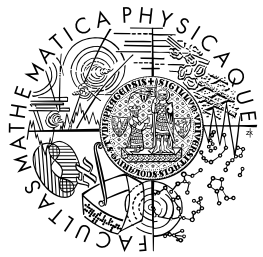

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Parth Mittal

Pseudorandom walks and chip firing games

Computer Science Institute of Charles University

Supervisor of the master thesis: Prof. Mgr. Michal Koucký, PhD

Study programme: Master of Computer Science

Study branch: Theoretical Computer Science

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor Prof. Michal Koucký for his patience, constant motivation, and consistently asking the right questions.

I would also like to thank my friends and family for their support, and my parents in particular for always encouraging my education.

Title: Pseudorandom walks and chip firing games

Author: Parth Mittal

Institute: Computer Science Institute of Charles University

Supervisor: Prof. Mgr. Michal Koucký, PhD, Computer Science Institute of Charles University

Abstract: We study two deterministic analogues of random walks. The first is the chip-firing game, a single player game played by moving chips around a directed graph, popularised by Björner and Lovász. We find an efficient simulation of boolean circuits and Turing machines using instances of the chip-firing game – after assigning a fixed strategy to the player. The second is the Propp machine, or the rotor router model, a quasirandom model introduced by Priezzhev. We improve results of Kijima et al. and show a bound of $\mathcal{O}(m)$ on the discrepancy of this process from a random walk on d-regular graphs with m edges.

Keywords: random walks chip firing games rotor-router computation discrepancy

Contents

Introduction

In this thesis, we study two deterministic analogues of random walks. In Chapter 1 we study the chip-firing game, which is a single-player game played on a (directed) graph G, with a non-negative integer number of "chips" on each vertex. While a prototype of the game on the infinite line was used in a paper of Spencer about balancing vectors [1], the general game was popularised by Björner, Lovász and Shor [2], and is of independent interest. Our focus is on the computational power of this game. We show effective simulations of boolean circuits and Turing machines with instances of the game on directed graphs.

In Chapter 2 we look at the Propp machine, or the rotor-router model, which is a quasirandom model proposed by [3], and rediscovered several times in various areas. Our focus is on the effort to use Propp machines to derandomize randomized algorithms; we improve known bounds on the discrepancy of this process from random walks in some special cases.

Concurrent work

During the final days of preparing this thesis, while doing a further literature search we encountered a paper by Goles and Margenstern [4] which proves essentially the same results as contained in Chapter 1. They use similar techniques, except that their constructions are on undirected graphs, while we use directed graphs. Our Theorem 1.7 and Theorem 1.9 are not in their paper, and can be seen as new. Further, our Theorem 1.8 is more general, since it does not fix the strategy of the player. We obtained our results independently.

Chapter 1

Computing using the chip-firing game

We consider the following single player chip-firing game: A directed graph $G = (V, A)$ is given, with an initial distribution $c_0 : V \to \mathbb{N}$ of chips at each vertex. A legal move consists of selecting a vertex with at least as many chips as its outdegree, and then "firing" one chip along each of its outgoing edges.

Björner, Lovász and Shor [2] showed (for undirected graphs) that the player strategy has no effect on whether the game is finite, or the terminating configuration if it is finite; these properties are decided entirely by the graph and the starting configuration. These results were extended to directed graphs by Björner and Lovász in [5].

Our approach to the game was motivated by (at least surface level) similarities between Conway's Game of Life and the chip-firing game. In particular, our *synchronous strategy* for the player is inspired by the parallel nature of the Game of Life.

Our main results are simulations of boolean circuits and Turing machines using instances of the chip-firing game. While the boolean circuit simulation is not sensitive to player strategy (this is proved and used in Theorem 1.8), the Turing machine simulation works only for the synchronous strategy. Accompanying the simulations are hardness results – we construct an undecidable language over infinite instances of the chip-firing game, and a PSPACE-hard language over finite instances.

Section 1.1 contains basic notation and definitions. Section 1.2 contains the boolean circuit simulation, and Section 1.3 contains the Turing machine simulation. Finally, Section 1.4 contains the hardness results corresponding to each simulation.

1.1 Preliminaries

For much of this chapter, we will be concerned with the following player strategy for the chip-firing game.

Definition 1.1 (Synchronous strategy)**.** *If no vertex can be fired, there is nothing to be done. Otherwise, compute for each vertex v, the value $f(v) = \lfloor c(v)/\deg^+(v) \rfloor$ – the number of times it can be fired without receiving any additional chips. And now fire each vertex v $f(v)$ times (in any order). Repeat this until no vertex can be fired (or forever if the game is infinite).*

Note that the order in which we fire the vertices within a single step does not matter, since at each vertex we only use chips that were available at the beginning of the step. Hence, one can think of this strategy as simultaneously firing all the vertices that can be fired, which is why we call it the synchronous strategy.

Throughout this chapter we draw several diagrams of chip-firing games in action using the following notation:

For a directed graph $G = (V, A)$ with initial configuration $c_0 \colon V \to \mathbb{N}$, we define $c_i(v)$ for all $i \geq 1$ as the number of chips at v at time-step i, for some fixed strategy of the player. In diagrams we represent vertices with rectangles with the vertex label inside and $\{c_i(v)\}_i$ as a string next to the vertex v. Whenever $c_i(v) \geq \deg^+(v)$, it is colored red, to signify that it will fire in the next time-step. To avoid clutter, if $c_{i+1}(v) = c_i(v)$, we replace it with a -.

Next, we define the notion of a gadget, which intuitively corresponds to a small unit of computation.

Definition 1.2 (d-gadget)**.** *A directed (sub)graph $H = (V, A)$, and a partial configuration of chips c on it is a length-d gadget, or simply a d-gadget if it can be partitioned into layers $V_0, \ldots, V_d$ such that:*

- *There is a path from V_0 to V_d. More formally, there is at least one pair of vertices (u, v) with $u \in V_0$, and $v \in V_d$ such that H has a path from u to v.*
- *Edges can only go from one layer to the next. That is, if (u, v) is an edge of H, there must exist an i such that $u \in V_i$ and $v \in V_{i+1}$.*
- *The partial configuration c is defined on $V_1, \ldots, V_{d-1}$, and it forms a stable configuration of H if it is extended to be 0 everywhere else. That is, each vertex of H has strictly fewer chips than its out-degree (and hence no vertex can fire). Intuitively, this means H doesn't compute anything until it receives some input.*

Further, we call V_0 and V_d the *input* and *output* of H respectively. Throughout this chapter, we will *give input* to a gadget by extending its partial configuration to the input layer in some way, and extending it to 0 on the output layer.

Note that this doesn't lose any generality over requiring a gadget to be a directed acyclic graph, with at least one source-sink path. In particular, let G be a DAG with maximum source-sink distance d. Then we can partition the graph G such that a vertex v is in layer i if the *longest* path from a source to v is of length i. To satisfy the requirement that edges do not skip layers, we can extend any such edge into a path by inserting vertices on it. See Figure 1.1 for an example.

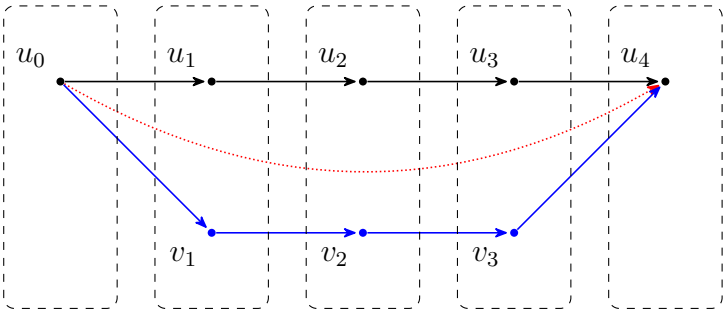

Figure 1.1 Converting a DAG to a d-gadget: The red edge is dropped, the blue edges and vertices are added.

A consequence of this definition is that if there is a path from u to v, with $u \in V_i$ and $v \in V_j$ respectively, the path has length $j - i$. Intuitively, a d-gadget is a unit of computation which takes d steps of the synchronous strategy to compute its output from its input.

1.2 Computing boolean circuits

In this section, we consider the problem of representing boolean circuits using instances of the chip firing game. Recall that boolean circuits are a non-uniform model of computation where for each natural number n, we have a directed acyclic graph $\mathcal{C}_n$ where each vertex is either

- a logical `AND`, `OR`, or `NOT` gate with incoming edges from each of its inputs
- or an input vertex.

$\mathcal{C}_n$ has exactly n input vertices, whose value can be specified to be 0 or 1, and every other vertex computes its value from its inputs. m vertices of $\mathcal{C}_n$ are identified as its output, and $\mathcal{C}_n(x)$ is the string of values of these vertices on the input $x \in \{0,1\}^n$. We will show the following theorem:

Theorem 1.3. *For any family of boolean circuits $\{\mathcal{C}_n\}$, there exists a family of gadgets $\{(G_n, c_n)\}$ such that:*

- *$|G_n| \leq p(|\mathcal{C}_n|)$ for some polynomial p.*
- *For each natural number n, and each input $x \in \{0,1\}^n$, the partial configuration c_n can be extended to the input layer such that the chip firing game with the synchronous player strategy on this instance computes $\mathcal{C}_n(s)$. Further, each input vertex receives either x_i or $1 - x_i$ chips, where x_i is the corresponding bit of the input (see Figure 1.5 and Figure 1.6). The output $\mathcal{C}_n(x)$ is given by the number of chips on the output layer vertices of G_n after the chip-firing stops (in atmost D steps), in the same encoding as the input.*
- *G_n is an D-gadget for $D = \mathcal{O}(d(n) \cdot \log(s(n)))$ where $d(n)$ and $s(n)$ are the depth and size of $\mathcal{C}_n$ respectively.*

1.2.1 The natural choice

On the way to Theorem 1.3, we consider a few ways to represent 0s and 1s as chips – the first attempt is to use a single chip to represent a 1, and no chip to represent a 0. At first sight, this might seem strange, since 0 chips cannot move. We step around this by imagining the computation as a wave, which moves forward one layer of the gadget at a time.

Consider the *fundamental gadget* in Figure 1.2. Note that technically it isn't a gadget, since we did not specify an initial configuration on its inner vertices (there is only one, g). It turns out that by varying $c(g)$, the fundamental gadget computes different boolean functions.

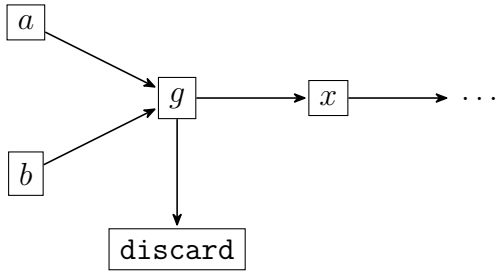

Figure 1.2 The fundamental gadget.

In particular, setting $c(g)$ to 0 yields an `AND` gate (see Figure 1.3 and Figure 1.4).

On the other hand, if $c(g) = 1$, we have an `OR` gate (to see this, look at Figure 1.4 and note that g would fire after receiving one token if $c(g)$ was 1).

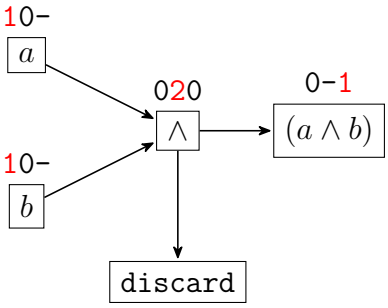

Figure 1.3 With $c(g) = 0$, and both the inputs set to 1, the output is 1.

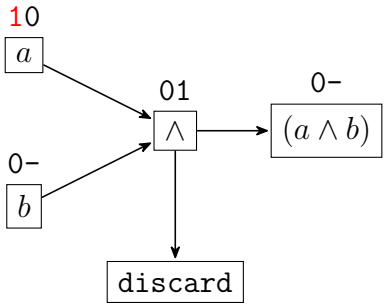

Figure 1.4 With only one of the inputs set to 1, the output is 0.

However, we cannot make NOT gates with this graph, or even this model. In particular, we have the following lemma:

Lemma 1.4 (Monotonicity). *Let $H = (V, E)$ be any $d-$gadget with input V_0 and output V_d, and partial initial configuration c. Suppose that when the inputs $S \subset V_0$ are set to 1, the outputs $T \subset V_d$ become 1 (after d steps). Then for any superset S' of S, if the inputs S' are set to 1, the outputs T still become 1.*
Or, in simpler language, making additional inputs 1 cannot flip any outputs from 1 to 0.

Proof. Let c_0 and c'_0 be initial configurations corresponding to the inputs S and S' respectively, or more formally:

$$c_0(v) = \begin{cases} c(v), & v \in \bigcup_{i=1}^{d-1} V_i \\ 1, & v \in S \\ 0, & \text{otherwise} \end{cases}$$

And c'_0 defined similarly for S'. Further for $t \geq 1$ let c_t and c'_t denote the configuration of chips after t steps of the synchronous strategy from the respective starting configurations.

First, note that neither computation can "run ahead". That is, after t steps of operation, for $k > t$, c_t and c'_t are equal on all vertices in V_k. This follows from the restriction that d-gadgets may not have edges that skip layers.

We will show by induction that for each $v \in V_t, c_t(v) \leq c'_t(v)$. The base case follows from the fact that $S \subseteq S'$. Suppose the induction holds until the k-th layer. Then note that for $u \in V_k$, $c_k(u) \geq \deg^+(u)$ implies $c'_k(u) \geq \deg^+(u)$. And hence if u fires at the k-th step with the initial inputs S, it also fires at the k-th step with the inputs S'. Hence each vertex $v \in N(u)$ receives at least as many chips under c'_k as it does under c_k. This gives us $c_{k+1}(v) \leq c'_{k+1}(v)$ for each $v \in V_{k+1}$, as desired. □

1.2.2 Bigger bits are better

This brings us to our second attempt. We use two vertices to represent a single bit, as follows:

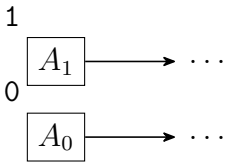

Figure 1.5 This is a 1.

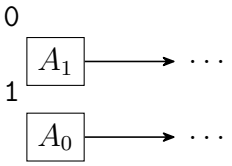

Figure 1.6 This is a 0.

We will call this two vertex representation a *full-bit*, and the two vertices that are part of it will be called *half-bits*. NOT gates are now easy – all we have to do is swap adjacent half-bits (see Figure 1.7).

By examining each half-bit in the tables for AND (see Table 1.1), we can easily design this gate over the full-bit representation. In particular, the upper half-bit of the output is the AND of the upper half-bits of the input, and the lower half-bit is the OR of the lower half-bits of the input. Hence we can combine the previously described gates to compute the AND over full-bits (see Figure 1.8). By following the same procedure for the OR gate we see that the upper bit of the output is the OR of the upper bits of the input, and the lower bit is the AND of the lower bits.

It is almost time to claim that we have a proof for Theorem 1.3; the final barrier is fanout. In particular, our designs are limited to a fanout of 1. We can fix this with a copy gadget that copies a half-bit (see Figure 1.9). Naturally, we combine two of these in parallel to copy a full bit, and many of them in a binary tree if we need larger fanout.

Synchronization: a brief interlude

We would like that all outputs of each layer of G_n are available at the same time-step. While this requirement follows directly from the last condition of Theorem 1.3, the real reason to have it is that it helps us simulate Turing machines

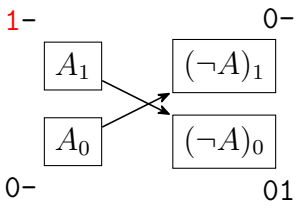

Figure 1.7 A NOT gate.

	01	10
01	01	01
10	01	10

Table 1.1 Table for AND.

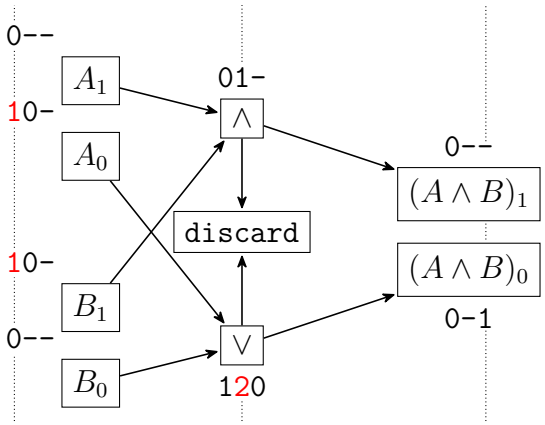

Figure 1.8 An AND gate over full-bits in action, with the inputs 0 and 1.

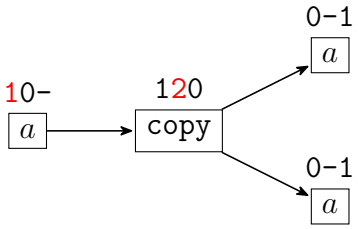

Figure 1.9 A half-bit copy gadget in action.

easily in the next section. As we shall see, it is not difficult to achieve.

In particular, note that the AND and OR gates (see Figure 1.8) over full-bits are 2-gadgets, while the NOT gate is a 1-gadget. Hence we will add a delay of 1 unit to each output of the NOT gate by replacing its outgoing edge with a path of length 2 (like in Figure 1.1).

Note also that gates of a circuit within the same layer may have different fanout, and hence different delay due to copy gadgets when we translate them to chip-firing instances. For some fixed layer, let m be the maximum fan-out of all gates in this layer. Then we will build a binary tree with 2^k leaves, where $2^{k-1} < m \leq 2^k$, of copy gadgets at *each* gate's output, with the extra copies all routed to the universal discard vertex.

The `discard` vertex

The fundamental gadget (Figure 1.2) and hence the logic gates over full-bits (Figure 1.8) feature a `discard` vertex, where extra chips are sent. While it would suffice to have a separate copy of this vertex for every logic gate for Theorem 1.3, we will eventually reuse discarded chips in the next section. Hence, we introduce a universal `discard` vertex, which will collect all such chips.

Proof of Theorem 1.3. We describe the mapping from $\mathcal{C}_n$ to (G_n, c_n) for some fixed n. Before we translate the circuit itself, we will partition it into layers. In particular, each gate will go into the layer i such that its furthest distance from an input bit that it uses is i (i.e. its depth in the circuit).

First, the input layer contains $2n$ vertices, where each consecutive pair of vertices represents a single bit of the input to $\mathcal{C}_n$ (using the encoding in Figure 1.5 and Figure 1.6). This is the extension to c_n which is different for each input x to $\mathcal{C}_n$.

Now, for each subsequent layer, we replace each gate in the boolean circuit with its corresponding gadget in the graph, adding edges from the outputs of gadgets (or their copies) from previous layers as required. Next we add the copy gadget binary tree to the output of each vertex to achieve the fanout we want.

Note that there may be edges arising from $\mathcal{C}_n$ that skip layers in this construction – we deal with them as in Figure 1.1, by extending them into paths of suitable length. Similarly, there may be edges to the universal `discard` vertex from vertices in different layers. We extend edges from earlier layers by the same method.

And now by induction each layer of $\mathcal{C}_n$ becomes a gadget which matches its computation, and hence G_n computes the same function as $\mathcal{C}_n$. It remains to show that G_n is of size polynomial in $s(n)$, the size of $\mathcal{C}_n$. The gadgets for `NOT`, `AND` and `OR` all have constant size, and each occurrence of one of these gadgets corresponds to a gate in the boolean circuit. The fanout of each gate is atmost $s(n)$ and we use atmost $2s(n)$ copy-gadgets to achieve this fanout. Hence there are $\mathcal{O}(s(n))$ vertices in G_n due to the logic gates of $\mathcal{C}_n$, and $\mathcal{O}(s(n)^2)$ vertices in the copy-gadgets due to the fanout.

Before we count the vertices added due to extending edges into paths, we need to bound the length of the gadget (G_n, c_n). G_n has length at most $D = \mathcal{O}(d(n) \cdot \log s(n))$, since the copy gadget trees have length $\mathcal{O}(\log s(n))$, the gadgets for the logic gates have constant length, and there are $d(n)$ layers in $\mathcal{C}_n$ (by definition, since $d(n)$ is the depth of $\mathcal{C}_n$).

Extending edges of $\mathcal{C}_n$ into paths can add at most D vertices per edge. Since there are $\mathcal{O}(s(n)^2)$ edges in $\mathcal{C}_n$, atmost $\mathcal{O}(s(n)^4)$ vertices are added to G_n for extending these edges to paths (since $d(n) \leq s(n)$, $\log s(n) \leq s(n)$). Similarly, the edges to the `discard` vertices may be extended to paths of length $\mathcal{O}(d(n) \cdot \log s(n))$, adding $\mathcal{O}(s(n)^3)$ vertices in total across all the logic gates of $\mathcal{C}_n$ (since the gadget corresponding to each logic gate has atmost 2 edges going to the `discard` vertex). Combining all of the bounds above, G_n has $\mathcal{O}(s(n)^4)$ vertices, and coupled with c_n forms a gadget of length $\mathcal{O}(d(n) \log s(n))$. □

1.3 From circuits to Turing machines

In this section, we extend our simulation of boolean circuits to Turing machines. The main result of this section is Theorem 1.7.

Recall that computation with Turing machines is local, in the sense that the contents of some tape cell at some time-step depend only on the contents of 3 cells in its neighbourhood, the state of the machine, and the head position at the previous time-step. Suppose that we locally encode both the state of the machine and the head position in the standard way – for example, by enhancing the alphabet of the machine's tape – then all the information needed to compute the contents of a tape cell is available in its neighbourhood. Further, if we interpret each letter of the alphabet as a fixed-width binary string, then each bit of the tape cell is a boolean function of the 3 neighbouring cells in the previous time-step. Note that boolean functions over b bits can be represented by circuits of size $\mathcal{O}(b \cdot 2^b)$, and hence we can build a boolean circuit of size $\mathcal{O}(s)$ and constant depth that simulates one step of a given s-space single-tape Turing machine's operation. This construction is standard, see Sipser's book [6] for more details.

Now we can stack together t copies of this circuit (with the outputs of each copy wired up to the inputs of the next copy) to get a circuit which simulates t steps of the Turing machine. Hence by using Theorem 1.3 we can construct a graph G and an initial configuration c such that the chip-firing game played with the synchronous strategy simulates t steps of the Turing machine. Note that the size of this graph is $\Omega(s \times t)$, even if the machine does nothing. In the rest of this section, we will show a modified construction that can simulate s-space Turing machines with graphs of size $\text{poly}(s)$.

Consider the circuit $\mathcal{C}_n$ corresponding to a single step of the Turing machine M, and the gadget G_n from Theorem 1.3 corresponding to $\mathcal{C}_n$. The main idea is to wire up the outputs of G_n to its inputs. The hope is that G_n simulates a step of the Turing machine M, and then its output passes through to the input again, simulating another step, and so on. This is almost enough, except for the *slight* problem that the gadgets for individual gates do not always return to their initial configuration after inputs pass through them. For example, in Figure 1.8, neither of the vertices labeled $\wedge$ and $\vee$ return to their initial configuration. The solution is to (yet again) enhance our representation of a bit: we will send the complement of a bit right behind it.

More precisely, we will aim to maintain the invariant that each half-bit is followed by its complement after each layer of computation. Intuitively, it helps to think of this complement bit as a *cleanup* wave following the computation wave closely. In the next lemma, we prove that this approach returns gates based on the fundamental gadget to their initial configuration.

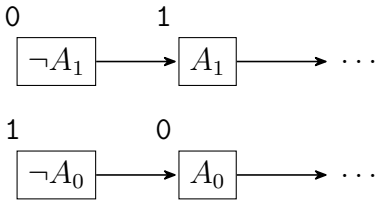

Figure 1.10 This is a 1.

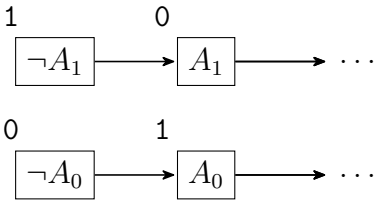

Figure 1.11 This is a 0.

Lemma 1.5 (Cleanup Signal)**.** *Let (G, c) be an AND or OR gadget over half-bits, i.e. the fundamental gadget with $c(g) = 0$, and $c(g) = 1$ respectively. Then after any pair of inputs (a, b) followed by $(\neg a, \neg b)$, the gadget returns to its initial configuration. Further, the cleanup signal propagates, i.e., if the gadget outputs $f(a, b)$, it outputs $\neg f(a, b)$ at the next time-step.*

Proof. The vertex g receives exactly 2 chips for any pair (a, b) and its complement $(\neg a, \neg b)$. Since it has out-degree 2, it fires exactly once, and returns to $c(g)$ chips at the end of the process.

To see the second part of the lemma, note that since g fires exactly once, if $f(a, b) = 1$ it cannot fire again in the next step, and outputs 0. On the other hand if $f(a, b) = 0$, it must fire in the next step, and hence outputs 1. □

Now since our construction for AND and OR gates was based on combining fundamental gadgets, it works as is, and the cleanup signal resets these gadgets to their initial configuration (except for the universal discard vertex, which we will address separately). Note that since the NOT gate merely switches its inputs, it is unaffected by any input passing through it.

Copy gadgets and synchronization, part II

The only thing left to do is consider how copy gadgets behave with the cleanup signal. Indeed, on the input a followed by $\neg a$, they produce the output a and $\neg a$ on both edges leaving the copy vertex, but the vertex does not return to its initial configuration. The advantage the cleanup signal gives us is that it ensures that the copy gadget is "broken", and hence we need to deliver one additional chip to each half-bit copy gadget to "repair" it after the computation and cleanup waves pass. The only challenge is to somehow generate these required extra chips – we claim that the discarded chips from the computation gadgets suffice. In particular:

Lemma 1.6. *The number of half-bit copy gadgets exactly equals the total number of chips sent to the discard vertex by all vertices in the graph.*

Proof. Note that the number of chips coming into the graph via the input and those leaving the graph via the output is balanced, as the number of inputs and outputs is equal, and we use the same number of chips to encode a 0 or a 1. Further, every internal vertex except the `copy` vertex inside each copy gadget returns to its initial configuration. Since no chips are created during the process, any missing chips must end up at the discard vertex. □

So we add an edge from the `discard` vertex to the `copy` vertex of each copy gadget. We want to ensure that the computation wave has already passed the copy gadget by the time the repair chip arrives, so we add a delay of D steps on this edge, where D is the length of the gadget G corresponding to the whole computation. Finally, we wire up the outputs of G to its inputs, each with a delay of D steps, to allow enough time for the copy gadgets to reset in the background.

Finally, we are ready to claim our theorem, with the following caveat: The chip-firing game on the instance described above is clearly infinite. Even for inputs x where the Turing machine M halts, the game runs forever – to recognise that the machine has halted, one must observe that the output of G_n did not change after simulating a full step of the Turing machine. Alternatively, we note that any s-space Turing machine must halt within $2^{\mathcal{O}(s(n))}$ steps if it halts at all, so we can just let G_n simulate enough rounds of M to ensure that the final state (if it exists) has been computed. The first approach requires an "observer" to compare the output to the input, and has only a polynomial slowdown, whereas the second has no external requirement, but can have an exponential slowdown in the worst case (for example, if M is a trivial machine which immediately accepts all inputs).

Theorem 1.7. *For any Turing machine M that uses at most $s(n)$ cells on its work tape on inputs of size n, there exists a family of graphs $\{G_n\}$ and partial configurations c_n such that:*

- $|G_n| \leq p(s(n))$ *for some polynomial p.*
- *G_n can be partitioned into two graphs H_n and L_n on the same vertex set such that (H_n, c_n) is a gadget which computes a single step of the Turing machine M, and L_n contains paths from each output layer vertex to each input layer vertex, and paths from the universal `discard` vertex to the `copy` vertex of each half-bit copy gadget.*
- *For each input $x \in \{0,1\}^n$, there exists a corresponding input (with encoding as in Figure 1.10 and Figure 1.11) to G_n such that it computes $M(x)$.*

The proof follows from the previous discussion.

1.4 Hardness considerations

In this section we look at what the simulation results imply for the hardness of the chip-firing game. First, we have an undecidable language of chip-firing game instances.

Theorem 1.8. *We define the language* `INF-REACH` *to contain* (G, c, v)*, where* G *is an infinite graph,* c *is an initial configuration of chips on* $V(G)$*, and* v *is a vertex of* G *such that at least one chip reaches* v *during the chip-firing game played on* (G, c) *under any player strategy. Then* `INF-REACH` *is undecidable.*

Before we look at a proof, it is important to talk about an encoding of G and c. With a bad encoding, it may be the case that the difficulty in deciding instances of `INF-REACH` lies in simply decoding the graph. We will restrict ourselves to a *tiling* encoding, where (except a finite number of exceptions) G is specified by repeated finite subgraphs tiled in an infinite grid.

Proof. We will reduce `HALT` to `INF-REACH`. Consider an arbitrary Turing machine M and an input x. First, we modify M by adding an extra cell to the left side of its tape, such M writes 1 to this cell iff it halts (for example, this can be done by adding extra halting states which travel to the left until they reach this cell and then write 1 to it). Then consider the boolean circuit $\mathcal{C}$ which computes the value of a cell given the values of the cells in its neighbourhood in the previous step of the Turing machine. We will tile an infinite grid with copies of $\mathcal{C}$, with the outputs of each layer wired up to the inputs of the previous layer (essentially an infinite analogue of the t-step s-space circuit described in Section 1.3). And finally, we use Theorem 1.3 to convert this circuit into a chip-firing instance (G, c). We introduce an additional vertex v into G which receives a chip iff the leftmost cell of M's tape becomes 1 at any time-step. This is equivalent to adding an edge from each node corresponding to the tape cell in the circuit to the (new) node corresponding to v (and hence equivalent to an edge in G, modulo copy gadgets added for fan-out).

Hence, if the player is restricted to the synchronous strategy, v receives a chip iff M halts on the input x. To get the theorem from here, we need to do two things. First, we want to show that (G, c) can be effectively tiled.

The first layer of G contains the input. We will encode x (the input to the Turing machine) here, using the *full-bit* encoding from Section 1.2. Note that this means we specify $c(\cdot)$ for a finite number of vertices. The remaining (infinitely many) vertices in the input layer can be tiled, since they all represent the binary encoding of an empty cell. We will tile the rest of G (roughly) with tiles corresponding to copies of $\mathcal{C}$. In particular, note that the chip-firing instance corresponding to a single copy of $\mathcal{C}$ almost suffices, except that by stitching multiple

copies together in a grid, we may need additional copy gadgets (since an input bit may be used by two different copies of $\mathcal{C}$). Let us be more precise – suppose a copy of $\mathcal{C}$ computes a tape cell i at time-step t. Suppose further that it uses the value of the tape cells $(i-1)$, i, and $(i+1)$ at time $(t-1)$. Then we will put the copy gadgets required for the fanout of the $(i-1)$-th tape cell in the tile corresponding to the $(i-1)$-th cell, and similarly for the i-th and $(i+1)$-th tape cell. Since the copies of $\mathcal{C}$ to the left and the right will interact with their neighbours in the same manner, placing these copy gadgets in the same manner suffices. The exception is the copy of $\mathcal{C}$ corresponding to the first tape cell, since it does not have a neighbour to the left (and in addition, there is an extra edge going to v). However, we note that all such copies for different time steps are indeed the same, and so can be tiled along that direction. Hence (G, c) can be effectively tiled.

Next, we need to show that (G, c) simulates M on the input x under *any* player strategy. We will sneakily use the fact that if the game is *finite* and ends in configuration c' under some strategy, it does so under any strategy (see [5] for a proof). Note that if we take any prefix of the layers of G, we obtain a D-gadget for some D. Note also that since G is a directed acyclic graph, we can decide if any vertex of this prefix receives a chip simply by examining the (finite) chip-firing instance induced by (G, c) on it. Suppose M halts on x after t steps, then we know that if we consider a prefix of G corresponding to t steps of M, a chip reaches v under the synchronous strategy. Hence a chip reaches v in this prefix under any player strategy, which implies it reaches v under any player strategy on G itself. For the other direction, suppose M does not halt on x. Then for any finite prefix of G, no chip reaches v under the synchronous strategy, which means no chip reaches v under any player strategy on this prefix, and by induction no chip reaches v under any player strategy on G. □

It is tempting to look for a similar result from the Turing machine simulation of Theorem 1.7, for example by simulating an arbitrary PSPACE machine. However, the Theorem 1.7 is reliant on the synchronization strategy in a fundamental manner. In particular, if the player is allowed an arbitrary strategy, they can fire the chips corresponding to the cleanup layer first, effectively changing the input to the machine. Hence we have the following weaker theorem:

Theorem 1.9. *Consider the problem REACH of determining whether some configuration c' is reachable in the chip-firing game played on G with the initial configuration c, under the synchronous strategy. REACH is PSPACE-hard.*

Proof. Suppose L is any language in PSPACE, and M is a PSPACE machine such that $L(M) = L$. Then we modify M so that if it accepts it writes 0 to all but the first cell of its work tape, and 0 everywhere if it rejects. Then for any input

x to M, the graph and initial configuration corresponding to (M, x) from Theorem 1.7 serve as G and c, and c' corresponds to the accepting configuration of M described above. Hence we have a reduction from an instance of L to an instance of REACH. □

Chapter 2

Discrepancy bounds for Propp machine

The Propp machine, or the rotor-router model, is a quasirandom process on graphs, where instead of sending tokens to a random neighbour, each vertex v sends tokens one-by-one to its neighbours in a fixed order. In particular, we have a graph G, and N indistinguishable tokens initially distributed arbitrarily. Associated with each vertex v is a permutation ρ_v of its neighbours, and a rotor which moves through this permutation. At each time-step, each vertex fires all of its tokens, with the i-th token to be fired from vertex v overall going to $\rho_v(i)$, with $i \geq \delta(v)$ wrapping around and mapping to $\rho_v(i \bmod \delta(v))$, where $\delta(v)$ is the out-degree of v.

Consider for any vertex v, the number of tokens $\chi_v^{(T)}$ at v at time-step T. We are interested primarily in bounding the discrepancy of this quantity from $\mu_v^{(T)}$: the expected number of tokens at v after T steps of each token walking randomly. We are interested in the machine's behaviour when N – the number of tokens – is large.

Cooper and Spencer [7] were the first to analyse this discrepancy; they showed for the special case where $G = \mathbb{Z}^d$ (the infinite hypercube) that the discrepancy is at most a constant. Kijima et al. [8] generalised their techniques and showed that on a finite graph G where the associated random walk is lazy and reversible, the discrepancy between the two processes is $\mathcal{O}(mn)$, where m and n denote the number of edges and vertices in G respectively. Our main contribution is to tighten their bound in some special cases – in particular we show that when the transition matrix P corresponding to the random walk on G is diagonalizable, the discrepancy is $\mathcal{O}(m\sqrt{n})$, and if G is d-regular, the discrepancy is $\mathcal{O}(m)$. This matches a lowerbound on discrepancy in d-regular graphs, also shown by [8].

In the first section of this chapter, we present the tools we need from pre-

vious work. In the second section we show the tighter bounds for special cases described above. In the third section we show a new approach to analyse the behaviour of the Propp machine in a "mixed" state (as opposed to at an arbitrary time-step). We prove that after mixing, the Propp machine matches a random walk exactly *on average*.

2.1 Preliminaries

In this section we import the results we need from [8].

We define $s_v(i)$ as the time-step when the vertex v fires its i-th token overall. Note that $s_v(i)$ is often equal to $s_v(i+1)$, since a vertex typically fires many tokens at each time-step. Let P denote the transition matrix of the random walk associated with G.

We want to bound $|\chi_w^T - \mu_w^T|$ for each vertex w of G, and every time-step T. We begin with the following "untelescoping" idea:

Proposition 2.1 ([8])**.**

$$\chi_w^T - \mu_w^T = \sum_{v \in V} \sum_{i=0}^{X_v^{(T-1)}-1} (P^{T-s_v(i)-1}(\rho_v(i), w) - P^{T-s_v(i)}(v, w))$$

Where $X_v^{(T-1)} := \sum_{t=0}^{T-1} \chi_v^{(t)}$, or all the tokens that visit the vertex v in the first $T-1$ steps.

Before we see the proof, we expose why this idea is useful. The discrepancy on the left side is between two (seemingly) very different processes – in the first each token follows some "fixed" path whereas in the second each token walks randomly. However, each term of the sum on the right side corresponds (as we shall see during the proof) to processes which are very similar. In particular, the tokens walk $s_v(i)(+1)$ steps on some fixed path, and the remaining $T - s_v(i)(-1)$ steps randomly.

Proof. Consider an arbitrary token, which starts at some vertex u, and ends at a vertex u', taking the path $u = u_0, u_1, \ldots, u_T = u'$ in the Propp Machine. Note that if $u' = w$, this token contributes 1 to χ_w^T, and otherwise it contributes 0.

What is the probability that this token is at w after T steps in a random walk? It is $P^T(u, w)$. This is also the contribution of this token to μ_w^T. We will untelescope this value along the path $u_0, \ldots u_T$, by imagining a process where the token proceeds along the path for t steps, and then walks randomly for the remaining $T - t$ steps. In particular, we have:

$$P^T(u, w) - P^0(u', w) = \sum_{i=0}^{T-1} (P^{T-i}(u_i, w) - P^{T-i-1}(u_{i+1}, w))$$

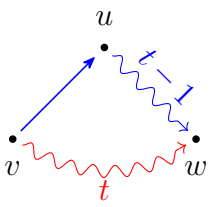

Figure 2.1 $f_{v,w}(u,t)$; the blue path represents a fixed step and a random walk of $t-1$ steps, the red path represents a random walk of t steps.

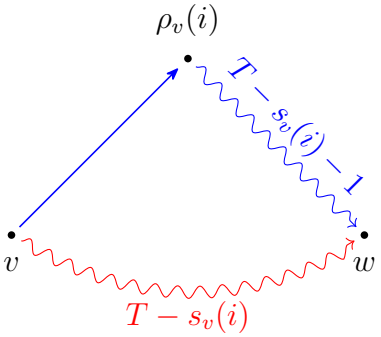

Figure 2.2 $f_{v,w}(\rho_v(i), T - s_v(i))$

Note that P^0 is simply the identity matrix, and hence $P^0(u', w)$ is 1 iff the token is at w in the Propp Machine after T steps. Finally to get the proposition, let $\mathcal{P}$ be the multiset of paths taken by each token in the Propp Machine's first T steps, and observe:

$$\begin{aligned}\mu_w^T - \chi_w^T &= \sum_{(u_0,\dots u_T)\in\mathcal{P}} P^T(u,w) - P^0(u',w) \\ &= \sum_{(u_0,\dots u_T)\in\mathcal{P}} \sum_{i=0}^{T-1} (P^{T-i}(u_i,w) - P^{T-i-1}(u_{i+1},w))\end{aligned}$$

And finally, by first summing over vertices, and then tokens passing through them, we get:

$$\mu_w^{(T)} - \chi_w^{(T)} = \sum_{v\in V} \sum_{i=0}^{X_v^{(T-1)}-1} (P^{T-s_v(i)}(v,w) - P^{T-s_v(i)-1}(\rho_v(i),w))$$

□

We now define a "single-step discrepancy", for v, w any pair of vertices, u a neighbour of v, and $t > 0$ (see Figure 2.1).

$$f_{v,w}(u,t) = P^{t-1}(u,w) - P^t(v,w)$$

This is the discrepancy between a random walk for t steps from v to w, and a walk with a fixed first step from v to u, and $(t-1)$ random steps. Note that

each term inside the sum from Proposition 2.1 is equal to $f_{v,w}(\rho_v(i), T - s_v(i))$ (see Figure 2.2).

If we take an average over the choice of this fixed first step, then by definition we are walking randomly, and hence we get:

$$\begin{aligned}\sum_{u\in N(v)} (P^{t-1}(u,w) - P^t(v,w)) &= |N(v)| \cdot \left(\sum_{u\in N(v)} \frac{P^{t-1}(u,w)}{|N(v)|} - P^t(v,w) \right) \\ &= |N(v)| \cdot (P^t(v,w) - P^t(v,w)) \\ &= 0\end{aligned}$$

Rewriting with the notation $f_{v,w}$ we have:

Proposition 2.2 ([8])**.**

$$\sum_{u\in N(v)} f_{v,w}(u,t) = 0$$

And the straightforward corollary:

Corollary 2.3. For any permutation ρ_v of $N(v)$

$$f_{v,w}(\rho_v(0), t) = -\sum_{r=1}^{\delta(v)-1} f_{v,w}(\rho_v(r), t)$$

Where $\delta(v)$ is the out-degree of the vertex v.

We also define:

$$g_w(v) = \sum_{i=0}^{X_v^{(T-1)}-1} f_{v,w}(\rho_v(i), T - s_v(i))$$

Intuitively, $g_w(v)$ is the discrepancy between $\chi_w^{(T)}$ and $\mu_w^{(T)}$ contributed by tokens as they pass through the vertex v. Using Proposition 2.1, and the definition of $f_{v,w}$ and $g_w(v)$ we have:

$$\mu_w^{(T)} - \chi_w^{(T)} = \sum_{v\in V} \sum_{i=0}^{X_v^{(T-1)}-1} f_{v,w}(\rho_v(i), T - s_v(i)) = \sum_{v\in V} g_w(v)$$

The following lemma is our main tool in bounding $g_w(v)$ (and hence the discrepancy):

Lemma 2.4 ([8])**.** *For any pair of vertices v and w, there exist $\delta(v) - 1$ non-decreasing sequences $\{z_j^{(1)}\}, \{z_j^{(2)}\}, \ldots, \{z_j^{(\delta(v)-1)}\}$ and a constant $c_{v,w}$ in $[-\delta(v), \delta(v)]$ such that*

$$g_w(v) = \sum_{r=1}^{\delta(v)-1} \sum_{j=0}^{\tau_r - 1} \left((-1)^j P^{z_j^{(r)}-1}(\rho_v(r), w) + (-1)^{j+1} P^{z_j^{(r)}}(v,w) \right) + c_{v,w}$$

Where ρ_v is the permutation of its neighbours followed by the rotor-router at v, and τ_r is the length of the sequence $z^{(r)}$.

While the statement might be hard to digest, the important takeaway is to see that we have $2\delta(v)$ alternating "geometric" sums – in the sense that each subsequent term is the same entry of a higher power of the transition matrix P.

Proof.

$$g_w(v) = \sum_{i=0}^{X^{(T-1)}-1} f_{v,w}(\rho_v(i), T - s_v(i))$$

Let K_r be one less than[1] the number of tokens fired from v to $\rho_v(r)$ over time $[0, T)$, then by collecting the tokens sent to $\rho_v(r)$ together, we have:

$$g_w(v) = \sum_{r=0}^{\delta(v)-1} \sum_{h=0}^{K_r} f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot h + r))$$

Next, we separate out the tokens sent to $\rho_v(0)$, and apply Corollary 2.3 to them. This step is where the alternating sum in the statement of the lemma comes from.

$$\begin{aligned} g_w(v) &= \sum_{r=1}^{\delta(v)-1} \sum_{h=0}^{K_r} f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot h + r)) + \sum_{h=0}^{K_0} f_{v,w}(\rho_v(0), T - s_v(\delta(v) \cdot h)) \\ &= \sum_{r=1}^{\delta(v)-1} \sum_{h=0}^{K_r} f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot h + r)) - \sum_{r=1}^{\delta(v)-1} \sum_{h=0}^{K_0} f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot h)) \end{aligned}$$

We bring $K_r + 1$ of these pseudo-tokens back into the first sum, leaving $K_0 - K_r \leq 1$ "error terms".

$$\begin{aligned} g_w(v) = &\sum_{r=1}^{\delta(v)-1} \sum_{h=0}^{K_r} \Big(f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot h + r)) - f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot h)) \Big) \\ &- \sum_{r=1}^{\delta(v)-1} (K_0 - K_r) f_{v,w}(\rho_v(r), T - s_v(\delta(v) \cdot K_0)) \end{aligned}$$

At this point, we can already see a decreasing sequence emerging in the second input to $f_{v,w}$; in particular for any r, the sequence $t^{(r)}$ defined as

$$\begin{aligned} t^{(r)}_{2h} &= T - s_v(\delta(v) \cdot h) \\ t^{(r)}_{2h+1} &= T - s_v(\delta(v) \cdot h + r) \end{aligned}$$

[1]For convenience of notation in sum-indices.

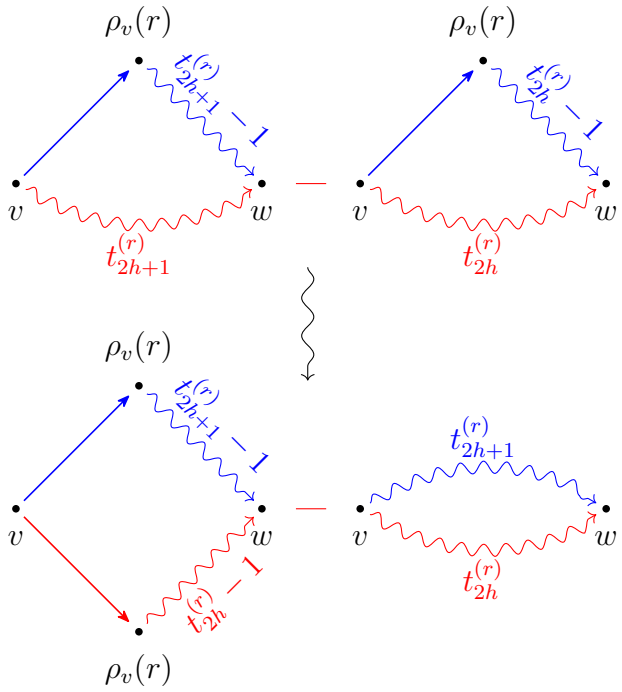

Figure 2.3 The main transformation in the proof of Lemma 2.4.

is non-increasing. Note also that $f_{v,w}$ is bounded between -1 and 1, so the error terms add up to some number $c_{v,w} \in [-\delta(v), \delta(v)]$ Plugging all of this back in, we have:

$$g_w(v) = \sum_{r=1}^{\delta(v)-1} \sum_{h=0}^{K_r} \left(f_{v,w}(\rho_v(r), t_{2h+1}^{(r)})) - f_{v,w}(\rho_v(r), t_{2h}^{(r)})) \right) + c_{v,w}$$

We will rearrange the term inside the sum to obtain the statement of the lemma; in particular, let

$$\Delta = f_{v,w}(u,t) - f_{v,w}(u,t')$$

By definition of $f_{v,w}$, we have:

$$\Delta = P^{t-1}(u,w) - P^t(v,w) - (P^{t'-1}(u,w) - P^{t'}(v,w))$$

Rearranging:

$$\Delta = P^{t-1}(u,w) - P^{t'-1}(u,w) - (P^t(v,w) - P^{t'}(v,w))$$

Applying this transformation to the summand in the sum for $g_w(v)$ above, we have (see Figure 2.3 for an illustration of what is happening):

$$g_w(v) = \sum_{r=1}^{\delta(v)-1} \sum_{h=0}^{K_r} \Bigg(P^{t^{(r)}_{2h+1}-1}(\rho_v(r), w) - P^{t^{(r)}_{2h}-1}(\rho_v(r), w) \\ - (P^{t^{(r)}_{2h+1}}(v, w) - P^{t^{(r)}_{2h}}(v, w)) \Bigg) + c_{v,w}$$

And finally, we have two alternating sums as promised:

$$g_w(v) = \sum_{r=1}^{\delta(v)-1} \sum_{j=0}^{2K_r+1} \left((-1)^{j+1} P^{t^{(r)}_j - 1}(\rho_v(r), w) + (-1)^j P^{t^{(r)}_j}(v, w) \right) + c_{v,w}$$

To get the statement of the lemma, we define $z^{(r)}_j = t^{(r)}_{2K_r+1-j}$ to get a non-decreasing sequence, and rewrite the sum above:

$$g_w(v) = \sum_{r=1}^{\delta(v)-1} \sum_{j=0}^{2K_r+1} \left((-1)^{j} P^{z^{(r)}_j - 1}(\rho_v(r), w) + (-1)^{j+1} P^{z^{(r)}_j}(v, w) \right) + c_{v,w}$$

□

We now shift our focus to bounding these alternating series. In particular we have the following lemma:

Lemma 2.5 ([8]). *Let G be a graph with transition matrix P such that P is diagonalizable, and the eigenvalues of P are $\lambda_1, \lambda_2, \ldots, \lambda_n$. Let $\vec{b_i}$ be an eigenvector of norm 1 corresponding to λ_i such that $B = \{\vec{b_1}, \ldots, \vec{b_n}\}$ is a basis of $\mathbb{F}^n$. For $\{z_j\}$ a non-decreasing sequence of length τ and any pair of vertices v, w of G:*

$$\sum_{j=0}^{\tau-1} (-1)^j P^{z_j}(v, w) \leq \sum_{i=1}^{n} a_i b_{i,v} \sum_{j=0}^{\tau-1} (-1)^j \lambda_i^{z_j}$$

Where $\vec{a}$ is the coordinate vector of $\vec{e_w}$ in the basis B.

Proof. Since P is diagonalizable, we can express $\vec{e_w}$ (the standard basis vector which is 1 at coordinate w and 0 everywhere else) as a linear combination of the vectors in B. In particular, suppose $\vec{e_w} = \sum_{i=1}^{n} a_i \vec{b_i}$. Then note that

$$\|\vec{e_w}\| = \sum_{i=1}^{n} a_i^2 \left\|\vec{b_i}\right\| \\ = \sum_{i=1}^{n} a_i^2 = 1$$

Now for any integer x, we can obtain the (v, w)-th entry of P^x:

$$\begin{aligned} P^x(v,w) &= \vec{e_v}^T P^x \vec{e_w} \\ &= \vec{e_v}^T P^x \left(\sum_{i=1}^{n} a_i \vec{b_i} \right) \end{aligned}$$

Moving scalars to the front, and repeatedly using the fact that b_i is an eigenvector of P corresponding to eigenvalue λ_i:

$$P^x(v,w) = \sum_{i=1}^{n} a_i \lambda_i^z (\vec{e_v} \cdot \vec{b_i})$$

$\vec{e_v}$ just extracts the v-th coordinate of $\vec{b_i}$, and we have:

$$P^x(v,w) = \sum_{i=1}^{n} a_i \lambda_i^z b_{i,v}$$

Finally:

$$\sum_{j=0}^{\tau-1} (-1)^j P^{z_i}(v,w) = \sum_{j=0}^{\tau-1} (-1)^j \sum_{i=1}^{n} a_i \lambda_i^{z_j} b_{i,v}$$

And the lemma follows by reordering the sums. □

2.2 Tighter bounds for special cases

This is finally where we diverge from previous work – while Kijima et al. generalised the previous lemma to graphs where the transition matrix P is not diagonalizable, we will look for tighter bounds in the special case where it is. First, we need the following elementary proposition about alternating geometric series.

Proposition 2.6. *For $1 \geq \lambda > -1 + c$ for some positive constant c, and any non-decreasing sequence of positive integers $\{z_j\}$ of length τ*

$$\left| \sum_{j=0}^{\tau-1} (-1)^j \lambda^j \right| \leq \frac{1}{1-c}$$

Proof. For $\lambda \geq 0$, the series alternates, so

$$\left| \sum_{j=0}^{\tau-1} (-1)^j \lambda^j \right| \leq \lambda^{z_0} \leq \lambda \leq 1$$

For $\lambda < 0$, the signs of the terms λ^{z_j} and $(-1)^j$ can match up, so we have:

$$\left|\sum_{j=0}^{\tau-1}(-1)^j\lambda^j\right| \leq \sum_{j=0}^{\tau-1}|\lambda^{z_j}| \leq \sum_{j\geq 0}|\lambda^j| = \frac{1}{1-|\lambda|} \leq \frac{1}{1-c}$$

□

Combining Lemma 2.5 and Proposition 2.6, we have:

Lemma 2.7. *With the same notation and conditions as in Lemma 2.5, and in addition the smallest eigenvalue λ_n of P bounded below by $(-1+c)$:*

$$\left|\sum_{j=0}^{\tau-1}(-1)^j P^{z_j}(v,w)\right| \leq \frac{1}{1-c}\sum_{i=1}^{n}|a_i b_{i,v}|$$

Proof. Immediate from bounding the inner-sum in Lemma 2.5 using Proposition 2.6. □

Corollary 2.8. Under the conditions of Lemma 2.7:

$$\left|\sum_{j=0}^{\tau-1}(-1)^j P^{z_j}(v,w)\right| \leq \frac{\sqrt{n}}{1-c}$$

Proof. From Lemma 2.7,

$$\begin{aligned}\left|\sum_{j=0}^{\tau-1}(-1)^j P^{z_j}(v,w)\right| &\leq \frac{1}{1-c}\sum_{i=1}^{n}|a_i b_{i,v}| \\ &= \frac{1}{1-c}\sum_{i=1}^{n}|a_i||b_{i,v}| \\ &\leq \frac{1}{1-c}\sum_{i=1}^{n}\|a_i\|\left\|\vec{b_i}\right\| \\ &= \frac{\sqrt{n}}{1-c}\end{aligned}$$

Where the second last step is the Cauchy-Schwarz inequality applied to the vectors $(|a_1|,\ldots,|a_n|)$, and $(|b_{1,v}|,|b_{2,v}|,\ldots,|b_{n,v}|)$. Each entry of $\vec{b_i}$ is bounded by 1, and hence we can bound its L2 norm by $\sqrt{n}$. □

Theorem 2.9. *For a graph G with the corresponding random-walk transition matrix P diagonalizable, and the smallest eigenvalue of P bounded below by $(-1+c)$ for some constant c, $|g_w(v)| = \mathcal{O}(\delta(v)\sqrt{n})$ for every pair of vertices w, v. Hence, the discrepancy $|\chi_w^{(T)} - \mu_w^{(T)}|$ is bounded by $\mathcal{O}(m\sqrt{n})$, where the constant in the $\mathcal{O}$ depends on c.*

Proof. We recall Lemma 2.4:

$$g_w(v) = \sum_{r=1}^{\delta(v)-1} \sum_{j=0}^{\tau_r - 1} \Big((-1)^j P^{z_j^{(r)}-1}(\rho_v(r), w) + (-1)^{j+1} P^{z_j^{(r)}}(v, w) \Big) + c_{v,w}$$

Where for each r, $\{z_j^{(r)}\}$ is a non-decreasing sequence, and $c_{v,w}$ is some constant between $[-\delta(v), \delta(v)]$.

Then the absolute value of each of the two series inside the sum can be bounded using Corollary 2.8, to give us:

$$|g_w(v)| \leq \sum_{r=1}^{\delta(v)-1} \left(\frac{2\sqrt{n}}{1-c} \right) + |c_{v,w}|$$

And now recalling the bounds on $c_{v,w}$, we have:

$$|g_w(v)| = \mathcal{O}\Big(\delta(v)\sqrt{n}\Big) + \delta(v) = \mathcal{O}\Big(\delta(v)\sqrt{n}\Big)$$

□

Now, we take a moment to recall the main condition of Lemma 2.5 (and hence Lemma 2.7, and Theorem 2.9); P (the transition matrix corresponding to a random walk walk on some graph G) must be diagonalizable. Note that this fact is important in the proof, since we express the elementary basis vector $\vec{e_w}$ as a linear combination of the eigenvectors of P.

For an undirected graph G, its adjacency matrix $A(G)$ is symmetric, and hence diagonalizable. However, the corresponding transition matrix P need not be diagonalizable.

We consider an important special case where P is indeed diagonlizable – when G is d-regular; in particular we will show that the discrepancy is only $\mathcal{O}(m)$ in this case.

Theorem 2.10. *When G is d-regular, and the smallest eigenvalue of its transition matrix P is at least $(-1 + c)$ for some constant c, the discrepancy $|\chi_w^{(T)} - \mu_w^{(T)}|$ is $\mathcal{O}(m)$, where the constant in the $\mathcal{O}$ depends on c.*

Proof. When G is d-regular, the transition matrix P of the random walk on it is just A/d, where A is the adjacency matrix. Since A is symmetric, P is also symmetric, and hence diagonalizable.

We will use Lemma 2.7 to bound the two series in Lemma 2.4; in particular, recall that a_i is the i-th coordinate of $\vec{e_w}$ in the orthonormal basis of eigenvectors

of P, and $b_{i,v}$ is the v-th coordinate of the i-th eigenvector of this basis. As before, we start with:

$$g_w(v) = \sum_{r=1}^{d-1} \sum_{j=0}^{\tau_r - 1} \left((-1)^j P^{z_j^{(r)}-1}(\rho_v(r), w) + (-1)^{j+1} P^{z_j^{(r)}}(v, w) \right) + c_{v,w}$$

Then, bounding the absolute value of the two series inside with Lemma 2.7, and applying the triangle inequality on $|g_w(v)|$ we have:

$$|g_w(v)| \leq \sum_{r=1}^{d-1} \frac{1}{1-c} \left(\sum_{i=1}^{n} |a_i b_{i,\rho_v(r)}| + \sum_{i=1}^{n} |a_i b_{i,v}| \right) + |c_{v,w}|$$

The plan is to consider $|\chi_w^{(T)} - \mu_w^{(T)}|$ as a whole (instead of just bounding $g_w(v)$). Note that the $\sum_{v \in V} |c_{v,w}| = \mathcal{O}(m)$, so by applying the triangle inequality to $|\chi_w^{(T)} - \mu_w^{(T)}| = |\sum_{v \in V} g_w(v)|$, we have:

$$|\chi_w^{(T)} - \mu_w^{(T)}| \leq \sum_{v \in V} \sum_{r=1}^{d-1} \frac{1}{1-c} \left(\sum_{i=1}^{n} |a_i b_{i,\rho_v(r)}| + \sum_{i=1}^{n} |a_i b_{i,v}| \right) + \mathcal{O}(m)$$

Reordering the sums, we have:

$$\begin{aligned} |\chi_w^{(T)} - \mu_w^{(T)}| &\leq \frac{1}{1-c} \sum_{i=1}^{n} \sum_{r=1}^{d-1} \sum_{v \in V} (|a_i b_{i,\rho_v(r)}| + |a_i b_{i,v}|) + \mathcal{O}(m) \\ &= \frac{1}{1-c} \sum_{i=1}^{n} |a_i| \sum_{v \in V} \sum_{r=1}^{d-1} (|b_{i,v}| + |b_{i,\rho_v(r),}|) + \mathcal{O}(m) \end{aligned}$$

Now, for each vertex u, we count the number of occurences of $|b_{i,u}|$ in the sum above; the first term clearly contributes $d-1$ occurences, and the second term contributes at most once per edge going into u, and hence d occurences in total. Hence the discrepancy is bounded by:

$$\begin{aligned} |\chi_w^{(T)} - \mu_w^{(T)}| &\leq \frac{1}{1-c} \sum_{i=1}^{n} |a_i| \sum_{v \in V} 2d |b_{i,v}| + \mathcal{O}(m) \\ &= \frac{2d}{1-c} \sum_{i=1}^{n} |a_i| \sum_{v \in V} |b_{i,v}| + \mathcal{O}(m) \end{aligned}$$

And finally we note that $\vec{b_i}$ and $\vec{a}$ are unit vectors, and hence their L1 norms are bounded by $\sqrt{n}$ (using Cauchy-Schwarz), to give us:

$$|\chi_w^{(T)} - \mu_w^{(T)}| \leq \frac{2dn}{1-c} + \mathcal{O}(m) = \mathcal{O}(m)$$

□

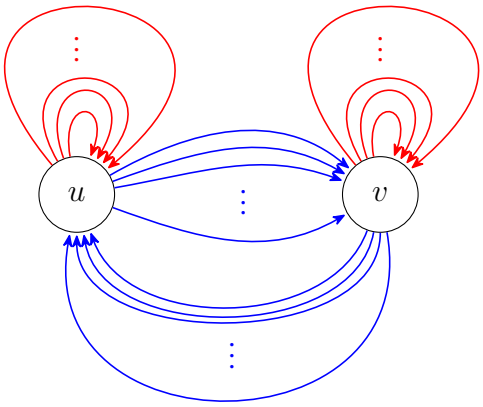

Figure 2.4 An example with $\Omega(m)$ discrepancy. The red edges are before the blue edges in the permutations ρ_u and ρ_v.

This matches (in some sense) a lowerbound from [8]. Consider the graph in Figure 2.4, with k tokens on u initially, and k self-loops and k edges going to the other vertex. The random walk mixes instantly, while in the Propp machine the tokens stay together forever, which gives a discrepancy of $k/2$ at any time-step. Note that this still leaves room for a better upper-bound (for example, it does not rule out a bound of $\mathcal{O}(\delta(v))$ on the discrepancy).

2.3 Towards combinatorial proofs

One of our motivations while studying the Propp machine was to find more intuitive or natural proofs for the discrepancy bounds in the previous section. While we failed to do so, we found some intermediate results of independent interest, which are documented in this section.

Let us define a *configuration* of the Propp machine on a graph G as a pair (c, z) where $c(v)$ is the number of tokens at the vertex v, and $z(v)$ is the index in the permutation ρ_v where at which the rotor is pointing. Then we show the following Theorem:

Theorem 2.11. *After a finite "mixing" time, the Propp machine on an undirected graph G with any starting configuration cycles between a finite number of configurations – the average of these configurations is equal to the stationary distribution of a random walk on G (multiplied by the number of tokens N).*

Proof. Note that the number of distinct configurations is finite, and the behaviour of the machine is determined entirely by its current configuration. Hence the machine reaches a cycle of configurations.

Let π_v denote the stationary distribution of the random walk on G. Let $\mathcal{C} = \{(c_0, z_0), \ldots, (c_k, z_k)\}$ denote the configurations in the cycle. Now consider the set S of vertices v that has more tokens than $N \cdot \pi_v$ on average over $\mathcal{C}$. If $S = \emptyset$, we are done, so assume S is non-empty. Further, assume that the subgraph induced by S on G is connected (if not, we can work with a maximal such subset of S).

Now, consider the edges between S and $V \setminus S$; since each vertex u of S has on average strictly more than $N \cdot \pi_u$ tokens, and each neighbour v of u receives an equal number of tokens over $\mathcal{C}$ (because the routers reset to the positions z_0), strictly more than $|\mathcal{C}| \cdot N \cdot \pi_u / d_u$ tokens leave S via the edge $\{u, v\}$ for $v \notin S$. By the same argument, fewer than $|\mathcal{C}| \cdot N \cdot \pi_v / \deg_v$ tokens enter S via the same edge.

Recall that for undirected graphs, $\pi_u \sim \deg_u$, and hence S leaks tokens over the cycle of configurations $\mathcal{C}$ and we have a contradiction. □

Next, we can show that the maximum positive discrepancy cannot increase once it falls below certain thresholds.

Theorem 2.12. *Suppose that the number of tokens N is divisible by $2m$. For some configuration (c, z) define for each vertex v the "bucket" $b(v)$ of discrepancy it falls into as the unique integer z such that $c(v) - N \cdot \pi_v \in (z \cdot \deg_v, (z+1) \cdot \deg_v]$. Then $\max_v b(v)$ is non-increasing with the operation of the Propp machine.*

Proof. Each vertex u can supply at most $b(u) + 1$ additional tokens over the $N \cdot \pi_u / \deg_u$ "expected" tokens to each of its neighbours. But this means each vertex v receives at most $\deg_v \cdot (b(v) + 1)$ extra tokens over its expected $N \cdot \pi_v$ tokens, and hence has discrepancy bracket at most $b(v)$. □

Bibliography

[1] J Spencer. "Balancing Vectors in the Max Norm". In: *Combinatorica* 6.1 (Jan. 1986), 55–65. ISSN: 0209-9683.

[2] Anders Björner, László Lovász, and Peter W Shor. "Chip-firing games on graphs". In: *European Journal of Combinatorics* 12.4 (1991), pp. 283–291.

[3] Vyatcheslav B Priezzhev et al. "Eulerian walkers as a model of self-organized criticality". In: *Physical Review Letters* 77.25 (1996), p. 5079.

[4] Eric Goles and Maurice Margenstern. "Universality of the chip-firing game". In: *Theoretical Computer Science* 172.1 (1997), pp. 121–134. ISSN: 0304-3975.

[5] Anders Björner and László Lovász. "Chip-firing games on directed graphs". In: *Journal of algebraic combinatorics* 1.4 (1992), pp. 305–328.

[6] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997. ISBN: 978-0-534-94728-6.

[7] Joshua N. Cooper and Joel Spencer. "Simulating a Random Walk with Constant Error". In: *Combinatorics, Probability and Computing* 15.6 (2006), 815–822.

[8] Shuji Kijima, Kentaro Koga, and Kazuhisa Makino. "Deterministic random walks on finite graphs". In: *Random Structures & Algorithms* 46.4 (2015), pp. 739–761.