u, so Q_H^* is a relaxed configuration on H. In creating Q_H from Q_H^*, we remove a chip from each $w \in V(H) - v$, so Q_H is a relaxed configuration on H.

Finally, we appeal to Proposition 2.4 to show that Q_H is a legal configuration on H. Partition $V(H)$ into independent sets as follows: for $t = 0, \ldots, h-1$, let $I_t = \{u_{h-t}\}$, and let $I_h = \{v\}$. For all $t = 0, \ldots, h-1$, those H-neighbors of u_{h-t} that follow u_{h-t} in F appear in $\bigcup_{r=0}^{t-1} I_r$. Therefore,

$$Q_H(u_{h-t}) \geq \deg_H(u_{h-t}) - l_{h-t} = \sum_{w \sim u_{h-t}} 1_{(w \in I_r \text{ and } r<t)}, \qquad \text{for all } t = 0, \ldots, h-1.$$

It is easy to check that the analogous inequality holds for v, and this proves that Q_H is legal.

We apply Algorithm 4.2 with the connected graph H and the relaxed legal configuration Q_H on H. The algorithm outputs a spanning tree T^* of H^*. Because v is the only node in $V(H)$ that is critical in Q_H, the only node adjacent to the special node x in H^* is v. Let $T = T^* - x$. Finally, define $B(\overline{Q}) = T$. This tree is clearly a member of $\mathcal{T}_{v,g}$, so B is well-defined.

Claim 3. B is injective.

Proof of claim. We will show that for distinct $\overline{Q}, \overline{Q'} \in \mathcal{Q}_{v,g}$, we have $B(\overline{Q}) \neq B(\overline{Q'})$. Let Q, Q' be representatives of $\overline{Q}, \overline{Q'}$ respectively. Let H, H' denote the subgraphs induced on G by the g nodes that fire when Q, Q' respectively are seeded at v.

First, we consider the case where $H = H' =: H_0$. Because $\overline{Q}$ and $\overline{Q'}$ are distinct, we know $Q|_{H_0} \neq Q'|_{H_0}$. Therefore, Q_{H_0} and Q'_{H_0} will be distinct relaxed legal configurations on H_0. The injective nature of Algorithm 4.2 ensures that $B(\overline{Q}) \neq B(\overline{Q'})$.

Second, we consider the case where $H \neq H'$. When each of these subgraphs is used as the underlying graph in an iteration of Algorithm 4.2, the output is a spanning tree of that subgraph (with the edge $\{v,x\}$, which we subsequently delete). Since $H \neq H'$, these two trees must be distinct, so $B(\overline{Q}) \neq B(\overline{Q'})$. ■

Assisted by the following claim, finally, we'll be able to turn our attention to the inner sum that appears in the statement of Theorem 4.9. Given $T \in \mathcal{T}_{v,g}$, let us denote $A(T)$ by $\overline{Q}_T$.

Claim 4. For each $T \in T_{v,g}$, we have $|\overline{Q}_T| = L(G - T)$.

Proof of claim. Because $\overline{Q}_T$ is an equivalence class of the relation $\simeq$ on $\mathcal{R}_{v,g}$, it collects all relaxed legal configurations that agree on $V(H)$. Thus, two elements of $\overline{Q}_T$ can differ only on $V(G-T)$. By Lemma 2.8, the legality of $Q \in \overline{Q}_T$ on G implies the legality of $Q|_{G-T}$ on $G-T$. Hence, $|\overline{Q}_T| \leq L(G-T)$. Let L represent any relaxed legal configuration counted by $L(G-T)$, and use L for Z in the definition of $A(T)$ (p. 74). This has the effect of extending L to the rest of G using $Q|_T$, which is common to all $Q \in \overline{Q}_T$. Because we used $A : \mathcal{T}_{v,g} \to Q_{v,g}$ in bringing about this extension, the resulting configuration is legal on G. Since this extension is clearly injective, we have $|\overline{Q}_T| \geq L(G-T)$. ■

To complete the proof of Theorem 4.9, it suffices to show that for each $v \in V$, the number $\mathcal{N}$ of relaxed legal configurations C that result in a game of length g when seeded at v is $\mathcal{S} := \sum_{T \in \mathcal{T}_{v,g}} L(G-T)$. From the definition of $\mathcal{R}_{v,g}$, we have $\mathcal{N} = |\mathcal{R}_{v,g}|$, so it remains to show that $|\mathcal{R}_{v,g}| = \mathcal{S}$.

Since both of A,B are injections, and both of $\mathcal{T}_{v,g}, Q_{v,g}$ are finite sets, it follows that A is in fact a bijection. (The same is true of B, but we won't need this fact.) Thus, as T runs through $\mathcal{T}_{v,g}$, its image $A(T) = \overline{Q}_T$ runs through $Q_{v,g}$, and it follows that

$$|\mathcal{R}_{v,g}| = \sum_{\overline{Q} \in Q_{v,g}} |\overline{Q}| = \sum_{T \in \mathcal{T}_{v,g}} |\overline{Q}_T| = \sum_{T \in \mathcal{T}_{v,g}} L(G-T) = \mathcal{S},$$

where Claim 4 justifies the third equation. ■

Example 4.10 We illustrate Theorem 4.9 on the graph G in Figure 4.5. Here we consider a game of length three that starts with v as the seed. One of the subtrees $T \in \mathcal{T}_{v,3}$ is shown with bold lines. To apply Theorem 4.9 in evaluating $L(G-T)$, we add the node x to $G-T$, resulting in the graph on the right.

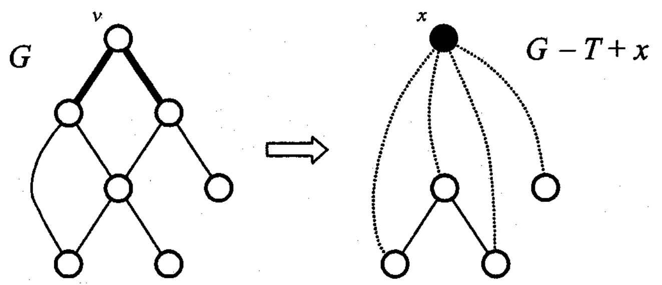

Figure 4.5. Determination of one term in the sum in Theorem 4.9

Theorem 4.1 depends on Algorithm 2.6, which may take disconnected graphs as input (see p. 24). By Theorem 4.1, then, we have $\tau(G-T+x) = L(G-T)$. Using the Matrix Tree Theorem (or by inspection), we find that $L(G-T) = 8$. Thus, this particular choice of subtree through v adds 8 legal configurations to the sum in Theorem 4.9. This is just one of the subtrees contributing to the inner sum; there are four more such subtrees.

4.4 All relaxed legal configurations are equally likely

Theorem 4.9 does not by itself accomplish our goal of mathematically predicting whether a sequence of burn-off games will exhibit the size versus frequency relationship exhibited by a system in a state of SOC. The results in this section give us the last tool we need in our pursuit of this goal. Recall that for a connected graph $G = (V,E)$ we denote the set of relaxed legal configurations on G by $\mathcal{R}$.

Proposition 4.11 *For $C \in \mathcal{R}$ and $v \in V$, there exists at least one $C^* \in \mathcal{R}$ such that if C^* is seeded at v, then the resulting relaxed configuration is C.*

Proof. If removing a chip from v in C results in a legal configuration C^*, then C^* has the desired property, for if C^* is seeded at v, then the resulting configuration is immediately C after a game of length zero.

So we may suppose either that $C(v) = 0$ or that removing a chip from v in C creates an illegal configuration in G. In either case, we define C^* by first defining a subgraph Y of G. If $C(v) = 0$, then let Y be the subgraph induced on v alone. Now suppose that $C(v) > 0$. Because C is a legal configuration, we may use Algorithm 2.6 to delete all of the nodes in some sequence according to the conditions specified in this algorithm. Since removing a chip from v creates an illegal configuration on G (see the preceding paragraph), there exists a set of nodes $X = \{v, y_1, y_2, \ldots, y_k\}$, with $k \geq 1$, that induces a connected subgraph of G where only v is critical in $C|_X$. Let Y be the subgraph induced on the largest such set; the arguments below do not require this choice to be unique.

Let $C^*(v) = \deg_G(v)$. For each $u \in V - v$, let

$$C^*(u) = (C(u) - |\Gamma_G(u) \cap Y|)(\bmod \deg_G(u) + 1). \qquad (4.9)$$

Note that

$$C(w) \geq |\Gamma_G(w) \cap Y| \text{ for } w \notin V(Y) \qquad (4.10)$$

because otherwise we must include w in $V(Y)$. We claim that C^* has the desired property. To establish this claim, we will show that (a) C^* is a legal configuration; and (b) if C^* is seeded at v, the resulting relaxed configuration is C.

(a) *C^* is a legal configuration.*

Since C is legal, using Algorithm 2.6 we may delete the nodes in V in some sequence $\hat{S} = (u_i)_{i=1}^{|V|}$. For $r = 1, \ldots, |V|$, let G_r be the subgraph of G that exists just prior to the deletion of u_r by Algorithm 2.6. Select an edge $e = \{u_j, u_l\}$ such that (without loss of generality) u_j precedes u_l in $\hat{S}$. Because Algorithm 2.6 can delete u_j from G_j, we know $C_{G_j}(u_j) \geq \deg_{G_j}(u_j) \geq 1$. Suppose that we delete e from G to create the graph G' (so that $\deg_{G'}(u_j) = \deg_G(u_j) - 1$), and simultaneously create configuration C' on G' by removing one chip from u_j (so that $C'_{G'}(u_j) = C_G(u_j) - 1$) and one chip from u_l, if possible (we discuss this case later in the paragraph). We claim that Algorithm 2.6 may delete the nodes in $V(G')$ in the same sequence $\hat{S}$. The only nodes affected by the deletion of e are u_j and u_l, so we consider each in turn.

Define G_r' analogously to G_r. If we delete the nodes $u_1, \ldots, u_{j-1}$ of G' in that order, we have $\deg_{G_j'}(u_j) = \deg_{G_j}(u_j) - 1$. To thereupon delete u_j we require $C'_{G_j'}(u_j) \geq \deg_{G_j'}(u_j)$, which follows from

$$\begin{aligned} C'_{G_j'}(u_j) &= C'_{G'}(u_j) \\ &= C_G(u_j) - 1 \\ &= C_{G_j}(u_j) - 1 \\ &\geq \deg_{G_j}(u_j) - 1 \\ &= \deg_{G_j'}(u_j). \end{aligned}$$

We now turn to u_l. Because Algorithm 2.6 can delete u_l from G_l, then $C_{G_l}(u_l) \geq \deg_{G_l}(u_l)$. If $C_{G_l}(u_l) = 0$, then all of the G-neighbors of u_l must precede u_l in $\hat{S}$. In this case, $\deg_{G_l'}(u_l) = 0$, so $C'_{G_l'}(u_l) \geq \deg_{G_l'}(u_l)$ even if we were not able to remove a chip from u_l as described in the paragraph above. If $C_{G_l}(u_l) > 0$, then we were able to remove a chip from u_l as described above. Here the same argument given for u_j applies to u_l. Thus, Algorithm 2.6 may delete the nodes in $V(G')$ in the order given in $\hat{S}$; this assures us that the new configuration on G' is legal.

Let S be the subsequence of $\hat{S}$ that contains the nodes in Y. With the help of the ideas in the paragraphs above, we now establish that Algorithm 2.6, acting on G, can also delete the nodes in $\hat{S} \backslash S$ before it deletes the nodes in S. Create the subgraph $\hat{G}$ of G by deleting every edge in $E(G)$ that is incident with a node in $V(Y)$. At each deletion, remove one chip from the nodes incident with these edges, if possible. Let $\hat{C}$ be the configuration that results on $\hat{G}$. We know from (4.10) that

$C_G(w) \geq |\Gamma_Y(w)|$ for $w \notin V(Y)$, so $\hat{C}_{\hat{G}}(w) \geq 0$; that is, we will be able to remove a chip from w for each edge incident with w that we delete. By the argument given above, we know $\hat{C}$ is legal. Because we have deleted every edge in $E(G)$ that is incident with a node in $V(Y)$, we know that $\deg_{\hat{G}}(y) = 0$ for all $y \in Y$. Therefore, if we use Algorithm 2.6 to investigate the legality of $\hat{C}$, we may delete the nodes in $V(Y)$ at any time we choose; in particular, we choose to delete them after we delete the nodes in $V(G - Y)$. Let $\hat{Q}$ be the sequence in which we delete the nodes in $\hat{G}$; specifically, let $\hat{Q} = (w_1, \ldots, w_\alpha, w'_1, \ldots, w'_\beta)$, where $\alpha + \beta = |V|$, $w_a \in V(G - Y)$, and $w'_b \in Y$. Define $\hat{G}_r$ analogously to G_r. For $w_a \in V(G - Y)$, we know $C_{\hat{G}_a}(w_a) \geq \deg_{\hat{G}_a}(w_a)$.

We claim that the nodes in $V(G - Y)$ may be deleted from G by Algorithm 2.6 in the same order they appear in $\hat{Q}$. Restore the edges that were deleted during the creation of $\hat{G}$, along with the chips that were removed from the nodes. After we restore both the deleted edges and the removed chips, we have

$$C_{G_a}(w_a) = C_{\hat{G}_a}(w_a) + |\Gamma_Y(w_a)| \geq \deg_{\hat{G}_a}(w_a) + |\Gamma_Y(w_a)| = \deg_{G_a}(w_a).$$

Thus, we may let w_a be the a^{th} node deleted as Algorithm 2.6 investigates the legality of G. Let W denote this sequence $(w_i)_{i=1}^{\alpha}$. Since $C|_Y$ is legal by Lemma 2.8, we may use Algorithm 2.6 to delete the nodes in Y in some order. Let W' be this sequence, which we note is a permutation of S.

So, if we concatenate the sequences W and W' to form the sequence T, we know that Algorithm 2.6, acting on G, can delete the nodes in V in the sequence T.

Now concatenate the sequences W' and W to form the sequence T^*. We argue now that when we check C^* for legality using Algorithm 2.6, we can delete the nodes in the order found in T^*.

First, we show that v must be the initial entry in W'. If v is the only entry in W', this is clear; further, since v is critical in C^*, Algorithm 2.6 may delete v as it works on C^*. In this case, we may pick up the argument as it resumes after (4.14). If v is not the only entry in W', then suppose $y \in Y$, with $y \neq v$, is the first entry in W'. By definition of Y, we have $C(y) < \deg_Y(y)$; thus, at least one neighbor of y in Y must be deleted before Algorithm 2.6 may delete y. This immediately gives our contradiction. Suppose then that $W' = (v, y_1, y_2, \ldots, y_k)$. For $i = 1, 2, \ldots, k$, let t_i denote the number of neighbors of y_i that precede y_i in W'. For each i we have

$$C(y_i) < \deg_Y(y_i). \tag{4.11}$$

But since y_i can be deleted (from Y) in the order specified by W', we have

$$C(y_i) \geq \deg_Y(y_i) - t_i. \tag{4.12}$$

In (4.9) we set

$$C^*(y_i) = (C(y_i) - \deg_Y(y_i))(\text{mod } \deg_G(y_i) + 1).$$

But (4.11) implies that

$$C^*(y_i) = \deg_G(y_i) + 1 + C(y_i) - \deg_Y(y_i). \tag{4.13}$$

Now (4.12) and (4.13) together imply that

$$\begin{aligned} C^*(y_i) &\geq \deg_G(y_i) + 1 + \deg_Y(y_i) - t_i - \deg_Y(y_i) \\ &> \deg_G(y_i) - t_i. \end{aligned} \tag{4.14}$$

By (4.14) we conclude that in C^* the nodes in W' can be deleted in the order given by this sequence. It remains to show that the nodes in W may be deleted sequentially following the deletion of every node in W'. By (4.9), the only nodes in W for which $C(w_i) \neq C^*(w_i)$ are those with neighbors in Y. Specifically, for $i = 1, 2, \ldots, \alpha$, we have $C^*(w_i) = C(w_i) - |\Gamma_Y(w_i)|$ by (4.9) and (4.10). However, we have just seen that *all* of the nodes in $V(Y)$ may be deleted first as Algorithm 2.6 checks the legality of C^*, so the $|\Gamma_Y(w_i)|$ neighbors of w_i in Y have been deleted when the algorithm begins to delete the nodes in W. Let t_i' equal the number of neighbors of w_i that precede w_i in W. Because C is a legal configuration on G, we know that $C(w_i) \geq \deg_G(w_i) - t_i'$. Thus,

$$\begin{aligned} C^*(w_i) &= C(w_i) - |\Gamma_Y(w_i)| \\ &\geq \deg_G(w_i) - t_i' - |\Gamma_Y(w_i)|, \end{aligned}$$

which implies that Algorithm 2.6 may delete w_i in the order it appears in W. We have now established that we may delete all of the nodes in the configuration C^* in the manner detailed in Algorithm 2.6; thus, by Proposition 2.9 we know that C^* is a legal configuration. ■

(b) *If v is seeded in C^*, then the resulting relaxed configuration is C.*

In our argument for part (a), we saw that an application of Algorithm 2.6 may delete the nodes of G when they are in the configuration C^* by deleting first the nodes in W' and then those in W. We now argue that when C^* is seeded at v, precisely the nodes in W' fire during the game.

It is clear that v will fire when it receives one chip because $C^*(v) = \deg_G(v)$. For each y_i in $W' = (v, y_1, y_2, \dots, y_k)$, we have $C^*(y_i) > \deg_G(y_i) - t_i$ by (4.14). If the nodes in $(v, y_1, y_2, \dots, y_{i-1})$ fire, then the number of chips on y_i increases to $C^*(y_i) + t_i > \deg_G(y_i)$. Thus y_i may now fire as well; so, all nodes in W' fire during the game. When we create C^* from C we remove $|\Gamma_Y(w)|$ chips from each node $w \notin V(Y)$ (see (4.9) and (4.10)). But if we fire all of the nodes in $V(Y)$, then w gains exactly that many chips. Since w is not supercritical in C, it is also not supercritical at any point in the game that starts on C^*. Thus none of the nodes in W fire during the game. Further, it is evident that when C^* has relaxed, the number of chips on w is $C(w)$.

We now establish that when C^* has relaxed, the number of chips on each $y_i \in V(Y) - v$ is $C(y_i)$. Since y_i fires during the game, it loses $\deg_G(y_i) + 1$ chips. But all $\deg_Y(y_i)$ of its neighbors also fire during the game, so it gains this many chips. Thus, from (4.13) we see that when C^* has relaxed, the number of chips on each $y_i \in V(Y) - v$ is $C(y_i)$.

Finally, we recall that $C^*(v) = \deg_G(v)$ by definition. When v is seeded and it fires, it loses all of its chips, but then it gains one chip from each of its $\deg_Y(v)$ neighbors in Y. When the game finishes, v will thus contain $\deg_Y(v)$ chips, which, as we saw when we defined Y, is $C(v)$. We have therefore established that C^* relaxes to C when v is the seed. ■

We intend to prove that all relaxed legal configurations are equally likely. As a step in this direction, it is helpful for us to show that choosing two seeds in a given configuration results in two distinct relaxed configurations; before we do, we pause to prove the following useful result.

Lemma 4.12 *Let C be a relaxed legal configuration on a connected graph G. Consider a burn-off game of length at least two that results in the configuration C'. Let H be the subgraph induced on the nodes that fire during the game. After relaxation, only the seed v will be critical in H.*

Proof. The seed v must fire, since the game length is at least two. When v fires it loses all its chips. Each of its neighbors in H fires, sending a chip back to v. By Lemma 4.8, none of these nodes may fire a second time. Thus, at the end of the game, $C'(v) = \deg_H(v)$; that is, v is critical in H.

Consider $u \in V(H) - v$. We know that $C(u) \leq \deg_G(u)$ as the game begins. By Lemma 4.8, each node in H fires exactly once. Thus, each of the $\deg_H(u)$ neighbors of u in H fires and adds one chip to u. By Proposition 2.1, the firing order has no effect on C'. Thus, $C'(u) = C(u) + \deg_H(u) - (\deg_G(u) + 1) < \deg_H(u)$. ■

Now we proceed to the main result that, along with Proposition 4.11, allows us to conclude that all relaxed legal configurations in a burn-off game on a given connected graph are equally likely.

Proposition 4.13 *Given $C \in \mathcal{R}$ on a connected graph $G = (V,E)$, it is not possible to choose two distinct seeds and relax the resulting configurations to the same legal configuration.*

Proof. Define $R_v : V \to \mathbb{N}$ as the relaxed configuration that results from seeding C at v. We wish to show that for $u, v \in V$, with $u \neq v$, we have $R_u \neq R_v$. Suppose, for a contradiction, that $R_u = R_v$. If u is the seed and does not fire, then $R_u(u) = C(u) + 1$. If v is the seed and does not fire, then $R_v(u) = C(u)$. Thus, $R_u \neq R_v$, which contradicts our assumption. Thus, at least one of u, v must fire when chosen as the seed.

Suppose first (without loss of generality) that if G is seeded at u, then u fires, while if G is seeded at v, then v does not fire. Since v is not critical in G and we have assumed $R_u = R_v$, then $C(u) = R_v(u) = R_u(u)$. If u is chosen as the seed, it fires and loses all of its chips. Because $R_u(u) = R_v(u)$, u must regain $\deg(u)$ chips. This happens only if every neighbor of u fires after u itself fires. Now for every neighbor of u to fire, at least one of those neighbors must itself be critical in G. Suppose that w is such a neighbor. In the game where v is the seed and does not fire, we thus know that w is critical in G. Because $R_u(w) = R_v(w)$, it must be true that w again becomes critical, after it fires, in the game where u is the seed. But this is impossible by Lemma 4.12.

Now we may suppose that both u and v fire if chosen as the seed. First, we show that if v is the seed (without loss of generality), then u must also fire. Suppose by way of contradiction that u does not fire, so $R_v(u) = \deg(u)$. If u and v are neighbors, then clearly u must fire if v is the seed; we therefore assume for the remainder of the proof that u and v are not neighbors. Then in the game where u is the seed, every neighbor of u must also fire so that $R_u(u) = \deg(u)$. Therefore, at least one neighbor w of u must be critical in G. By Lemma 4.12, $R_u(w) < C(w)$. But $R_u = R_v$ implies that w must also fire in the game where v is the seed. Since u is critical in G, it too will fire when w fires. This contradicts our assumption. Thus, if either u or v is the seed, the other node must fire during the game.

So in the game where v is the seed, at least one neighbor of v must also fire so that u will fire. Suppose that k neighbors of v fire. After v fires, it contains zero chips, so $R_v(v) = k$. Because $R_u = R_v$, we must have $R_u(v) = k$ as well. In the game where u is the seed, at least one neighbor of v must fire in order to allow v to fire. Say that w is such a neighbor. Now after v fires, it contains zero chips, so k of its neighbors must now fire to achieve $R_u(v) = k$. Since w has already fired in this game, it may not be one of these k neighbors (by Lemma 4.8). So at least $k + 1$ neighbors of v fire in the game that starts with u as the seed.

Thus, we have k neighbors of v firing in the game where v is the seed and at least $k + 1$ neighbors of v firing in the game where u is the seed. There is therefore a node z that is adjacent to v and that fires when u is the seed but not when v is the seed. The number of chips on z at the end of a game in which it does not fire must be at least $C(z)$. But the number of chips on z at the end of a game in which it does fire must be at most $C(z) + \deg(z) - (\deg(z) + 1) < C(z)$. Thus, $R_u(z) < R_v(z)$, contradicting our assumption that $R_u = R_v$. ■

We may now proceed with the main result of this section. Recall that we are considering a sequence of burn-off games as a Markov chain where the states are the relaxed legal configurations and the transitions are determined by the configurations that result when a node is chosen as the seed and the configuration is relaxed. Let D be the digraph that represents this Markov chain, and let P be the transition probability matrix. Let $\mathcal{R}$ be the set of relaxed legal configurations on G and let $r = |\mathcal{R}|$.

Theorem 4.14 *All relaxed legal configurations in a sequence of burn-off games on a connected graph $G = (V,E)$ are equally likely.*

Proof. Let $C^* \in \mathcal{R}$ be a relaxed legal configuration on G. When we initiate a burn-off game on G by seeding one of its nodes, C^* relaxes to a legal configuration (by Proposition 2.1). As a result, we know that

$$\mathrm{outdeg}_D(C^*) \le |V| \quad \text{for all } C^* \in \mathcal{R}. \tag{4.15}$$

Summing over all relaxed legal configurations, we have

$$\sum_{C^* \in \mathcal{R}} \mathrm{outdeg}_D(C^*) \le r|V|. \tag{4.16}$$

In Proposition 4.11 we saw that for any legal configuration C and node v there is at least one legal configuration C^* that relaxes to C when v is seeded. Then in Proposition 4.13 we found that such C^* must be distinct for each choice of v. Together, these results imply that

$$\mathrm{indeg}_D(C) \ge |V| \quad \text{for all } C \in \mathcal{R}. \tag{4.17}$$

Summing over all relaxed legal configurations, we have

$$\sum_{C \in \mathcal{R}} \mathrm{indeg}_D(C) \ge r|V|. \tag{4.18}$$

Inequalities (4.16) and (4.18) together give

$$r|V| \geq \sum_{C \in \mathcal{R}} \text{outdeg}_D(C) = \sum_{C \in \mathcal{R}} \text{indeg}_D(C) \geq r|V|,$$

which establishes the identity

$$\sum_{C \in \mathcal{R}} \text{outdeg}_D(C) = \sum_{C \in \mathcal{R}} \text{indeg}_D(C) = r|V|. \tag{4.19}$$

Taking (4.19) along with (4.15) and (4.17) respectively, we conclude that

$$\text{outdeg}_D(C) = \text{indeg}_D(C) = |V| \quad \text{for all } C \in \mathcal{R}. \tag{4.20}$$

In a burn-off game, the seed is chosen at random, so every nonzero entry in P is $1/|V|$. Since $\text{indeg}_D(C) = |V|$ for all $C \in \mathcal{R}$, it follows that P is doubly stochastic. Therefore (see, e.g., [10]), the entries of the stationary distribution are all equal. ■

Theorem 4.9 and Theorem 4.14 together give us a method by which we may mathematically determine, without resorting to computer simulations, the probabilities associated with the lengths of burn-off games on any given connected graph. They do not, however, provide a closed formula for finding these probabilities. In Chapter 5, we return to the special case of complete graphs and, in this more restrictive setting, find such a closed formula. We conclude the present chapter with an example to show how our mathematical results can be compared to our earlier empirical ones.

Example 4.15 Consider the graph G shown in Figure 1.1 on p. 4. Applying Theorem 4.1 to G, we find that there are 40 relaxed legal configurations. Because G has four nodes, there are 160 pairs (C,v) of relaxed legal configurations and seed choices. Of these, 82 pairs result in a game of length zero (Proposition 4.7). We know that burn-off games on G may not have length greater than four (Lemma 4.8).

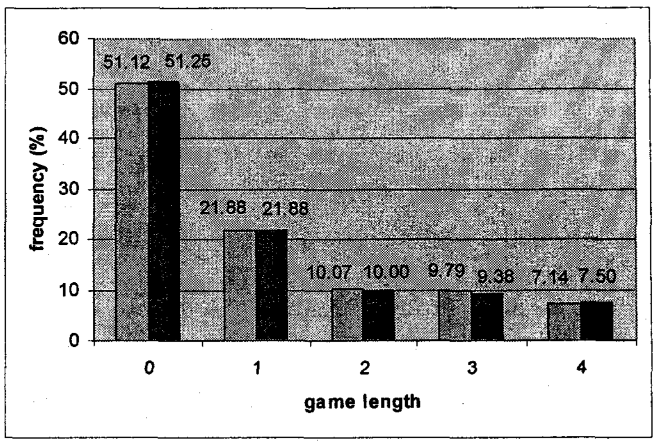

Figure 4.6. Comparison of analytic results to previously simulated results in Figure 1.2, p. 9

Four applications of Theorem 4.9 show that the number of pairs resulting in games of length one, two, three, and four are 35, 16, 15, and 12 respectively. By Theorem 4.14, each relaxed legal configuration is equally likely to appear during a sequence of burn-off games on G. Thus, we may calculate the probability distribution of the game length. These results are shown in Figure 4.6: the lefthand bars record the results of 10,000 computer simulations, previously appearing in

Figure 1.2; the righthand bars are those found in this example using the methods of the present chapter. The close visual agreement between the analytical and simulated data was confirmed by a χ^2 goodness-of-fit test. Even with the level of significance α as high as 0.1, this test did not reject the hypothesis that our analytical results correctly model the simulated data.

Chapter 5

Distribution of game length on a complete graph

Our main goal in this chapter is to find a closed formula for the probability distribution of the burn-off game length on a complete graph. By finding this distribution, we shall draw conclusions about whether such games exhibit SOC behavior.

5.1 Overview

Refer to the introduction of Chapter 4 for a description of the probability model we associate with burn-off games. Let us define the random variable X to be the length $g \geq 0$ of a burn-off game on K_n, and, as in Chapter 3, let $\mathcal{R}$ be the set of relaxed legal configurations on K_n. To determine the distribution of X, we condition on the configuration $q \in \mathcal{R}$ that is seeded:

$$\Pr\{X = g\} = \sum_q \Pr\{X = g \mid q \text{ is seeded}\} \Pr\{\text{seeded configuration is } q\}. \qquad (5.1)$$

In Section 5.2, we find that Pr{seeded configuration is q} does not depend on q, but only on n. In Section 5.3, we find an expression for

$$P_{g,q} := \Pr\{X = g \mid q \text{ is seeded}\}$$

in terms of just n and g. These results yield a formula for finding the probability of any game length we choose.

5.2 All relaxed legal configurations on K_n are equally likely

Recall from the introduction to Chapter 4 that we view burn-off games as a sequence of experiments in which a pair $(q, v) \in \mathcal{R} \times V$ results in an outcome $q^* \in \mathcal{R}$. In this section, we investigate the long-run probability that, during the relaxed phase of a burn-off chip-firing game on K_n, the chips are distributed on the nodes in a particular configuration q. We know from Theorem 4.14 that this long-run probability does not depend on q; we include this alternate proof because it results in an explicit formula in the case of complete graphs.

Theorem 5.1 *In a sequence of burn-off games played on K_n, the stationary probability of a burn-off game relaxing to configuration $q \in \mathcal{R}$ is $\frac{1}{(n+1)^{n-1}}$.*

Recalling Theorem 3.3, we see that the denominator here is the number of relaxed legal configurations for a burn-off chip-firing game on K_n. Theorem 5.1 simply states that every relaxed legal configuration is equally likely to appear during the relaxed phase of a sequence of burn-off games. Since the proof occupies a number of pages, we first provide an outline.

Let q be a legal configuration on K_n, and $v \in V(K_n)$. We will find a legal configuration q^* such that, if v is the seed, the resulting relaxed configuration is q (cf. Proposition 4.11, where our proof is less transparent). We will also show that if the chips are in a configuration q^*, it is impossible to choose two *different* nodes as the seed and result in the *same* configuration q. Thus, we will show that there are at least n configurations on which a particular choice of seed will result in the configuration q.

Thanks to the deterministic nature of chip-firing games (see Proposition 2.1), it is easy to see that for any given configuration q^*, there are at most n configurations that can result from adding a chip to a node. Every chip-firing game must relax to just one configuration, and since there are n nodes to choose as the seed, there are at most n possible outcomes.

These bounds together imply that for each configuration q there are exactly n configurations q^* that relax to q. Since each node is equally likely to be chosen as a seed, we will be able to use a key property of Markov chains with doubly-stochastic transition matrices to complete our argument.

Proof of Theorem 5.1. Let q be a legal configuration on K_n and $v \in K_n$. We seek a legal configuration q^* such that, if q^* is seeded at v, the resulting relaxed configuration is q. We consider two cases. Before jumping in, let us note that if $q(v) = 0$, then Proposition 3.1 implies that we cannot have $q(u) = 0$. Thus, any configuration with $q(v) = 0$ will be addressed in Case 2.

Case 1. There exists $u \neq v$ such that $q(u) = q(v) \geq 1$.

Suppose that $q(u) = q(v) = j$. In this case, it suffices to take q^* as the configuration agreeing with q except for one fewer chip on v. Since q is legal, Proposition 3.1 shows that at most $j+1$ nodes contain j or fewer chips. But $q(u) = q(v) = j$ implies that at most $j-1$ nodes contain $j-1$ or fewer chips. When a chip is removed from v to obtain q^*, at most j nodes will contain $j-1$ or fewer chips. Thus, q^* is a legal configuration; clearly, q^* relaxes to q (after zero steps).

Case 2. No other node contains the same number of chips as does v.

Denote the number of chips on the nodes of K_n with a vector as follows:

$$q = (x_{n-1}, x_{n-2}, \ldots, x_{i+1}, x_i, x_{i-1}, \ldots, x_2, x_1, x_0),$$

where $x_{n-1} \geq x_{n-2} \geq \cdots \geq x_1 \geq x_0$ (we shall refer to this as the *canonical order* on q), and x_i is the number of chips on v. Later, it will be useful to rewrite these inequalities as: $x_{i-k} \geq x_{i-(k+1)}$ for $0 \leq k < i$, and $x_{n-j} \geq x_{n-(j+1)}$ for $1 \leq j < n - i$.

Since q is legal, at most i nodes contain $i - 1$ or fewer chips. In addition, q is written in canonical order, so $x_i \geq i$. If $x_i > i$, then we may reduce the number of chips on v by one to create q^*, a configuration in which at most i nodes contain $i - 1$ or fewer chips. Therefore, it remains only to consider those cases where $x_i = i$.

Since q is legal, we know that $x_{i-k} \geq i - k$, for $0 \leq k < i$, and that $x_{n-j} \geq n - j$, for $1 \leq j < n - i$. We therefore know that $x_{i-1} = i - 1$, because $i = x_i \geq x_{i-1} \geq i - 1$, and v is the only node containing x_i chips. Since all legal configurations have at least one critical node, and q is written in canonical order, we also have $x_{n-1} = n - 1$, the common degree in K_n. For later reference, we restate and label these findings.

$$x_i = i. \tag{5.2}$$
$$x_{i-1} = i - 1. \tag{5.3}$$
$$x_{i-k} \geq i - k, \text{ for } 0 \leq k < i. \tag{5.4}$$
$$x_{n-1} = n - 1. \tag{5.5}$$
$$x_{n-j} \geq n - j, \text{ for } 1 \leq j < n - i. \tag{5.6}$$
$$x_{i-k} \geq x_{i-(k+1)} \text{ for } 0 \leq k < i. \tag{5.7}$$
$$x_{n-j} \geq x_{n-(j+1)} \text{ for } 1 \leq j < n - i. \tag{5.8}$$

The configuration q^* that we claim relaxes to q when seeded at v is

$$q^* = (x_i^*, x_{i-1}^*, \ldots, x_2^*, x_1^*, x_0^*, x_{n-1}^*, x_{n-2}^*, \ldots, x_{n-(n-(i+1))}^*),$$

where

$$\begin{aligned}
x_i^* &= n-1, \\
x_{i-k}^* &= x_{i-k} - x_i + n \quad \text{for } 1 \leq k \leq i, \qquad (5.9)\\
x_{n-j}^* &= x_{n-j} - (x_i + 1) \quad \text{for } 1 \leq j \leq n-(i+1).
\end{aligned}$$

Example 5.2 Consider K_6, and $q = (5,4,4,2,1,0)$. Suppose we pick the node containing 2 chips as our special node v. Note that $x_i = 2 = i$, and that v is the only node containing this many chips—these two qualities fit the requirements of Case 2.

According to the definition of q^*, we have $q^* = (5,5,4,2,1,1)$. Since $x_i^* = 5$, if we add a chip to v, then we obtain the following firing sequence:

$$\begin{gathered}
(6,5,4,2,1,1)\\
(0,6,5,3,2,2)\\
(1,0,6,4,3,3)\\
(2,1,0,5,4,4).
\end{gathered}$$

If re-written in canonical order, the final configuration is q.

Following our outline, to complete the proof of Theorem 5.1, we will first establish four facts about q^*:

(a) q^* is in canonical order;

(b) q^* is a legal configuration;

(c) if v is seeded in q^*, then q^* will relax to q;

(d) starting from q^*, it is not possible to seed a node other than v and relax to q.

(a) *q^* is in canonical order.*

The configuration q^* is in canonical order if its entries are nonincreasing. To demonstrate this, we observe that the entries $x_i^*, x_{i-1}^*, \ldots, x_2^*, x_1^*, x_0^*$ are nonincreasing. First note that

$$\begin{aligned} x_{i-1}^* &= x_{i-1} - x_i + n \\ &= (i-1) - i + n \\ &= n-1, \end{aligned}$$

so that $x_i^* = n - 1 = x_{i-1}^*$. Now for $0 < k < i$,

$$\begin{aligned} x_{i-k}^* &= x_{i-k} - x_i + n \\ &\geq x_{i-(k+1)} - x_i + n \\ &= x_{i-(k+1)}^*, \end{aligned}$$

where the inequality holds by (5.7). Observe that $x_0^* \geq x_{n-1}^*$ because

$$\begin{aligned} x_0^* &= x_0 - x_i + n \\ &\geq n - x_i \\ &= n - i \\ &> n - i - 2 \\ &= (n-1) - (i+1) \\ &= x_{n-1} - (x_i + 1) \\ &= x_{n-1}^*. \end{aligned}$$

Finally, we show that the entries $x_{n-1}^*, x_{n-2}^*, \ldots, x_{n-(n-(i+1))}^*$ are nonincreasing. For $1 \leq j \leq n-(i+1)$,

$$\begin{aligned} x_{n-j}^* &= x_{n-j} - (x_i + 1) \\ &\geq x_{n-(j+1)} - (x_i + 1) \\ &= x_{n-(j+1)}^*, \end{aligned}$$

where the inequality holds by (5.8). Thus, q^* is in canonical order. ■

(b) *q^* is a legal configuration.*

If a configuration $r = (r_{n-1}, r_{n-2}, \ldots, r_j, r_{j-1}, r_{j-2}, \ldots, r_2, r_1, r_0)$ is in canonical order, and for all $0 \leq j \leq n-1$, only the entries $r_{j-1}, r_{j-2}, \ldots, r_2, r_1, r_0$ are $j-1$ or less, then at most j entries of r are $j-1$ or less, so r is legal (by Proposition 3.1). In other words, if r is in canonical order and $r_j \geq j$ for all $0 \leq j \leq n-1$, then r is legal. We have established in part (a) that q^* is in canonical order, so here we show that $q_j^* \geq j$ for all such j.

We begin by noting that $x_i^* = n-1$, so that $q_{n-1}^* \geq n-1$. We turn our attention to the terms that follow x_i^*. First, for $0 < k \leq i$, note that $q_{n-(k+1)}^* = x_{i-k}^*$ and that

$$\begin{aligned} x_{i-k}^* &= x_{i-k} - x_i + n \\ &\geq i - k - i + n \\ &= n - k \\ &> n - (k+1), \end{aligned}$$

where the first inequality holds by (5.2) and (5.4). Second, for $1 \leq j \leq n - (i+1)$, note that $q_{j-1}^* = x_{n-(n-(i+j))}^*$ and that

$$\begin{aligned} x_{n-(n-(i+j))}^* &= x_{n-(n-(i+j))} - (x_i + 1) \\ &= x_{i+j} - x_i - 1 \\ &\geq i + j - i - 1 \\ &= j - 1, \end{aligned}$$

where the inequality holds by (5.2) and (5.6). Since q^* is in canonical order and each of its entries is sufficiently large, q^* is legal. ■

(c) *If q^* is seeded at v, then the resulting relaxed configuration is q.*

Recall that $q_{n-1}^* = x_i^*$, the number of chips on v in the configuration q^*. Since $x_i^* = n - 1$ (by (5.9)), then v will fire when q^* is seeded at v. We proceed to show that this causes all of the nodes represented in the next i entries of q^* also to fire but none of the remaining $n - (i+1)$ nodes.

For convenience, we will refer to the nodes of K_n by their corresponding positions in q^*, so, e.g., the node starting with x_i^* chips will be called node $(n-1)$

and the node starting with $x^*_{n-(n-(i+1))}$ chips will be called node 0. Note that for $k = 0, 1, \ldots, i$, there are x^*_{i-k} chips on the node $n - (k + 1)$.

We demonstrate now that the node $n - k$, for $2 \leq k \leq i + 1$, will fire if all nodes $n - a$, for $a = 1, 2, \ldots, k - 1$, have fired. The node $n - k$ begins with $x^*_{i-(k-1)}$ chips, and, using the estimates from part (b) above, we have $x^*_{i-(k-1)} > n - k$. Thus, after the nodes $n - a$, for $a = 1, \ldots, k - 1$, have fired and thus added $k - 1$ chips to the node $n - k$, the number of chips on this node will exceed $(n - k) + (k - 1) = n - 1$. Therefore, the node $n - k$ indeed fires.

We show now that none of the nodes $n - a$, for $a = i + 2, i + 3, \ldots, n$, will fire. We need only check the status of the node $n - (i + 2)$, because q^* is in canonical order, implying that the node $n - (i + 2)$ contains at least as many chips as any of the nodes $n - a$ for which $a = i + 3, \ldots, n$. The number of chips on the node $n - (i + 2)$ is $x^*_{n-1} = x_{n-1} - (x_i + 1) = (n - 1) - (i + 1) = n - i - 2$. By the argument above, all nodes earlier than the node $n - (i + 2)$ in the canonical ordering have fired. There are $i + 1$ such nodes, so $i + 1$ chips are added to the node containing $x^*_{n-1} = n - i - 2$ chips in q^*. The new number of chips on the node $n - (i + 2)$, then, is $(n - i - 2) + (i + 1) = n - 1$, so that node does not fire.

Finally, we argue that q^* relaxes to q. First, consider the nodes that do fire, namely, the nodes containing x^*_{i-k} chips (where $k = 0, 1, \ldots, i$). Recall from (5.9) that $x^*_{i-k} = x_{i-k} - x_i + n$, and note that $i + 1$ nodes fire during the course of the game, including the node in question. When a node fires, it loses n chips, so the new number of chips on a node that begins with x^*_{i-k} chips is $x^*_{i-k} + i - n = x_{i-k} - x_i + n + i - n = x_{i-k}$, since $x_i = i$ (by (5.2)).

Now consider the nodes that do not fire, namely, the nodes with x^*_{n-j} chips on them (where $j = 1, 2, \dots, n-(i+1)$). Since $x^*_{n-j} = x_{n-j} - (x_i + 1)$, and $i+1$ nodes fire (adding one chip each to every other node), the new number of chips on a node that begins with x^*_{n-j} chips is $x^*_{n-j} + (i+1) = x_{n-j} - (x_i + 1) + (i+1) = x_{n-j}$, again using (5.2). Thus, we see that for both sets of nodes, the corresponding entries of q^* are transformed to those of q by the firing sequence. ■

(d) *Starting from q^*, it is not possible to seed at a node $u \neq v$ and relax to q.*

We show that if we start with the configuration q^* and seed a firing sequence at a node $u \neq v$, the resulting configuration cannot be identical to q.

One of two outcomes can occur: (1) neither u nor v fires if a chip is added, or (2) one or both fire if a chip is added. In the first case, the conclusion clearly follows: adding a chip to u increases the number of chips on u by one, but does not change the number of chips on v, while adding a chip to v has the opposite effect.

As we turn to the second case, we recall Lemma 4.8 which implies that a burn-off game started in a relaxed configuration may take at most n turns. (To establish that such a game may take exactly n turns, consider K_n with every node containing $n - 1$ chips.) Note also that if a node x fires once during the course of a game of length g, then the number of chips on x increases by $g - 1$ (or g, if x is the seed) and decreases by n (one chip moves to each neighbor, and one burns off). On the other hand, if x does not fire during the course of such a game, then the number of chips on x increases by g. Thus, if u is the seed, and it fires once, the number of chips on u increases by g and decreases by n.

Now we consider case (2): let u be the node that fires if a chip is added to it and v be the node that may or may not fire. We examine these two possibilities in turn. Suppose that v contains fewer than $n - 1$ chips, so that if it were chosen as the seed, it would not fire—in this case, the number of chips on u would remain constant. If u is the seed, it will fire and lose all of its chips. For u to regain its $n - 1$ chips, g would have to be equal n (by Lemma 4.8, g cannot exceed n). But if $g = n$, then the number of chips on v increases by $g - 1 = n - 1$ and decreases by n, resulting in the loss of one chip from v.

Suppose instead that v contains $n - 1$ chips, so that if it were chosen as the seed, it would fire, and the number of chips on it would increase by g and decrease by n. In this case, were u to be the seed, v would fire as well, which would increase the number of chips on v by $g - 1$ and decrease it by n. In these cases, the resulting numbers of chips on v differ, so the final configurations cannot be the same. ■

Combining the results (a) through (d) with the remarks in the outline of this proof (p. 100), we have now established that for every relaxed configuration q, there are exactly n relaxed configurations that, when seeded, will relax to q.

Consider the burn-off chip-firing game as a Markov chain, where the states are the relaxed legal configurations, and the transitions are determined by the configurations that result when a node is seeded. Let D be the digraph representing this Markov chain. If $\mathcal{R}$ is the set of relaxed legal configurations, then

$$n\left|\mathcal{R}\right| \le \sum_{C\in\mathcal{R}} \mathrm{indeg}_D(C) = \sum_{C\in\mathcal{R}} \mathrm{outdeg}_D(C) \le n\left|\mathcal{R}\right|. \tag{5.10}$$

The first inequality follows from the conclusions just reached, and the second is justified in the outline of the proof (p. 100). In a burn-off game, the seed is chosen at random, so the transition probabilities are all $1/n$. Since by (5.10) we have $\mathrm{indeg}_D(C) = \mathrm{outdeg}_D(C) = n$ for all $C \in \mathcal{R}$, the transition matrix for this Markov chain is doubly stochastic. It follows (see, e.g., [10]) that the entries of the stationary distribution are all equal. In other words, we have shown that all relaxed legal configurations on K_n are equally likely. We saw earlier that the number of legal configurations on K_n is $(n+1)^{n-1}$. Therefore, the stationary probability of a burn-off game relaxing to configuration q is $\frac{1}{(n+1)^{n-1}}$. ■

5.3 The probabilities associated with varying game lengths

In this section, we consider the conditional probability that, during a burn-off chip-firing game on K_n, a firing sequence will be of a particular length $g \geq 0$, given that the game is in a particular legal configuration q. In other words, we investigate

$$P_{g,q} = \Pr\{X = g \mid q \text{ is seeded}\},$$

for a nonnegative integer g and $q \in \mathcal{R}$. Recall this is the conditional probability appearing in Section 5.1.

Recall that a node in a relaxed configuration is *critical* if it contains as many chips as its degree, that is, if the node will fire with the addition of a single chip. Call a node *unstable* if it will fire only when a critical node is chosen as the seed. Finally, call a node a *dud* if it cannot fire during *any* game that is played on its configuration. Either a node will fire when seeded (so is critical) or will not (so is unstable or a dud). But an unstable node will fire when a critical node is seeded, whereas a dud will not. Since each node must be one of these three types, the nodes are partitioned into critical nodes, unstable nodes, and duds.

Example 5.3 Consider the configuration $q = (6,6,5,4,1,1,0)$ on K_7. This configuration has two critical nodes, each containing 6 chips. Either of these nodes will fire if chosen as the seed. The configuration has two unstable nodes that contain

5 and 4 chips. If either of these is seeded, it will not fire; however, if one of the critical nodes is seeded, both of the critical nodes will fire, and both will contribute one chip to the node containing 5 chips, allowing that node to fire. Now, with the addition of the three chips provided by these three firings, the node containing 4 chips will fire as well. Since none of the remaining three nodes will collect enough chips to fire, these nodes are duds.

Given a configuration q, let c denote the number of critical nodes and u the number of unstable nodes. Notice that there are only two possible game lengths that can result when a random node is seeded: either the node will not fire, resulting in a game of length zero; or the node will fire, resulting in a game of length $c + u$, by Lemma 4.8. For this reason, we will profit from counting the configurations that have c critical nodes and u unstable nodes, for fixed nonnegative integers c and u.

We handle the case $c = 1$ separately after considering the case $c \geq 2$.

Proposition 5.4 *For all integers $2 \leq c \leq n$, $0 \leq u \leq n - c$, and $d = n - (c + u)$, the number of configurations on K_n with c critical nodes, u unstable nodes, and d duds is*

$$\binom{n}{d} L_{d,d-1} \binom{n-d}{u} L_{u,n-d-3}. \tag{5.11}$$

Proof. Let Q be the set of configurations that satisfy the conditions of our proposition. We shall work toward determining the size of Q by first examining all possible chip placements on the duds.

If a critical node of $q \in Q$ is chosen as the seed, then all $c + u$ critical and unstable nodes will fire, contributing $c + u$ chips to each dud. Therefore, each dud contains at most $n - (c + u) - 1 = d - 1$ chips, for otherwise the added chips will cause it to fire. Since there are $\binom{n}{d}$ ways to choose which of the n nodes will be duds, and $L_{d,d-1}$ ways to place chips on these duds, the first two factors in (5.11) account for the chip placement on the duds.

Now we shall account for the number of ways we may place chips on the u unstable nodes of q. Since nodes containing $n - 1$ chips are critical, unstable nodes contain at most $n - 2$ chips. The greatest number of chips that may be added during a game to an unstable node is $c + (u - 1)$, which implies that the smallest number of chips that an unstable node may contain at the outset is $n - (c + (u - 1)) = d + 1$. Thus, unstable nodes contain between $d + 1$ and $n - 2$ chips.

Within the set of u unstable nodes, there is one that contains the fewest chips, and it may contain anywhere from $d + 1$ to $n - 2$ chips. If two unstable nodes v and w both contain $d + 1$ chips, then there would be at most $c + (u - 2)$ other nodes that could fire and add chips to v and w. These additional chips would increase the number of chips on v and w to at most

$$(d+1)+c+(u-2) = c+d+u-1 = n-1,$$

which is not enough for either of them to fire. Thus, both v and w would be duds, a contradiction. Therefore, if the unstable node containing the fewest chips contains $d+1$ chips, then the unstable node containing the next-to-fewest chips must contain between $d+2$ and $n-2$ chips.

By a similar argument, we see that the node containing the k^{th} fewest chips (for $k = 1,2,\ldots,u$) among the unstable nodes must contain at least $d+k$ and at most $n-2$ chips. (Note that $d+u \le n-2$ because we are considering the case where $c \ge 2$.)

Consider the vector $(x_1,x_2,\ldots,x_u)$, where x_k is the number of chips on the unstable node containing the k^{th} fewest chips. We have just seen that $d+k \le x_k \le n-2$. Subtracting $d+1$ yields $k-1 \le x_k-(d+1) \le n-d-3$, for $k = 1,2,\ldots,u$. Since $x_k \ge d+1$, we see that counting the number of ways to distribute chips onto the unstable nodes is equivalent to counting the number of legal configurations on u nodes where each node can have at most $n-d-3$ chips (cf. characterization (3) of Proposition 3.1). By the definition of $L_{n,m}$ (preceding Theorem 3.3), we have $L_{u,n-d-3}$ ways to distribute the chips. Since there are $\binom{n-d}{u}$ ways to choose which of the $n-d$ non-duds will be unstable and $L_{u,n-d-3}$ ways to place chips on these unstable nodes, the second two factors in (5.11) account for the chip placement on the unstable nodes.

Once we have placed chips on all duds and unstable nodes, it remains only to place chips on the critical nodes. Since all critical nodes must contain $n - 1$ chips, and we have no choice as to which nodes will be critical (with the unstable nodes and duds already decided), the placement of these remaining chips is uniquely determined. Thus, for $c \geq 2$, the total number of legal configurations on K_n with c critical nodes, u unstable nodes, and d duds is

$$\binom{n}{d} L_{d,d-1} \binom{n-d}{u} L_{u,n-d-3}. \qquad \blacksquare$$

We shall find it convenient to simplify the expression (5.11). The next result is an easy consequence of Theorem 3.3.

Corollary 5.5 *With the parameters as in Proposition 5.4, the number of relaxed legal configurations on K_n is*

$$\binom{n}{d}(d+1)^{d-1}\binom{n-d}{u}(c-1)(n-d-1)^{u-1}. \qquad \blacksquare$$

Since the expression in Corollary 5.5 equals zero when $c = 1$, it does not apply in this case. We therefore turn to the case $c = 1$ now.

Proposition 5.6 *The number of relaxed legal configurations on K_n with one critical node is n^{n-1}.*

Proof. A relaxed legal configuration with just one critical node cannot have any unstable nodes since, when the critical node fires, it gives just one chip to each other node, which is not sufficient to fire any of the non-critical nodes. Thus, in the proof of Proposition 5.4, once we count the number of ways to distribute the chips on the duds, the remaining choice for the critical node is determined. We find that the number of relaxed legal configurations on K_n with $c = 1$ critical node and $d = n - 1$ duds is

$$\binom{n}{d} L_{d,d-1} = \binom{n}{n-1} L_{n-1,n-2} = n^{n-1}. \qquad \blacksquare$$

In our investigation of $P_{g,q} = \Pr\{X = g \mid \text{legal configuration is } q\}$, we may now group together all configurations q that have $c \geq 2$ critical nodes, u unstable nodes, and d duds. Each of these has probability $\frac{c}{n}$ of having a game of length $g \geq c$, since one of the c critical nodes must be chosen in order for this to occur. Thus,

$$\sum_{q:c\geq 2} P_{g,q} = \sum_{c=2}^{g} \binom{n}{d}(d+1)^{d-1}\binom{n-d}{u}(c-1)(n-d-1)^{u-1}\left(\frac{c}{n}\right). \tag{5.12}$$

The sum stops at g because we cannot have more than g critical nodes without having a game length longer than g, should a critical node be chosen as the seed.

Note that if a critical node is chosen as the seed, then $c + u = g$ (by Lemma 4.8). Now (5.12) may be simplified to a form involving only n, g, and c:

$$\begin{aligned}\textstyle\sum_{q:c\geq 2} P_{g,q} &= \textstyle\sum_{c=2}^{g} \binom{n}{n-g}(n-g+1)^{n-g-1}\binom{g}{g-c}(c-1)(n-(n-g)-1)^{g-c-1}\left(\frac{c}{n}\right) \\ &= \frac{1}{n}\binom{n}{g}(n-g+1)^{n-g-1}\sum_{c=2}^{g}\binom{g}{c}c(c-1)(g-1)^{g-c-1}. \end{aligned} \tag{5.13}$$

The summation in this last expression may be simplified further.

Lemma 5.7 *For $g \geq 2$, we have* $\sum_{c=2}^{g}\binom{g}{c}c(c-1)(g-1)^{g-c-1} = g^{g-1}$.

Proof. The lefthand side is

$$\begin{aligned}\sum_{c=2}^{g}\binom{g}{c}c(c-1)(g-1)^{g-c-1} &= \sum_{c=2}^{g}\frac{g!}{(c-2)!(g-c)!}(g-1)^{g-c-1} \\ &= g\sum_{c=2}^{g}\frac{(g-2)!}{(c-2)!((g-2)-(c-2))!}(g-1)^{(g-2)-(c-2)} \\ &= g\sum_{c=0}^{g-2}\binom{g-2}{c}(g-1)^{(g-2)-c} \\ &= g(1+(g-1))^{g-2}. \end{aligned}$$

■

Applying Lemma 5.7 to (5.13) yields

$$\sum_{q:c\geq 2} P_{g,q} = \frac{1}{n}\binom{n}{g}(n-g+1)^{n-g-1}g^{g-1} = \binom{n-1}{g-1}(n-g+1)^{n-g-1}g^{g-2}. \tag{5.14}$$

Note that this expression does not depend on q.

A necessary condition for having a game length g at least two is that

$c \geq 2$. Thus, it follows from (5.1), (5.14), and Theorem 5.1 that for $g \geq 2$, we have

$$\Pr\{X = g\} = \sum_{g:c\geq 2} P_{g,q} \cdot \frac{1}{(n+1)^{n-1}} = \binom{n-1}{g-1} \frac{(n-g+1)^{n-g-1} g^{g-2}}{(n+1)^{n-1}}. \qquad (5.15)$$

Proposition 5.6 and Theorem 5.1 together imply that

$$\Pr\{X = 1\} = \frac{n^{n-1}}{n(n+1)^{n-1}} = \frac{n^{n-2}}{(n+1)^{n-1}}. \qquad (5.16)$$

Conveniently, (5.16) agrees with (5.15) with $g = 1$; the remark following Corollary 5.5 necessitated the separate handling of this case here. We delay discussion of the remaining case $g = 0$ until Section 5.4, where we will employ the methods of Chapter 3.

At this point, we may use our analytical results to see if a sequence of burn-off games on a complete graph exhibit one of the main properties of a system in a state of SOC: namely, that the frequency of games and their lengths have a power law relationship (as discussed on p. 10). We show now that the relationship is in fact vertically symmetric and thus does not follow any power law.

Proposition 5.8 *For* $1 \leq g \leq n$, *we have* $\Pr\{X = g\} = \Pr\{X = n - g + 1\}$.

Proof. Using (5.15), we have

$$\begin{aligned}\Pr\{X = n-g+1\} &= \binom{n-1}{n-g}\frac{(n-(n-g+1)+1)^{n-(n-g+1)-1}(n-g+1)^{n-g+1-2}}{(n+1)^{n-1}} \\ &= \binom{n-1}{g-1}\frac{g^{g-2}(n-g+1)^{n-g-1}}{(n+1)^{n-1}} \\ &= \Pr\{X = g\}. \qquad \blacksquare\end{aligned}$$

We can thus conclude that a burn-off chip-firing game on a complete graph will not possess one of the key features of a system that exhibits SOC. That is, we do not find a power law relationship (as discussed in Section 1.4) between the size (i.e. game length) and frequency (i.e. probability) of events in the system. A complete graph features total communication among its nodes during a chip-firing game. In other words, any event triggered by one node will affect every other node. Systems that feature SOC generally do not communicate so thoroughly. It is typical (see, e.g., [3] and [7]) for SOC models to represent the individuals in a system as nodes with low degree on a large graph.

5.4 Establishing these results using the methods of Chapter 3

In this section, we show how our results from Chapter 3 verify our results for complete graphs established in the present chapter.

First we recover the formula $L_{n,n-1} = (n+1)^{n-1}$ (see p. 40) using Theorem 4.1 for enumerating the legal configurations on a general connected graph. For K_n, the special node x in Theorem 4.1 makes $G^* \cong K_{n+1}$. Thus, Theorem 3.5 is really a corollary of Theorem 4.1, and we have $L_{n,n-1} = \tau(K_{n+1}) = (n+1)^{n-1}$ by Cayley's Formula (see, e.g., [19]).

We now enumerate the games of length zero on K_n using Proposition 4.7. Recall that for each node v, we remove the edge $\{x, v\}$ from G^* and enumerate the spanning trees of the resulting graph. We use the Matrix Tree Theorem (see, e.g., [19]) to count the spanning trees. We reduce the Laplacian matrix of G^* by eliminating the row and column associated with v, resulting in the $n \times n$ matrix

$$\mathrm{A} = \begin{bmatrix} n & -1 & -1 & & -1 & -1 \\ -1 & n & -1 & \cdots & -1 & -1 \\ -1 & -1 & n & & -1 & -1 \\ & \vdots & & \ddots & & \vdots \\ -1 & -1 & -1 & & n & -1 \\ -1 & -1 & -1 & \cdots & -1 & n-1 \end{bmatrix}.$$

To enumerate the spanning trees of G^*, we find det(A). Denoting the rows of A by $(r_j)_{j=1}^{n}$ and performing the elementary row operations $\sum_{j=1}^{n} r_j \to r_1'$, followed by $r_1' + r_j \to r_j$ (for $j = 2, \ldots, n$), we find the determinant to be $(n-1)(n+1)^{n-2}$. There are n nodes, so the number of pairs (q, v) with $q \in \mathcal{R}$ and $v \in V$ that result in a game of length zero when v is the seed is $n(n-1)(n+1)^{n-2}$. Given q, a node v is selected as the seed with probability $1/n$. By Theorem 5.1, each legal configuration on K_n occurs with probability $1/(n+1)^{n-1}$. Thus, conditioning on the choice of (q, v) with $q \in \mathcal{R}$ and seed v we find that

$$\begin{aligned}\Pr\{X=0\} &= \sum_{(q,v)} \Pr\{X=0 \mid (q,v)\} \Pr\{v|q\} \Pr\{q\} \\ &= n(n-1)(n+1)^{n-2}(1/n)(1/(n+1)^{n-1}) \\ &= \frac{n-1}{n+1}. \end{aligned} \tag{5.17}$$

As a check on our work, we pause to show that the probabilities in (5.15), (5.16), and (5.17) sum to one.

Proposition 5.9 For each positive integer n, we have

$$\frac{n-1}{n+1} + \frac{n^{n-2}}{(n+1)^{n-1}} + \sum_{g=2}^{n} \frac{\binom{n}{g}(n-g+1)^{n-g-1}g^{g-1}}{n(n+1)^{n-1}} = 1. \tag{5.18}$$

Proof. We first manipulate (5.18) into a form that is amenable to a combinatorial proof. Multiplying both sides of the equation by $n(n+1)^{n-1}$, we see that it will be equivalent to establish

$$(n-1)n(n+1)^{n-2} + n^{n-1} + \sum_{g=2}^{n} \binom{n}{g}(n-g+1)^{n-g-1}g^{g-1} = n(n+1)^{n-1}.$$

As noted following equation (5.16), the second term may be absorbed into the sum. Collecting together the remaining terms external to the sum, we see that (5.18) is equivalent to

$$\sum_{g=1}^{n} \binom{n}{g}(n-g+1)^{n-g-1}g^{g-1} = 2n(n+1)^{n-2}. \tag{5.19}$$

Finally, using the identity $\binom{n}{g} = \binom{n+1}{g}\frac{n-g+1}{n+1}$ we reduce (5.19) to

$$\sum_{g=1}^{n} \binom{n+1}{g} g^{g-2}(n+1-g)^{n-1-g} g(n+1-g) = 2n(n+1)^{n-1}. \tag{5.20}$$

To see that (5.20) holds, first observe that the right side enumerates the pairs $(T,\vec{e})$, where T is a spanning tree of K_{n+1} for which one edge e (of its n edges) has been distinguished and oriented (in one of the two possible directions). The left side also enumerates these pairs. Given $(T,\vec{e})$, notice that deleting the oriented edge $\vec{e}$ from T leaves behind a spanning forest of K_{n+1} with two components L, R (that we may consider ordered from left to right). If $|V(L)| = g$, for an integer g with $1 \le g \le n$, then $|V(R)| = n+1-g$. Conversely, given such a spanning forest, we can recover $(T,\vec{e})$ by selecting a node x of L and a node y of R and letting $\vec{e} = (x,y)$. On the left side of (5.20), the factor $\binom{n+1}{g}$ accounts for the selection of $V(L)$ (hence for the selection of $V(R)$). Since L, R are, respectively, spanning trees of the induced (complete) subgraphs $K_{n+1}[V(L)]$, $K_{n+1}[V(R)]$, the factors g^{g-2} and $(n+1-g)^{n-1-g}$ are delivered by Cayley's Formula. Finally, the factors g, $(n+1-g)$ count the number of ways to select the vertices $x \in V(L)$ and $y \in V(R)$ determining $\vec{e}$. ■

Finally, we recapture the general formula (5.15), namely that for $g \ge 2$,

$$\Pr\{X = g\} = \binom{n-1}{g-1}\frac{(n-g+1)^{n-g-1}g^{g-2}}{(n+1)^{n-1}}.$$

We appeal to Theorem 4.9. For each v in K_n, we find all subtrees having g nodes that include v. There are $\binom{n-1}{g-1}$ ways to choose the $g-1$ nodes $u \neq v$ for these subtrees; each choice results in a complete subgraph K_g of K_n, for which there are g^{g-2} spanning trees (by Cayley's Formula). Now we delete each subtree in turn, counting the number of legal configurations on the resulting graph. When K_g is deleted from K_n, the graph K_{n-g} remains. By Theorem 4.1 (or Theorem 3.5), the number of legal configurations on K_{n-g} is $(n-g+1)^{n-g-1}$. Thus, Theorem 4.9 yields $\binom{n-1}{g-1}g^{g-2}(n-g+1)^{n-g-1}$ as the number of ways to have a game of length $g \geq 2$ on K_n. Since each legal configuration is equally probable by Theorem 5.1, we see that

$$\Pr\{X = g\} = \binom{n-1}{g-1}\frac{(n-g+1)^{n-g-1}g^{g-2}}{(n+1)^{n-1}},$$

and have thus re-derived (5.15).

In Chapter 3, we investigated burn-off games on complete graphs with a direct approach. Our methods of Chapter 4 were more general. In the present chapter, we have re-established several important results from Chapter 3 using the methods of Chapter 4. We revisited the link between Cayley's Theorem and relaxed legal configurations, enumerated the pairs $(q,v) \in \mathcal{R} \times V$ leading to games of length zero, and confirmed the probability distribution of burn-off game lengths. Of final note is the fact that we were able to obtain an independent, combinatorial proof that these probabilities sum to one. We conclude the dissertation with a brief list of suggestions for further research.

5.5 Further research

Early models of SOC allowed stresses to escape the system. In [3], earthquake stresses that reach the edge of the grid are considered to be lost. In [7], avalanches can spill sand out of the system as if it were falling from a table. To represent these models as chip-firing games, at least one node must have the capacity to hold an infinite number of chips. This leads to a natural question: can a chip-firing game with such a node be reconciled with the results in this dissertation concerning burn-off games?

Many modified chip-firing games have been studied, including games with mutating edges [8] and games played in parallel [5]. It would be interesting to determine whether our results extend to these situations as well.

Finally, Theorems 4.9 and 4.13 together provide us with a method for determining the probability distribution of the burn-off game length on a general connected graph. We do not, however, have a closed formula for this distribution as we do in Chapter 5 for complete graphs. Finding such a formula presents a tantalizing open problem.

Appendix 1

Glossaries

Glossary of notation

$\sim$	is a neighbor of
$\tau(G)$	the number of spanning trees of the graph G
$indeg(v)$	in a directed graph, the number of incoming arcs incident with v
$outdeg(v)$	in a directed graph, the number of outgoing arcs incident with v

Glossary of terminology

configuration	on a graph $G = (V,E)$, a distribution $C : V \rightarrow \mathbb{N}$ of chips on V
connected	describes a graph in which every pair of nodes is joined by a path
critical	in a classical chip-firing game (e.g., [6]), describes a node that contains as many chips as its degree; in a burn-off game, describes a node that will fire if a chip is added to it
critical number	in a chip-firing game, the least number of chips a node can contain for it to fire
deletion (edge)	the removal of a single edge from a graph, not including its ends
deletion (node)	the removal of a single node, and all edges incident with it, from a graph
fire	the process in a chip-firing game by which the chips on a node are redistributed
independent	describes a set of nodes inducing a subgraph with an empty edge set
induced subgraph	of a graph $G = (V,E)$, with $V' \subseteq V$, the subgraph $H = (V',E')$ of G, where $E' = \{\{x,y\} \in E : x,y \in V'\}$
language	a collection of words
legal	describes a configuration that may result from relaxing a supercritical configuration, or result from relaxing another legal configuration that has been seeded
length of a game	the number of nodes that fired during a chip-firing game, during one iteration of a seed-to-relaxation sequence
relax	the process in a chip-firing game that begins with seeding a node and continues until no nodes can fire

relaxed	describes a configuration in which no node can currently fire
seed	as a noun, the node to which a chip is added to initiate a chip-firing game; as a verb, the process of adding a chip to such a node
simple	describes a graph that has no loops or multiple edges
supercritical	describes a node that contains more than a critical number of chips, or describes a graph in which every node is supercritical
word	the concatenation of labels that corresponds to the sequence of nodes that fire as a legal configuration relaxes to another legal configuration

Appendix 2

Smallbasic code

The following code, written in Smallbasic, was used to produce the results shown in Figure 1.2.

```
option base 1
dim gameLengths(0 to 4)
adjMatrix = [0,1,0,0;1,0,1,1;0,1,0,1;0,1,1,0]
config = [0;0;0;0]
critNumbers = [1;3;2;2]
input "How many games";numGames
for thisGame = 1 to numGames
  ' Initialize
  gameLength = 0 : canFire = true
  ' Choose a node at random as seed
  seed = int(4 * rnd + 1)
  ' Add a chip to the seed
  config(seed,1) = config(seed,1) + 1
  repeat
    didFire = false
    for thisNode = 1 to 4
      if config(thisNode,1) > critNumbers(thisNode,1) then
        gameLength = gameLength + 1
        tmpVector01 = [0;0;0;0]
        tmpVector01(thisNode,1) = 1
        tmpVector02 = adjMatrix * tmpVector01
        config(thisNode,1) = config(thisNode,1) -
(critNumbers(thisNode,1) + 1)
        config = config + tmpVector02
        didFire = true
      endif
    next thisNode
  until didFire = false

  gameLengths(gameLength) = gameLengths(gameLength) + 1

next thisGame
print gameLengths
```

The following code, written in Smallbasic, was used to produce the results shown in Figure 1.3.

```
dim arrGrid(0 to 51,0 to 51)
dim arrGameLengths(50)

' Initialize arrGrid
for x = 1 to 50
  for y = 1 to 50
    arrGrid(x,y) = int(4 * rnd + 1)
  next y
next x

input "Number of games";conNumberGames

for ctrGameNumber = 1 to conNumberGames
  ' select seed
  x = int(50 * rnd + 1)
  y = int(50 * rnd + 1)
  ' perturb seed
  arrGrid(x,y) = arrGrid(x,y) + 1
  ' see if any node can fire
  conGameLength = 0
  repeat
    flgDidFire = false
    for i = 1 to 50
      for j = 1 to 50
        if arrGrid(i,j) > 3 then
          flgDidFire = true
          conGameLength = conGameLength + 1
          arrGrid(i,j) = arrGrid(i,j) - 4
          arrGrid(i-1,j) = arrGrid(i-1,j) + 1
          arrGrid(i+1,j) = arrGrid(i+1,j) + 1
          arrGrid(i,j-1) = arrGrid(i,j-1) + 1
          arrGrid(i,j+1) = arrGrid(i,j+1) + 1
        endif
      next j
    next i
  until flgDidFire = false
  ' record the game length
  ' first, collect all games of length longer than 50 into
one category
  if conGameLength > 50 then conGameLength = 50
  arrGameLengths(conGameLength) =
arrGameLengths(conGameLength) + 1
next ctrGameNumber

print arrGameLengths
```

References

[1] P. BAK AND K. CHEN, Self-organized criticality, *Scientific American* (Jan. 1991), 46-53.

[2] P. BAK, C. TANG, AND K. WIESENFELD, Self-organized criticality, *Phys. Rev. A (3)* **38** (1988), 364-374.

[3] P. BAK AND C. TANG, Earthquakes as a self-organized critical phenomenon, *J. Geophys. Res.* **94** (1989), 15635-15637.

[4] N. L. BIGGS, Chip-firing and the critical group of a graph, *J. Algebraic Combin.* **9** (1999), 25-45.

[5] J. BITAR AND E. GOLES, Parallel chip firing games on graphs, *Theoret. Comput. Sci.* **92** (1992), 291-300.

[6] A. BJORNER, L. LOVÁSZ, AND P. W. SHOR, Chip-firing games on graphs, *European J. Combin.* **12** (1991), 283-291.

[7] D. DHAR, Self-Organized critical state of sandpile automaton models, *Phys. Rev. Lett.* **64** (1990), 1613-1616.

[8] K. ERIKSSON, Chip-firing games on mutating graphs, *SIAM J. Discrete Math.* **9** (1996), 118-128.

[9] K. ERIKSSON, Node firing games on graphs, *Contemp. Math.* **178** (1994), 117-127.

[10] W. FELLER, *Introduction to Probability Theory and Its Applications*, vol. 1, 3rd ed., John Wiley & Sons, New York, 1971.

[11] H. JENSEN, *Self-Organized Criticality: Emergent Complex Behavior in Physical and Biological Systems*, Cambridge University Press, Cambridge UK, 1998.

[12] S. KAUFFMAN, *At Home in the Universe*, Oxford University Press, New York, 1995.

[13] B. KOLMAN, R. BUSBY, AND S. ROSS, *Discrete Mathematical Structures*, 4th ed., Prentice Hall, New Jersey, 2000.

[14] S. H. LIU, T. KAPLAN, AND L. H. GRAY, Geometry and dynamics of deterministic sand piles, *Phys. Rev. A (3)* **42** (1990), 3207-3212.

[15] K. NAGEL AND M. PACZUSKI, Emergent traffic jams, *Phys. Rev. E* **51** (1995), 2909-2918.

[16] J. RAUCH, Seeing around corners, *Atlantic Monthly* **289** (2002), 35-48.

[17] S. ROBINSON, The power grid as complex system, *SIAM News* **36** (2003).

[18] D. STASSINOPOLOUS AND P. BAK, Democratic reinforcement: A principle for brain function, *Phys. Rev. E (3)* **51** (1995), 5033-5039.

[19] D. B. WEST, *Introduction to Graph Theory*, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2001.