Influence Maximization Over a Network: Static and Dynamic Policies

by

Zied Ben Chaouch

Joint B.S. in Mathematics and Physics
McGill University, Canada (2014)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
August 31, 2016

Signature redacted

Certified by .....
John N. Tsitsiklis
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Influence Maximization Over a Network: Static and Dynamic Policies

by

Zied Ben Chaouch

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

The problem of maximizing the spread of an opinion inside a social network has been investigated extensively during the past decade. The importance of this problem in applications such as marketing has been amplified by the major expansion of online social networks. In this thesis, we study opinion control policies, first under a broad class of deterministic dynamics governing the interactions inside a network, and then under the classical "Voter Model". In the former case, we design a policy that a controller can follow in order to spread an opinion inside a network with the smallest possible cost. In the latter case, we consider networks whose underlying graph is the $d-$dimensional integer torus $\mathbb{Z}_n^d$, and we design policies that minimize the expected time until the network reaches a consensus. We also show that, in dimension $d \geq 2$, dynamic policies do not perform significantly better than static policies, while, in dimension $d = 1$, optimal dynamic policies perform much better than optimal static policies.

Thesis Supervisor: John N. Tsitsiklis
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to first thank my advisor, Professor John Tsitsiklis, for his guidance and support at MIT. It has been a true privilege to meet weekly with Professor Tsitsiklis and to be a Teaching Assistant for his class. His unique style in teaching and in research have been a constant source of inspiration over the last two years... $\epsilon\upsilon\chi\alpha\rho\iota\sigma\tau\omega$!

I would also like to thanks all my friends at MIT for making my graduate life very enjoyable, especially when trying out some great Bostonian restaurants (I know two outstanding foodies who will probably recognize themselves here!), going to the theatre (Igor and Mario?) or attending a concert at the BSO. I also thank them for all the interesting conversations we often have about math, engineering, life, fiction,...

Finally, I want to thank my family for their constant love and support. First of all, my parents Sonia and Sami: thank you for always being here for me, for being a source of inspiration and encouragement, and for bringing so much joy in my life. I also want to thank my grandparents Ferida, Maherzia and Boubaker, whom I hope to see soon in Boston!

This research was carried out at the Laboratory for Information and Decision Systems, and was mainly supported by an MIT Jacobs Presidential Fellowship.

Festinare nocet, nocet et cunctatio saepe;
tempore quaeque suo qui facit, ille sapit.

— *Ovid*

Contents

List of Figures

List of Tables

s

Chapter 1

Introduction

One of the core problems in the Opinion Dynamics literature is to maximize the spread of an opinion through a network. In this thesis, we study opinion control policies in two different scenarios. In the first part of the thesis, we consider a broad class of deterministic dynamics governing the interactions inside a network, and we design a policy that a controller can follow in order to spread an opinion inside a network with the smallest possible cost. In the second part of the thesis, we focus on the classical "Voter Model" (over networks whose underlying graph is the $d-$dimensional integer torus $\mathbb{Z}_n^d$), and we design policies that minimize the expected time until the network reaches a consensus.

1.1 Main Questions

In the networks we consider, each node has an opinion which can be either 0 or +1. A controller is interested in driving the state of the network to a $1-$consensus (i.e., all nodes should have the +1 opinion by the end of the experiment).

In the first part of the thesis, we consider networks with deterministic dynamics, under which the opinion of a node is determined by the opinion of a single given neighbor at the previous time (unless there is an intervention from the controller). The controller can either do nothing, or influence directly some nodes in the network at a certain cost. The goal of the controller is to drive the network to a $1-$consensus by some time T (T can be infinite) at the smallest possible cost. We also generalize the problem: in this case, the controller wants

to have a fraction $\eta \in (0,1]$ nodes with the $+1$ opinion by some time T. To answer these questions, we develop what we call the *Descendant Algorithm* and the *Ancestral Algorithm.*

In the second part of the thesis, we focus on the Voter Model over networks whose underlying graph is the $d-$dimensional integer torus $\mathbb{Z}_n^d$. We control the network by placing B "stubborn nodes" (i.e., nodes whose opinion is fixed at $+1$) over the torus. We first consider the case where we place the stubborn nodes at the start of the experiment and let the system evolve spontaneously from there (this is the "static case"), and then we consider the case where we are allowed to change the position of our stubborn nodes over the torus during the run of the experiment (this is the "dynamic case"). Our goal here is to determine, for both the static and the dynamic cases, the expected time it takes the network to reach a $1-$consensus when the network is initialized with all nodes in the 0 state, for optimal policies. We conclude this part of the thesis by showing that, in dimension $d \geq 2$, dynamic policies do not perform significantly better than static policies. However, in dimension $d = 1$, optimal dynamic policies perform much better than optimal static policies.

1.2 Special Notation

Throughout the thesis, we will often use notation describing the limiting behavior of a function when the argument tends to infinity. We clarify this notation in this section.

Consider non-negative functions f and g of some argument n:

1. We write $f(n) = O\left(g(n)\right)$ when there exists some positive constant k such that $f(n) \leq kg(n)$ for all large enough n.
2. We write $f(n) = \Omega\left(g(n)\right)$ when there exists some positive constant k such that $f(n) \geq kg(n)$ for all large enough n.
3. We write $f(n) = o\left(g(n)\right)$ when $\limsup_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.
4. We write $f(n) = \Theta\left(g(n)\right)$ when there exists some positive constants k_1 and k_2 such that $k_1 g(n) \leq f(n) \leq k_2 g(n)$ for all large enough n.

Furthermore, given the non-negative functions f, g_1 and g_2, we will use the shorthand notation $g_1(n) \lesssim f(n) \lesssim g_2(n)$ to mean that $f(n) = \Omega\left(g_1(n)\right)$ and $f(n) = O\left(g_2(n)\right)$.

In this thesis, the functions f we will use often depend on multiple arguments: in that case, we will always consider the limiting behavior of f with respect to the number of nodes (n in Part I, and N in Part II of the thesis) in the network, while keeping all other parameters constant.

Finally, given two real numbers a and b, we will sometimes use the shorthand notation $a \vee b$ for $\max(a, b)$.

1.3 Outline of the Thesis

The thesis is divided into two parts:

The first part of the thesis, introduced in Chapter 2, is organized as follows. In Chapter 3 we introduce the formalism used in this first part of the thesis, and define the general time-dependent model we will be using to describe the network (the Dynamic Deterministic System – or DDS). Then, in Chapter 4 we restrict our analysis to a time-invariant model (the Static Deterministic System – or SDS), and we develop what we call the Descendant Algorithm for finite and infinite time-horizon problems by analyzing the structure of the underlying graph of the SDS. Finally, in Chapter 5, we study the DDS and develop what we call the Ancestral Algorithm. A brief comparison between the Descendant and the Ancestral Algorithm will conclude that chapter. For both the SDS and the DDS, we will consider both finite and infinite time-horizon problems. Although most of the algorithms are developed for the case where we want to influence all the nodes in the network, we will provide a dynamic program which solves the case in which we only need to influence a fraction of the nodes in the network.

The second part of the thesis, introduced in Chapter 6, is organized as follows. In Chapter 7 we provide some background on the Voter Model, and introduce the "dual Voter Model" approach, a classical tool often used to compute consensus times. Then, in Chapter 8 we restrict our analysis to the Voter Model over a $d-$dimensional integer torus, and we construct a static placement of stubborn nodes that minimizes the expected time needed to set all nodes in the state 1. Finally, in Chapter 9, we study the same model but allow ourselves to move our set of stubborn nodes during the experiment. In this last chapter, we

want to understand whether there exist dynamic policies providing strictly smaller consensus times than optimal static policies: we will actually show that dynamic policies do not perform significantly better than static policies when $d \geq 2$.

Part I

A Deterministic Model

Chapter 2

Introduction

The problem of maximizing the spread of an opinion inside a social network has been investigated extensively during the past decade. The importance of this problem in applications such as marketing has been amplified by the major expansion of online social networks.

The first part of this thesis focuses on opinion control policies, under a broad class of deterministic dynamics governing the interactions inside a network. We start with a network in which each node has an opinion (which we denote 0 or 1 for simplicity). These nodes can influence each other according to some dynamics. An external controller wants to influence the nodes in order to get a desired fraction of 0 and 1 opinions among the population, at the end of the experiment. Each time the controller decides to influence a node, it will incur a certain cost. Our goal is to design a policy that the external controller can follow in order to attain its goal with the smallest possible cost.

2.1 Scope and Related Work

The problem studied in the first part of the thesis has been inspired by various works in the Opinion Dynamics literature. For example, Domingos and Richardson [5] suggested to take advantage of the internal dynamics of the network to propagate an opinion through a network. In their model, the external controller incurs no cost when nodes spontaneously influence each other. It is therefore crucial to exploit the structure of the network efficiently and look at the impact of each node on the whole network. By modeling the social network as

a Markov random field, they proposed an approximate procedure to determine which nodes should be influenced. In a subsequent paper [13] they simplified their model of interactions between nodes and focused on a question of the form *"how much should I market to this node?"*.

Inspired by [5,13], Kempe, Kleinberg and Tardos [9] provided more efficient approximate algorithms (as the exact solution is NP-hard to compute) to the problem of finding the most influential nodes in a network under the Linear Threshold (nodes take the weighted average opinion of their neighbors and compare it to a threshold) or the Independent Cascade (nodes can influence their neighbors only once) models of diffusion. In a subsequent paper [10], the authors considered a Decreasing Cascade model of diffusion and provided a greedy approximate algorithm for selecting a set of k nodes which maximizes the expected spread of the diffusion.

From a different perspective, Bharathi, Kempe and Salek [2] introduced a game in which each of multiple competitors selects a set of nodes to maximize the expected spread of their own opinion over the network. Although the diffusion model used is probabilistic, once a node adopts an opinion, it keeps it indefinitely.

The works mentioned above focus only on choosing a good initial set of nodes to influence to maximize the spread at the end of the experiment. This corresponds to a static policy. Our goal is to also develop dynamic policies: we are able to influence nodes at each time step, therefore we want to decide who should we influence and at what time, in order to reach a desired proportion of 0 and 1 opinions in the network at the end of the experiment, at minimal cost. The policies we will provide will be of the form "at this time t, influence the following set of nodes".

Another major difference with earlier work lies in the diffusion model used to describe the interaction between the nodes. While the papers mentioned above consider stochastic dynamics such as the Linear Threshold or the Independent Cascade models of diffusion, in this part of the thesis we will only consider deterministic dynamics. Furthermore, we assign possibly different costs to each node the controller can influence directly. Thus, the policy will be node-specific and the algorithm will depend heavily on the structure of the network. Finally, in our model, we allow a node to change its opinion while the experiment

is running, whereas in most previous studies, once a node adopts an opinion it will keep it. Kempe, Kleinberg and Tardos [9] showed in their Theorem 5.1 how to overcome such a limitation, for the model they consider, by creating as many copies of the network as there are time steps in the experiment.

2.2 Contributions

The main contributions of this part of the thesis are as follows. We focus on networks whose graph can evolve with time in a deterministic fashion, and we assume that the opinion dynamics over this network follow some specific deterministic rules. In particular, we focus only on networks composed of nodes of in-degree equal to one (each node can be influenced by exactly one node at each time, also allowing for self-influences). We formulate our problem as a mathematical program in order to find dynamic policies the controller can follow to influence the network at the least possible cost.

For the case where the graph of the network does not change with time, we will produce an algorithm which determines an optimal set of nodes to influence at each time step. We will call it the Descendant Algorithm. We will also focus on the case where the network is a line: when the costs of the nodes are initialized randomly from some discrete distribution, we will analyze the behavior of the optimal cost required to influence the whole network as a function of the time-horizon T of the experiment. We will show that, for small values of T, the optimal cost decreases sharply as a function of T, and that for large values of T the optimal cost decreases linearly as a function of T. We show that this linear behavior tends to occur for times T greater than $O(\log(n))$ when n is large, where n is the number of nodes. Conversely, this tells us that, for large values of n, if we are willing to pay a cost of $O(n)$ we will be able to influence the whole line in time $O(\log(n))$, while if we wish to incur a cost of $o(n)$ we will only be able to influence the whole line in time $O(n)$.

Our last contribution concerns time-evolving networks. In this case, we allow the underlying graph to change with time, i.e., we allow the influencer of a node to change with time. As long as this evolution is deterministic and is known to the controller at the start of the experiment, the algorithm we provide will determine an optimal set of nodes to in-

fluence at each time step. We will call it the Ancestral Algorithm. A small variation of this algorithm will also allow us to solve cases for which we do not aim to influence all nodes in the network, but aim instead to arrive at a given fraction of 0 and 1 opinions.

The algorithms we provide run in polynomial time. For example, if the goal is to drive the network to a consensus of $+1$ opinions, the Descendant Algorithm will run in time $O(n)$ on a path of n nodes, and the Ancestral Algorithm will run in time $O(nT)$ on a path of n nodes for a time-horizon T.

2.3 Outline of Part I

The rest of this part of the thesis is organized as follows. In Chapter 3 we introduce the formalism used in this first part of the thesis, and define the general time-dependent model we will be using to describe the network (the Dynamic Deterministic System – or DDS). Then, in Chapter 4 we restrict our analysis to a time-invariant model (the Static Deterministic System – or SDS), and we develop what we call the Descendant Algorithm for finite and infinite time-horizon problems by analyzing the structure of the underlying graph of the SDS. Finally, in Chapter 5, we study the DDS and develop what we call the Ancestral Algorithm. A brief comparison between the Descendant and the Ancestral Algorithm will conclude that chapter. For both the SDS and the DDS, we will consider both finite and infinite time-horizon problems. Although most of the algorithms are developed for the case where we want to influence all the nodes in the network, we will provide a dynamic program which solves the case in which we only need to influence a fraction of the nodes in the network.

Chapter 3

Model Description

In this chapter, we present the model, notation, and terminology for the first part of the thesis. We also discuss the validity of a key assumption of our model.

3.1 Model Description

We model our network as a directed graph $G = (\mathcal{N}, \mathcal{E})$, in which $\mathcal{N}$ is the set of nodes in the network and $\mathcal{E}$ corresponds to the set of directed edges. We can also allow our network to change with time: in that case, we define a sequence of graphs $G(t) = (\mathcal{N}, \mathcal{E}(t))$ (for $t \in \mathbb{N}$) in which the set of edges can evolve in time.

Each of the nodes can be in state 0 or 1 – this is the *opinion* of the node. Formally, we denote the state (or opinion) of node i at time t by the variable $X_i(t)$, which takes values 0 or 1. We will assume that a controller external to the network wants to drive the network to a final state in which the fraction of 1 opinions is higher than some given parameter η. To achieve his goal, the controller can either wait for the network to update opinions spontaneously (according to the network's intrinsic dynamics which we will define shortly), or can influence a node directly while incurring a cost. Figure 3-1 provides an example of such a network.

The following three definitions introduce some terminology we will use throughout this first part of the thesis. A more detailed formal discussion will be presented in Section 3.3.

Definition 3.1.1. We say that the controller *influences a node i at time t* if the controller

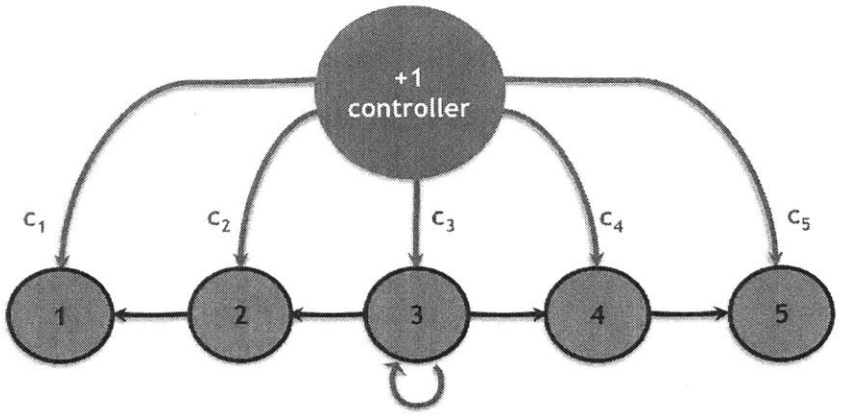


Figure 3-1: In this example, we see a network of 5 nodes. An arrow going from a node to another node means that the former node influences the latter node. The external controller can influence the nodes directly but will incur a certain cost (here a cost c_i for node i).

performs an action at time t to set the state of node i to state 1. The state of node i is then updated at time $t+1$. Formally, the controller will set $X_i(t+1)$ to 1.

Definition 3.1.2. We say that a node j *influences a node* i *at time* t if node j can change the state of node i to his own state at time t: unless the controller influences node i at time t, node j will influence node i at time t. Formally, $X_i(t+1)$ will be set to $X_j(t)$, unless the controller also influences node i at time t. Graphically, if node j influences node i at time t, we draw an arrow from node j to node i at time t.

Definition 3.1.3. We say that a network is *fully influenced at time* t if all nodes in the network are in state 1.

Suppose that only node j can influence node i at time t: if the controller decides not to influence node i at time t, then node i will be influenced by node j at time t; if the controller decides to influence node i at time t, then node i will ignore node j at time t and the state $X_i(t+1)$ of node i at time $t+1$ will be set to 1. We will formalize these rules in Section 3.3.

3.2 The Unique Influencer Assumption

Throughout this first part of the thesis, we will *always* assume that each node has a *unique* influencer at any given time. Graphically, this means that the nodes in the network are constrained to have an in-degree equal to one. Furthermore, we allow the influencer of a node to change with time. However this is done in a deterministic fashion: the controller of the network is assumed to know $\mathscr{E}(t)$ for all $t \in \mathbb{N}$ at the start of the experiment. For example, we can think of a network of three nodes $\{1, 2, 3\}$ such that node 2 influences itself at all times, node 3 influences itself at all times, and node 1 is influenced by node 2 at even times, and is influenced by node 3 at odd times (Figure 3-2).

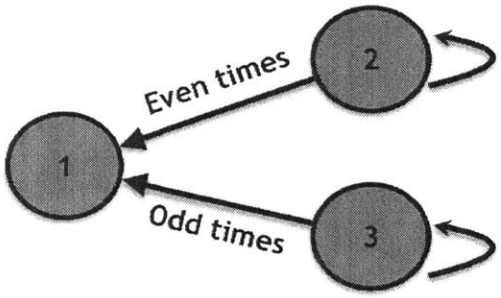


Figure 3-2: In this example, we see a network of 3 nodes that evolves in time. Node 2 influences itself at all times and Node 3 influences itself at all times. However, node 1 is influenced by node 2 at even times, and is influenced by node 3 at odd times.

Why do we restrict ourselves to networks with nodes of in-degree equal to one ? From a computational point of view, we will see in the following chapters that the optimization problem over such a network is solvable in polynomial time. If nodes are allowed to have in-degree higher than one, we need to use more complicated rules for the opinion dynamics: an example would be to consider the Linear Threshold model, which takes the average of the opinions of a node's neighbor and compares it to a fixed threshold. However, hopes to solve the optimization problem for a Linear Threshold model in polynomial time are very thin as Linear Threshold models usually lead to NP-hard optimization problems [9]. Our main goal is to get insight from problems that are tractable.

From the point of view of practical applications, considering only in-degree one nodes may seem restrictive. We partly address this issue by allowing the influencer to change

The state of a node at time $t+1$ depends on the state of:	Example:
- a fixed influencing node at time t	- Static Deterministic Model
- exactly one node at time t (the influencer may change in a deterministic way)	- Dynamic Deterministic Model
- multiple fixed nodes in the network at time t	- Linear Threshold Model
- multiple nodes at time t (the influencers may change in a deterministic way)	- Dynamic Linear Threshold Model

Table 3.1: Classification of different possible deterministic models, in order of increasing complexity.

over time in a deterministic fashion. In-degree one nodes can be viewed as nodes that are influenced only by their most influential neighbor at a given time. Therefore if we are given an arbitrary directed graph with edges weighted by the intensity of the influence a node exerts on his neighbors, we can reduce the in-degree of each node to one by keeping only an incoming edge with a highest weight.

In Table 3.1, we outline a classification of different deterministic models we can work with, in order of increasing complexity. In the simplest case, nodes have in-degree one and the influencer does not change with time: this corresponds to the Static Deterministic Model. If we allow the influencer to change with time we obtain the Dynamic Deterministic Model. Finally, by removing the in-degree one condition, we can obtain respectively a Linear Threshold Model, or a Dynamic Linear Threshold Model. We will always assume here that the influencer(s) are known to the controller at the start of the experiment.

3.3 Formalism

We introduce here the formalism used in the following chapters, and define the general time-dependent model we will be using to describe the network (the Dynamic Deterministic System – or DDS). Suppose that the system is initially in a state $X(1) = (X_1(1), \ldots, X_n(1)) \in \{0,1\}^n$ at $t = 1$. For the moment, it is easier to assume that we start in a state $X(1) = (0, \ldots, 0)$. We will later show how to adapt our results to an arbitrary initial state. Since the nodes in our network have a unique influencer, they have in-degree equal to one. We define the following.

Definition 3.3.1. The **Influencer** $I(t,i)$ (or $I_t(i)$) of node i at time t is the unique node that influences node i at time t. Graphically, we draw a directed edge from $I(t,i)$ to i in the network at time t, and so $\mathscr{E}(t) = \{(j,i) : I_t(i) = j\}$ for all $t \in \mathbb{N}$ (this is well defined because, for all $i \in \mathscr{N}$ and $t \in \mathbb{N}$, $I_t(i)$ is unique). Furthermore, $I(i) = (I_1(i), I_2(i), \ldots, I_T(i))$ is the chronologically ordered set of nodes that influences node i at times $t = 1, 2, \ldots, T$ respectively. The matrix $I = [I(t,i)]$ for $t = 1, 2, \ldots, T$, and $i = 1, 2, \ldots, n$, contains all the information about the dynamics of the system during the whole experiment.

In this deterministic model, we assume that the controller already knows the time horizon T and the entire matrix I at the start of the experiment, and can plan accordingly. At each time, a node i will adopt the state of its influencer $I_t(i)$ unless the controller intervenes.

In the presence of the controller, and in case the controller decides to influence a node, we assume that the node will adopt the state of the controller and will ignore his influencer. When the controller influences a node, it incurs a cost.

Definition 3.3.2. We define $c(t,i)$ (or $c_i(t)$), for $t = 1, 2, \ldots, T$, to be the **cost** (or price) the controller must pay at time t to influence node i. We only require $c(t,i) \geq 0$. The costs of every node at time t are given by the vector $c(t) = \Big(c_1(t), c_2(t), \ldots, c_n(t)\Big)$, for $t = 1, 2, \ldots, T$. The matrix $C = [c(t,i)]$, for $t = 1, 2, \ldots, T$, and $i = 1, 2, \ldots, n$, contains all the information about the cost of influence that the controller may incur.

In this deterministic model, we assume the controller already knows the entire matrix C at the start of the experiment and can plan accordingly. If both matrices I and C are constant in time, we obtain the Static Deterministic Model: in this case, the matrix I can be replaced by the vector $I = \Big(I(1,1), I(1,2), \ldots, I(1,n)\Big)$ and C by the vector $C = \Big(c(1,1), c(1,2), \ldots, c(1,n)\Big)$.

We see that the Dynamic Deterministic Model is specified using only two matrices: $I = [I(t,i)]$ which summarizes the dynamics of the network, and $C = [c(t,i)]$ which contains the costs to the controller if he decides to influence a given node. Hence, from the controller's point of view, we are led to the following definitions.

Definition 3.3.3. Let $\alpha_i(t)$ (with $t \in \{1, \ldots, T-1\}$) represent the **decision** the controller takes about influencing node i at time t:

$$\alpha_i(t) = \begin{cases} 1, & \text{if the controller influences node } i \text{ at time } t, \\ 0, & \text{if the controller does not influence node } i \text{ at time } t. \end{cases}$$

We can now formally define the evolution of the state of the nodes:

Definition 3.3.4. The **state** of a node i at time $t+1$ is given by:

$$X_i(t+1) = \begin{cases} X_{I(t,i)}(t), & \text{if } \alpha_i(t) = 0 \text{ (with cost } = 0) \ , \\ 1, & \text{if } \alpha_i(t) = 1 \text{ (with cost } = c_i(t)) \ . \end{cases}$$

In a more compact form :

$$X_i(t+1) = \alpha_i(t) + \Big(1 - \alpha_i(t)\Big) X_{I(t,i)}(t). \tag{3.1}$$

Our goal is to reach a state $X(T) = (1, \ldots, 1)$ at minimal cost, given an initial state $X(1)$ of the system. In summary, we choose control parameters $\alpha_i(t)$ at each time t, based on the knowledge of I, C, and $X(1)$. We formulate this problem as a mathematical program in Equation 3.2.

$$\begin{aligned} \text{minimize (over } \alpha) \quad & \sum_{t=1}^{T} \sum_{i=1}^{n} c_i(t)\alpha_i(t) \\ \text{subject to} \quad & \sum_{i=1}^{n} X_i(T) = n \\ & \alpha_i(t) \in \{0,1\}, \forall i = 1, \ldots, n, \forall t = 1, \ldots, T \\ & X_i(1) \in \{0,1\} \text{ is given}, \forall i = 1, \ldots, n \\ & X_i(t) \text{ evolves according to Equation 3.1.} \end{aligned} \tag{3.2}$$

When $T = \infty$, the controller wants $\sum_{i=1}^{n} X_i(t) = n$ for all t large enough.

It is also possible to generalize the problem in the following way. Fix some parameter $\eta \in [0,1]$. The controller may want to drive the network to a final state in which the fraction of 1 opinions is higher than the given parameter η. In that case, the mathematical program becomes:

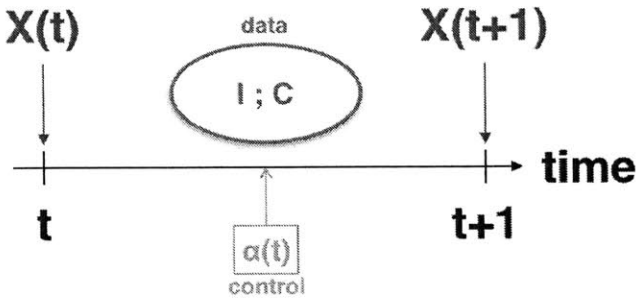


Figure 3-3: Control sequence of the system.

$$\begin{aligned} &\text{minimize (over } \alpha) && \sum_{t=1}^{T}\sum_{i=1}^{n} c_i(t)\alpha_i(t) \\ &\text{subject to} && \tfrac{1}{n}\sum_{i=1}^{n} X_i(T) \geq \eta \\ & && \alpha_i(t) \in \{0,1\}, \forall i = 1,\ldots,n, \forall t = 1,\ldots,T \\ & && X_i(1) \in \{0,1\} \text{ is given}, \forall i = 1,\ldots,n \\ & && X_i(t) \text{ evolves according to Equation 3.1.} \end{aligned} \tag{3.3}$$

When $T = \infty$, the controller wants $\sum_{i=1}^{n} X_i(t) \geq \eta n$ for all t large enough. In this case, we only require the fraction of nodes in the 1 state to be higher than η. Therefore the network must reach, at some finite time, what we will call a *recurrent state*:

Definition 3.3.5. Fix some finite time t_0. The state $X(t_0)$ of the system at time t_0 is called *recurrent* if the network will reach a state $X(t) = X(t_0)$ at some finite time $t > t_0$ without any intervention from the controller.

Chapter 4

The Static Deterministic Model (SDM)

The Static Deterministic Model is based on two time-invariant vectors: $I = (I_1, I_2, \ldots, I_n) := \Big(I(1,1), I(1,2), \ldots, I(1,n)\Big)$ which summarizes the dynamics of the network, and $C = (c_1, c_2, \ldots, c_n) := \Big(c(1,1), c(1,2), \ldots, c(1,n)\Big)$ which contains the costs the controller incurs if he decides to influence a given node. The state of a node i at time $t+1$ is given by:

$$X_i(t+1) = \begin{cases} X_{I_i}(t), & \text{if } \alpha_i(t) = 0 \text{ (with cost } = 0)\ , \\ 1, & \text{if } \alpha_i(t) = 1 \text{ (with cost } = c_i)\ . \end{cases}$$

We assume that we start at a known state $X(1) = \Big(X_1(1), \ldots, X_n(1)\Big)$. In more compact form:

$$X_i(t+1) = \alpha_i(t) + \Big(1 - \alpha_i(t)\Big) X_{I_i}(t) \tag{4.1}$$

In this chapter, we explore properties of time-invariant directed graphs with nodes of in-degree one, and use them to solve our optimization problem by developing the Descendant Algorithm. Unless specified otherwise, we *always* assume the nodes in our networks to have in-degree one.

In Section 4.1, we assume that the network forms a line: we start by analyzing the

structure of such networks and develop the Descendant Algorithm over paths. We finish that section by studying the behavior of the optimal cost as we vary the time horizon T. In Section 4.2, we analyze more general in-degree one graphs and apply the Descendant Algorithm over such networks.

4.1 Analysis for a Line Network

We will start our analysis by assuming our network is a line, according to the following definition.

Definition 4.1.1. A directed graph is called a **line network** if the following holds: when we ignore the directions of the edges and remove duplicate edges and self edges, we are left with an undirected line graph.

For example, all graphs in Figure 4-1 and Figure 4-2 are line networks. We also define the notion of a *(directed) path* in a directed network:

Definition 4.1.2. A **(directed) path** is a line graph in which all edges (excluding self-pointing edges) point in the same direction.

A directed path is a special case of a line graph. We have drawn in Figure 4-1 the typical directed path network.

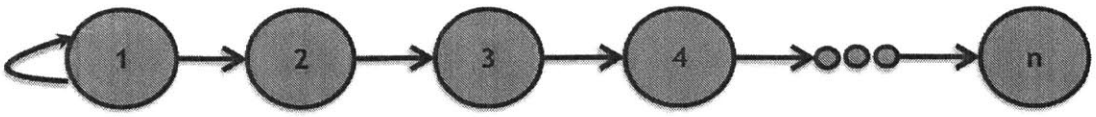
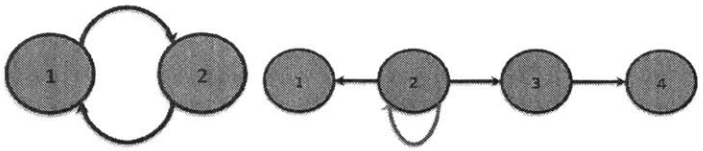
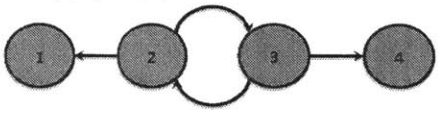


Figure 4-1: Directed path of n nodes: node 1 is the self-influential node, and node n is the endpoint.

In this chapter, we will use the structure of the network to derive the Descendant Algorithm. We will consider both finite and infinite time-horizon problems. We will first assume that the controller wants to influence every node by time T. We will then consider a more general objective by assuming that the controller only wants to influence a fraction η (with $0 < \eta < 1$) of the nodes by time T.



(a) 2-Cycle 2-$C(1,2)$. (b) Linear Chain with 4 nodes $LC(2)$.



(c) Mixed Chain with 4 nodes $MC(2,3)$.

Figure 4-2: Types of connected components encountered in the simplified graph.

4.1.1 Properties of the Graph Representation

We start by analyzing the structure of line networks under the following condition: we assume that all nodes have in-degree equal to one. We first define:

Definition 4.1.3. A **2-Cycle** is a pair of distinct nodes (i, j) in which each node only influences the other one (Fig.4-2a). We will refer to it as the 2-cycle 2-$C(i, j)$. For convenience, we assume node i to have a smaller cost than node j ($c_i \leq c_j$).

Definition 4.1.4. A **Linear Chain** is a connected set of nodes on the line with no cycles present, but with exactly one self-influencing node i (Fig.4-2b). We will refer to it as the linear chain $LC(i)$.

Definition 4.1.5. A **Mixed Chain** is a connected set of nodes on the line with exactly one 2-cycle (i, j) (Fig.4-2c). We will refer to it as the mixed chain $MC(i, j)$. For convenience, we assume node i to have a smaller cost than node j ($c_i \leq c_j$).

Remark. Note that a 2-cycle is a special case of a mixed chain.

Given a network, we say that a subset of nodes $\mathcal{N}_1$ (with $\mathcal{N}_1 \subset \mathcal{N}$) forms a *connected component* if, after ignoring the directions of the edges and removing duplicate edges and self edges, we are left with an undirected connected graph which contains no edges between $\mathcal{N}_1$ and $\mathcal{N} \setminus \mathcal{N}_1$. We now show that any line network is a disjoint collection of linear chains and of mixed chains.

Theorem 4.1.6. *In a line network, a connected component must be either:*

1. *a linear chain $LC(i)$, or*

2. *a mixed chain $MC(i,j)$ (which could be a a 2-cycle 2-$C(i,j)$).*

To prove this theorem, it is useful to note the following properties of the network:

Lemma 4.1.7. *The number of edges equals the number of nodes.*

Proof. Suppose that there are n nodes. Since nodes have in-degree one, we must have one incoming edge per node, hence n edges in total. □

Fact. *If a line network contains a cycle, it is a 2-cycle.*

Lemma 4.1.8. *In a line network, a connected set of k nodes either has a single 2-cycle or a single node influencing itself.*

Proof. Suppose that the network is not a 2-cycle: then by the previous fact, the network is not a cycle. By Lemma 4.1.7, we must have exactly k edges connecting these k nodes. To connect k nodes on a line, we need only $k-1$ edges. However, if we only have $k-1$ edges for k nodes, there must be a node i with in-degree zero. The additional edge can either be added from node i to itself, or from a neighbor of node i to node i. In the former case, we get the unique self-influencing node i. In the latter case, node i and his neighbor will form a 2-cycle. □

We can now return to the proof of Theorem. 4.1.6:

Proof. (Theorem. 4.1.6) From the previous lemmata, we have shown that a connected set of k nodes must have either a unique 2-cycle, or a unique self-influencing node. If this connected set of nodes has a unique self-influencing node, the graph is a linear chain. If this connected set of nodes has a unique 2-cycle, the graph is a mixed chain. □

The following property will be very useful to construct an optimal policy to our control problem.

Proposition 4.1.9. (The Cascade Property) *If we remove the self-influencing node (resp. 2-cycle) of a linear chain (resp. mixed chain), we obtain two chains (one of which*

may be empty). In each chain, all arrows point in the same direction. In other words, edges go away from a self-influencing node, or a 2-cycle.

Proof. If we remove a self-influencing node from a linear chain, we remove 3 edges if the self-influencing node has two neighbors, and 2 edges if the self-influencing node has only one neighbor. In the second case, we get a unique chain: the neighbor of the removed node now has in-degree zero, and all edges in the chain must have the same orientation as the edge between the self-influencing node (otherwise some node will have in-degree 2). In the first case, we get two chains: the argument is similar to the one-chain argument.

Alternate proof. If an edge is going towards a self-influencing node, then this node has in-degree two which is forbidden. If an edge goes towards a 2-cycle, then one node of the 2-cycle has in-degree two which is forbidden. □

We now have a complete characterization of the types of connected components we can encounter in the line network. This will immediately provide an optimal policy for our control problem over line networks, when $T = \infty$ and $\eta = 1$.

Infinite Time Horizon Case ($T = \infty$) with $\eta = 1$:

We start with the case of an infinite time horizon ($T = \infty$) where $\eta = 1$ (we want to influence all nodes). This means that the controller's objective is to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. We also assume that $X(1) = \Big(X_1(1), \ldots, X_n(1)\Big)$.

Proposition 4.1.10. *If $T = \infty$ and $\eta = 1$:*

1. *Suppose the controller wants to fully influence a 2-cycle 2-$C(i,j)$ (with $c_i \leq c_j$). An optimal policy is to influence node i at times $t = 1$ and $t = 2$ (i.e., choose $\alpha_i(1) = (1 - X_j(1))$, $\alpha_i(1) = (1 - X_i(1))$ and set all other components of $\alpha(t)$ to zero). The total cost of influence for this connected component will be: $c_i\Big(2 - X_i(1) - X_j(1)\Big)$. The whole component will be entirely in the $+1$ state at $t = 3$.*

2. *Suppose the controller wants to fully influence a linear chain $LC(i)$. An optimal policy is to influence node i at time $t = 1$ (i.e., choose $\alpha_i(1) = 1 - X_i(1)$ and set all other*

components of $\alpha(t)$ to zero). The total cost of influence for this connected component will be: c_i. The whole component will be entirely in the $+1$ state at $t = k+1$, where k is the number of nodes in the longest path starting at i.

3. *Suppose the controller wants to fully influence a mixed chain $MC(i,j)$ (with $c_i \leq c_j$). An optimal policy is to influence node i at times $t = 1$ and $t = 2$ (i.e., choose $\alpha_i(1) = (1 - X_j(1))$, $\alpha_i(2) = (1 - X_i(1))$ and set all other components of $\alpha(t)$ to zero). The total cost of influence for this connected component will be: $c_i\Big(2 - X_i(1) - X_j(1)\Big)$. The whole component will be entirely in the $+1$ state at time $t = k+1$, where k is the number of nodes in the longest path or cycle starting at i.*

In general, the optimal cost for fully influencing the whole line network is given by:

$$\sum_{LC(i)} c_i\Big(1 - X_i(1)\Big) + \sum_{MC(i,j)} c_i\Big(2 - X_i(1) - X_j(1)\Big), \tag{4.2}$$

where we sum the costs over all Linear Chains and all Mixed Chains in the network. The time to influence the whole network is equal to:

$$T^* = \max_{\substack{\text{all connected} \\ \text{components}}} \begin{cases} 3, & \text{if in a 2-cycle } 2\text{-}C(i,j), \\ 1+\text{number of nodes in the longest path starting at } i, & \text{if in a linear chain } LC(i), \\ 1+\text{number of nodes in the longest path or cycle starting at the node to influence } (i \text{ or } j), & \text{if in a mixed chain } MC(i,j). \end{cases} \tag{4.3}$$

Proof. For simplicity, assume that $X(1) = (0, \ldots, 0)$.

1. Suppose the controller wants to fully influence a 2-cycle $2-C(i,j)$. We are considering an infinite time-horizon problem, therefore the controller wants to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. Notice that if the controller only influences one node in the cycle, only one node will be in the $+1$ state at each time. Therefore the controller can influence either both nodes simultaneously, or one node at two consecutive times. Clearly, influencing the cheapest node twice

will be an optimal solution. It is easy to check that an optimal policy the controller can apply is to influence node i at times $t = 1$ and $t = 2$ (i.e., choose $\alpha_i(1) = \alpha_i(2) = 1$ and set all other components of $\alpha(t)$ to zero).

2. Suppose the controller wants to fully influence a linear chain $LC(i)$. We are considering an infinite time-horizon problem, therefore the controller wants to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. For node i to reach a $+1$ state, the controller will need to influence node i at some finite time, because node i is only influenced by itself. Once the controller does that, node i will remain indefinitely in the $+1$ sate. Therefore the optimal cost of the control problem is at least equal to c_i. By the Cascade Property (Proposition. 4.1.9), we see that node i will spontaneously influence both sides of the linear chain in a finite amount of steps. This cascade effect does not require any action from the controller. Therefore this policy has a cost equal to c_i, and our policy must be optimal.

3. Suppose the controller wants to fully influence a mixed chain $MC(i, j)$. We are considering an infinite time-horizon problem, therefore the controller wants to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. First, we view the cycle (i, j) as a unique self-influencing node. By part (2), we see that the controller must influence the cycle directly in order to reach the desired steady state. Once the cycle is in the $(1, 1)$ state, the whole mixed chain will be in the $+1$ state by the Cascade Property (Proposition. 4.1.9). We now use part (1) to influence the cycle at an optimal cost.

For an arbitrary initial state $X(0)$, we need to change our policy only if the self-influencing node, or a node of the 2-cycle is already in the $+1$ state at time $t = 0$. The policies described in the statement of the proposition are clearly optimal.

□

We can now use the previous proposition to solve the infinite time-horizon control problem when $0 < \eta < 1$.

Infinite Time Horizon Case ($T = \infty$) With $0 < \eta < 1$:

Suppose that, in our infinite time-horizon problem, the controller only wants to influence a given fraction η of the nodes from the network (where $0 < \eta < 1$). This means that the controller wants $\sum_{i=1}^{n} X_i(t) \geq \eta n$ at all times t large enough. We still need to reach a recurrent state in a finite (but arbitrarily large) amount of time. The line network is a disjoint collection of linear chains and mixed chains. Since the controller does not need to get all nodes in the $+1$ state, we simply need to influence the cheapest connected components of the network in order to get at least ηn nodes: this is clearly a knapsack problem. For simplicity, assume that $X(1) = \big(0, \ldots, 0\big)$.

Proposition 4.1.11. *For simplicity, assume that* $X(1) = \big(0, \ldots, 0\big)$. *If* $T = \infty$ *and* $0 < \eta < 1$*:*

1. *Suppose we have a 2-cycle 2-*$C(i, j)$ *(with* $c_i \leq c_j$*):*

   To have only one node in the $+1$ *state, an optimal policy will be to influence node* i *once at time* $t = 1$ *(i.e.,* $\alpha_i(1) = 1$ *and set all other components of* $\alpha(t)$ *to zero). This is done at a cost* c_i *by time* $t = 2$.

   To have both nodes in the $+1$ *state, an optimal policy will be to influence node* i *at times* $t = 1$ *and* $t = 2$ *(i.e.,* $\alpha_i(1) = \alpha_i(2) = 1$*). This is done at a cost* $2c_i$*, by time* $t = 3$.

2. *Suppose we have a linear chain* $LC(i)$*:*

   Since $T = \infty$*, the controller can influence the node* i *at time* $t = 1$ *(i.e.,* $\alpha_i(1) = 1$ *and set all other components of* $\alpha(t)$ *to zero). Note that this procedure will eventually influence all nodes in the linear chain. This is done at a cost* c_i*. The whole linear chain will be entirely in the* $+1$ *state by time* $t = k+1$*, where* k *is the number of nodes in the longest path starting at node* i.

Proof. This proposition clearly follows from Proposition 4.1.10. □

When we have a Mixed Chain $MC(i, j)$ (with $c_i \leq c_j$), the problem of influencing a fraction η of nodes in the network should be formulated differently. Indeed, when we reach

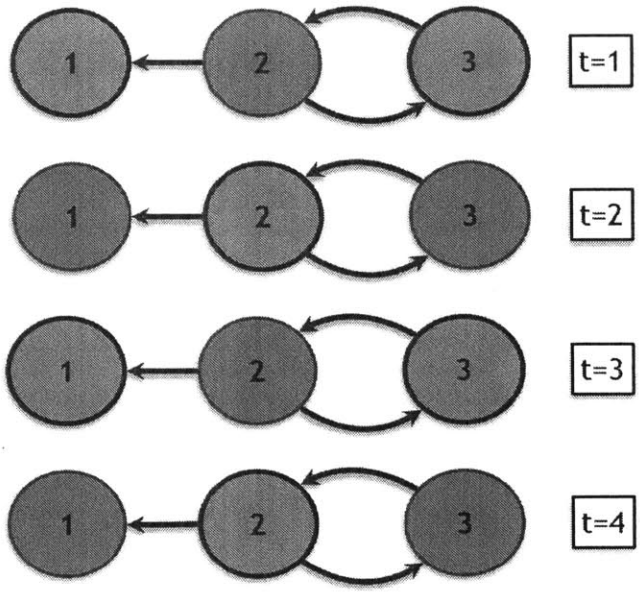


Figure 4-3: Mixed chain of three nodes evolving in time: red nodes are nodes in the +1 state while blue nodes are in the 0 state. Notice that the number of nodes in the +1 state oscillates with time: there is one node in the +1 state at odd times, and there are two nodes in the +1 state at even times.

a recurrent state and then let the network evolve spontaneously from there, the number of nodes in the +1 state may not remain constant. For example, in Figure 4-3, the network oscillates between a state in which two nodes are in state +1 (this happens at even times), and a state in which only one node is in state +1 (this happens at odd times).

For Mixed Chains, when we influence one node of the cycle, we eventually reach a recurrent state. We show below that a recurrent state can either have $\lfloor \frac{n}{2} \rfloor$ nodes in the +1 state, or $\lceil \frac{n}{2} \rceil$ nodes in the +1 state. Actually, when the system is in one of these recurrent states at time t, it will be in the other recurrent state at time $t+1$. Therefore, if we take a time-average of the number of nodes in the +1 state after reaching a recurrent state, we will have "on average" $\frac{n}{2}$ nodes in the +1 state.

Proposition 4.1.12. *Suppose we have a mixed chain $MC(i,j)$ (with $c_i \leq c_j$):*

1. *To influence "on average" half of the nodes, we can influence only node i at $t=1$ with a value $\alpha_i(1)=1$, at a cost c_i, by time $t=k+1$ (where k is the number of nodes in the longest path starting at node i).*

2. *To influence all nodes, we can influence node i at $t = 1$ and $t = 2$, with a value $\alpha_i(0) = \alpha_i(1) = 1$, at cost $2c_i$, by time $t = k + 1$ (where k is the number of nodes in the longest path starting at node i).*

Proof. 1. By influencing only one node in the Mixed Chain's 2-cycle, we eventually reach a recurrent state. Observe that such recurrent states must be of the form $X = (X_1, X_2, \ldots, X_n)$ where $X_1 = 0$ or 1, and $X_i = 1 - X_{i-1}$ for all $i \in \{1, \ldots, n\}$ (i.e., $X = (1, 0, 1, 0, \ldots, X_n)$ or $X = (0, 1, 0, 1, \ldots, X_n)$ in such a recurrent state). Once such a recurrent state is reached, the system will alternate between these two states at each iteration. When $X_1 = 0$, the recurrent state has $\lfloor \frac{n}{2} \rfloor$ nodes in the $+1$ state, and when $X_1 = 1$, the recurrent state has $\lceil \frac{n}{2} \rceil$ nodes in the $+1$ state. Therefore, if we take a time-average of the number of nodes in the $+1$ state after reaching a recurrent state, we will have "on average" $\frac{n}{2}$ nodes in the $+1$ state.

2. This proposition clearly follows from Proposition 4.1.10.

□

Proposition 4.1.13. *We are given a network consisting of a collection of Linear Chains and Mixed Chains. To eventually reach a fraction η of nodes in the $+1$ state, we can follow the procedure outlined in Proposition 4.1.11 and Proposition 4.1.12, and choose the connected components to influence by solving the following knapsack problem.*

$$\text{Let } \alpha_i(t) = \begin{cases} 1, & \text{if we influence node } i \text{ at time } t, \\ 0, & \text{if we don't influence node } i \text{ at time } t. \end{cases}$$

We solve:

$$\begin{aligned} \textit{minimize} \quad & \sum_{LC(i)} \alpha_i(1) c_i + \sum_{MC(i,j)} \Big(\alpha_i(1) + \alpha_i(2)\Big) c_i \\ \textit{subject to} \quad & \eta n \leq \sum_{LC(i)} \alpha_i(1) |LC(i)| + \sum_{MC(i,j)} \frac{|MC(i,j)|}{2} \Big(\alpha_i(1) + \alpha_i(2)\Big), \\ & \alpha_i(t) \in \{0, 1\}, \forall i = 1 \ldots n, \forall t = 1, 2 \end{aligned} \tag{4.4}$$

where the sums run over all Linear Chains and all Mixed Chains in the network.

Proof. This proposition clearly follows from Proposition 4.1.11 and Proposition 4.1.12. □

Equation (4.4) involves complicated notation, but the idea behind it is simple: we want to select the cheapest connected components and decide how many nodes we need to influence in each case. This basically boils down to a knapsack problem.

4.1.2 The Descendant Algortithm

We will now consider finite time-horizon problems, first assuming that $\eta = 1$, and then for a general $\eta \in (0, 1)$. In the former case, we will solve the control problem by developing the Descendant Algorithm (DA). We will also show that the DA can be extended to the infinite time-horizon problem. Throughout this section, we will assume for simplicity that $X(1) = (0, \ldots, 0)$.

Finite Time-Horizon Case ($T < \infty$) With $\eta = 1$:

Note that when T is finite, Proposition 4.1.10 still holds if $T \geq T^*$ (where T^* is defined in Equation (4.3)). T^* is interpreted as the smallest time to reach the targeted state under an optimal policy. If $T < T^*$, the influencing process must be accelerated by influencing more nodes, so that the entire network will be in the $+1$ state at time T. The following proposition provides an optimal policy:

Proposition 4.1.14. *Suppose that $T < \infty$ and $\eta = 1$.*

1. *Suppose that the controller wants to fully influence a 2-cycle $2 - C(i, j)$. In this case, $T^* = 3$. Hence if $T \geq 3$, we just need to apply the optimal policy given for the infinite time-horizon case. If $T = 2$, then the controller must influence both nodes at $t = 1$ (i.e. $\alpha_i(1) = \alpha_j(1) = 1$) at a cost of $c_i + c_j$.*

2. *Suppose that the controller wants to fully influence a linear chain $LC(i)$. In this case, an optimal policy is provided by applying the "Descendant Algorithm on a Path" (see below) on each path of $LC(i)$ from node i to an end of the chain (i.e., a node l with out-degree equal to zero). Note that Linear Chains can have either one or two such paths (or none: in that case the Linear Chain $LC(i)$ consists only of node i).*

3. *Suppose that the controller wants to fully influence a mixed chain $MC(i,j)$. If $T = 2$, then an optimal policy consists of influencing every node in the mixed chain at $t = 1$. If $T \geq 3$, an optimal policy is reached by influencing the cheapest node in the cycle (here node i) at times $t = 1$ and $t = 2$; we then apply the "Descendant Algorithm on a Path" (see below) on each path from node i to an end of the chain (i.e., a node l with out-degree equal to zero). In other words, once we influence node i at $t = 1$ and $t = 2$, we can ignore the influence node j has on node i (remove this edge) and treat the problem as a Linear Chain.*

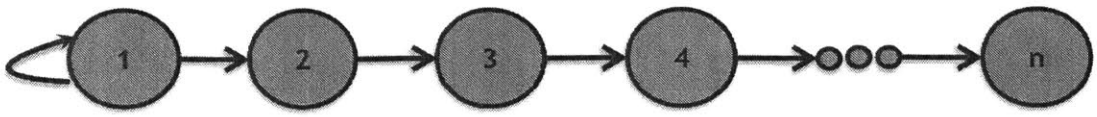


Figure 4-4: Linear chain of n nodes: node 1 is the self-influencing node, and node n is the endpoint.

DESCENDANT ALGORITHM ON A PATH:

Without loss of generality, call this path $(1, 2, 3, \ldots, n)$ where 1 is the self-influencing node, and i influences $i + 1$ for all $i \leq n - 1$ (see Fig.4-4). We want to influence the whole chain by time T and at minimal cost.

Detailed Algorithm:

1. Let $j_0 = 1$ and influence node j_0 at time $t = 1$. This will spontaneously influence nodes $\{j_0, j_0 + 1, \ldots, \min(T - 1 + j_0 - 1, n)\} = \{1, 2, \ldots, \min(T - 1, n)\}$ until time T, at a cost $c_{j_0} = c_1$.

   If $T - 1 = T - 1 + j_0 - 1 \geq n$, then STOP. Otherwise, go to the next step.

2. Let j_1 be the node with cheapest cost c_i among the set of nodes $\{j_0 + 1, j_0 + 2, \ldots, \min(j_0 + T, n)\}$ (if there is a tie, pick one of the cheapest nodes arbitrarily). Let j_2 be the cheapest node in the set $\{T - 1 + j_0, T - 1 + j_0 + 1, \ldots, \min(T - 1 + j_1 - 1, n)\}$.

   If j_1 is cheaper than j_2, influence node j_1 $\min(j_1 - j_0, n - T + 1 - j_0 + 1)$ consecutive

times starting at $t = 1$. This will spontaneously influence nodes $\{T-1+j_0, T-1+j_0+1, \ldots, \min(T-1+j_1-1, n)\}$, at a cost $c_{j_1} \min(j_1 - j_0, n-T+1-j_0+1)$.

If j_2 is strictly cheaper than j_1, influence node j_1 $(j_2 - j_0 - T + 1)$ consecutive times starting at $t = 1$. This will spontaneously influence nodes $\{T-1+j_0, T-1+j_0+1, \ldots, j_2-1\}$, at a cost $c_{j_1}(j_2 - j_0 - T + 1)$.

If $T - 1 + j_1 - 1 \geq n$, then STOP. Otherwise, go to the next step.

3. At step k: Let j_k be the node with cheapest cost c_i among the set of nodes $\{j_{k-1}+1, j_{k-1}+2, \ldots, \min(j_{k-1}+T-1, n)\}$ (if there is a tie, pick any of the cheapest node). Let j_{k+1} be the cheapest node in the set $\{T-1+j_{k-1}, \ldots, \min(T-1+j_k-1, n)\}$.

   If j_k is cheaper than j_{k+1}, influence node j_k $\min(j_k - j_{k-1}, n - T + 1 - j_k + 1)$ consecutive times starting at $t = 1$. This will spontaneously influence nodes $\{T-1+j_{k-1}, T-1+j_{k-1}+1, \ldots, \min(T-1+j_k-1, n)\}$, at a cost $c_{j_k} \min(j_k - j_{k-1}, n-T+1-j_{k-1}+1)$.

   If j_{k+1} is strictly cheaper than j_k, influence node j_k $(j_{k+1} - j_k - T + 1)$ consecutive times starting at $t = 1$. This will spontaneously influence nodes $\{T-1+j_{k-1}, T-1+j_{k-1}+1, \ldots, j_{k+1}-1\}$, at a cost $c_{j_k}(j_{k+1} - j_{k-1} - T + 1)$.

   If $T - 1 + j_k - 1 \geq n$, then STOP. Otherwise, go to the next step.

If the algorithm terminates at $k = k^*$ then the total cost to influence the whole chain is equal to: $c_{j_0} + \sum_{k=1}^{k^*-1} c_{j_k} * \min(j_k - j_{k-1}, j_{k+1} - j_{k-1} - T + 1) + (n - T + 1 - j_{k^*-1} + 1) c_{j_{k^*}}$.

This algorithm obviously terminates since the j_k form a strictly increasing sequence of integers which is bounded above by n, hence $1 \leq j_{k^*} \leq n - T + 1$ when $T < n$, and $j_{k^*} = 1$ for $T \geq n$. Hence the run-time of the algorithm is at worst of order $O(nT)$.

Proof. If we influence the self-influencing node, then we will have influenced the first $T-1$ nodes in the path by time T. We now need to influence the remaining $n - T + 1$ nodes by influencing any non-self-influencing node of the path: this means that we must choose $n - T + 1$ nodes to influence among the remaining $n - 1$ non-self-influencing nodes at

convenient times (the nodes chosen do not need to be distinct).

We observe that each node can influence at most $T-1$ nodes (including himself) by time T. Therefore, in an optimal policy, the cheapest node j_1 in $\{2,\ldots,T\}$ should be used to influence the whole set $\{T,\ldots,T-1+j_1-1\}$. Actually, if some node $j_2 \in \{T,\ldots,T-1+j_1-1\}$ has a strictly cheaper cost than j_1, then j_1 will influence only the set $\{T,\ldots,j_2-1\}$ (because j_2 influences the rest at a strictly cheaper cost). We are now guaranteed to have influenced all nodes in $\{T,\ldots,j_2-1\}$ at an optimal cost. If it was not optimal, then some other node $j \in \{2,\ldots,j_2-1\}$ would be strictly cheaper than j_1. This is impossible because, by definition, j_1 is the cheapest node of that set.

We continue analogously by looking choosing j_2 among the nodes $\{j_1+1,\ldots,T-1+j_1-1\}$, etc. The same idea applies.

□

The algorithm above is presented in much detail in order to understand better the machinery behind it. We now present a simplified version of the Descendant Algorithm.

Simplified Algorithm:

We first select the nodes we will influence directly. This will be a collection $J = \{j_0, j_1, \ldots, j_{k^*}\}$ of nodes in $\{1,\ldots,n\}$.

1. Let $j_0 = 1$.
2. Let j_k be the cheapest node in $\{j_{k-1}+1,\ldots,\min(j_{k-1}+T-1,n)\}$ for all $k \geq 1$.
3. Stop when $k = k^*$, where k^* is defined by $j_{k^*-1} \geq n-T+1$.

We now calculate the number N_k of consecutive times we want to influence node j_k:

$$N_k = \min(j_k - j_{k-1}, j_{k+1} - j_{k-1} - T + 1),\ \forall 1 \leq k \leq k^*.$$

If the algorithm terminates at $k = k^*$ then the total cost to influence the whole

chain was equal to:

$$c_{j_0} + \sum_{k=1}^{k^*-1} c_{j_k} \min(j_k - j_{k-1}, j_{k+1} - j_{k-1} - T + 1) + (n - T + 1 - j_{k^*-1} + 1)c_{j_{k^*}}.$$

The intuition behind the Descendant algorithm is that we pick a node and see how many times we should influence it directly to reach the largest number of descendants of that node by the time-horizon.

Finite Time-Horizon Case ($T < \infty$) With $0 < \eta < 1$:

If T is finite and $0 < \eta < 1$, the solution is much trickier. Since we don't need to influence the entire network, and the time-horizon is finite, then we may not want to influence systematically the self-influencing nodes or nodes in 2-cycles. For example, consider the following linear chain $LC(2)$ of four nodes:

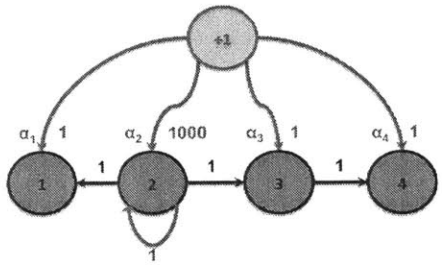


Figure 4-5: How things can go wrong when $T < T^*$ and $0 < \eta < 1$.

Example 4.1.15. Consider the $LC(2)$ linear chain in Figure 4-5 (the costs per unit influence are written in red next to the decisions α). Assume that $X(1) = (0, \ldots 0)$.

Clearly, if $T = \infty$ and $\eta = 1$, then we must influence only node 2 (i.e., set $\alpha_2(1) = 1$), and the optimal cost to influence the linear chain is equal to 1000. Note that $T^* = 3$: if $T \geq 3$, the previous policy remains optimal as we need to influence all nodes, so we must influence node 2 directly at some point. If $T < 2$, then we will have to influence

more nodes: for example, if $T = 1$, we need to influence all nodes in the network (the optimal cost will be equal to 1003). If $T = 2$, an optimal policy would be to influence node 2 at $t = 1$, and then influence node 4 at $t = 2$ (the optimal cost is equal to 1001); another optimal policy would be to influence nodes 2 and 3 at $t = 1$ (the optimal cost is equal to 1001).

If $T = \infty$ and $0 < \eta < 1$, the optimal policy will also be to influence node 2 at $t = 1$ at a cost of 1000. Indeed, because $T = \infty$, we cannot influence any smaller number of nodes. For example take $\eta = 50\%$: then the controller wants to have at least 2 nodes in the $+1$ state at $T = \infty$. If we don't influence node 2, we will have to influence other nodes (say only node 3) at infinitely many times: this leads to an infinite cost.

The case is much trickier when T is finite and $0 < \eta < 1$. Let $\eta = 0.5$: in this example, an optimal policy will be to influence nodes 3 and 4 at time T (or influence node 3 at times $T - 1$ and T). The optimal cost is therefore equal to 2 (which is much cheaper than if we would influence node 2). However this solution is "unstable" as all nodes will go back to the 0 state at time $T + 2$: this may not be an issue if the problem we are interested in does not care about the evolution of the system after time T (for example, during elections, T would be the time at which the voting sites close: a party wants to have enough voters in its favor at that time, but the opinion of the voters after that time does not matter much ... until the next elections perhaps !).

From the previous example, we see that optimal policies for finite T and $0 < \eta < 1$ would consist of influencing "cheap" nodes at times "close" to T. It is possible to find a similar approach as that of the Descendant Algorithm, however the Ancestral Algorithm we will develop later on will provide a much simpler way of tackling this case.

4.1.3 More About the Optimal Cost vs. Time Horizon Plot

Throughout this section, we assume that $\eta = 1$ and $X(1) = (0, \ldots, 0)$. The plots presented here are obtained by running the "Descendant Algorithm on a Path". Notice that the first point (at T=1) has an optimal cost equal to the sum of all the costs of the nodes in the path,

and the last point (when T is equal to the number of nodes in the path) has an optimal cost equal to the cost of the first node (the self-influencing node).

In Fig. 4-6, we consider a path of 10 nodes $\{1, 2, \ldots, 10\}$ with associated costs $C = (8, 1, 3, 1, 1, 9, 7, 4, 10, 1)$. We then plot the value of the optimal cost as a function of the time-horizon T. We immediately see from this example that such a function is not necessarily convex. However, it is easy to see that such a function will always be non-increasing, as increasing the time-horizon cannot increase the optimal cost.

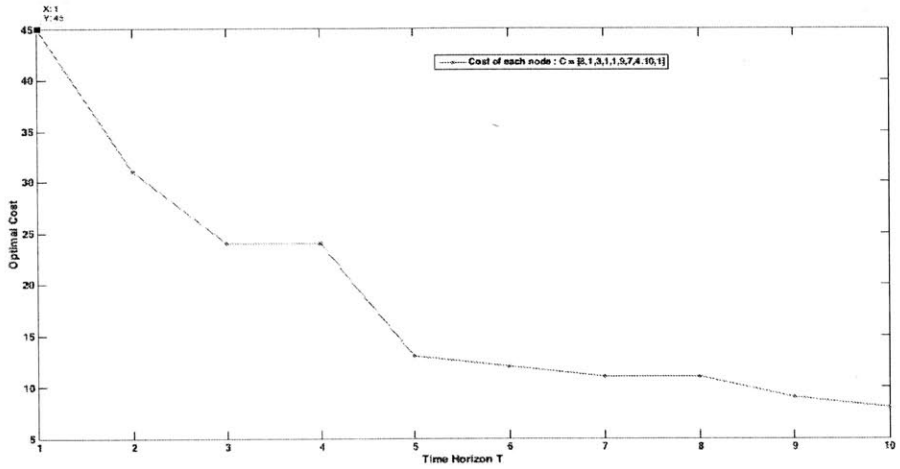


Figure 4-6: Example of an Optimal Cost vs. Time Horizon Plot: this example shows that the function does not need to be convex.

In Figures 4-7 to 4-9, we consider a path of n nodes $\{1, 2, \ldots, n\}$ for large values of n. We generate costs $c_1, \ldots, c_n$ independently from a discrete uniform distribution over $\{1, \ldots, 10\}$. We then plot the value of the optimal cost as a function of the time-horizon T. In Fig. 4-7, we consider a path of 1000 nodes in which the cost of each node is generated from a discrete uniform distribution over $\{1, \ldots, 10\}$; we then plot the corresponding optimal-cost vs. time-horizon graph. In Fig. 4-8, we generate 200 paths of 1000 nodes and plot their corresponding optimal-cost vs. time-horizon plot. Finally, in Fig. 4-9 we consider a path of 10^4 nodes; we then plot the corresponding optimal-cost vs. time-horizon graph.

We notice the following trend for long paths: the optimal cost tends to decay sharply for smaller values of T, and linearly for values of T closer to T^*. Indeed, for smaller values of T, we need to influence fewer nodes as we increase the value of T and therefore we stop

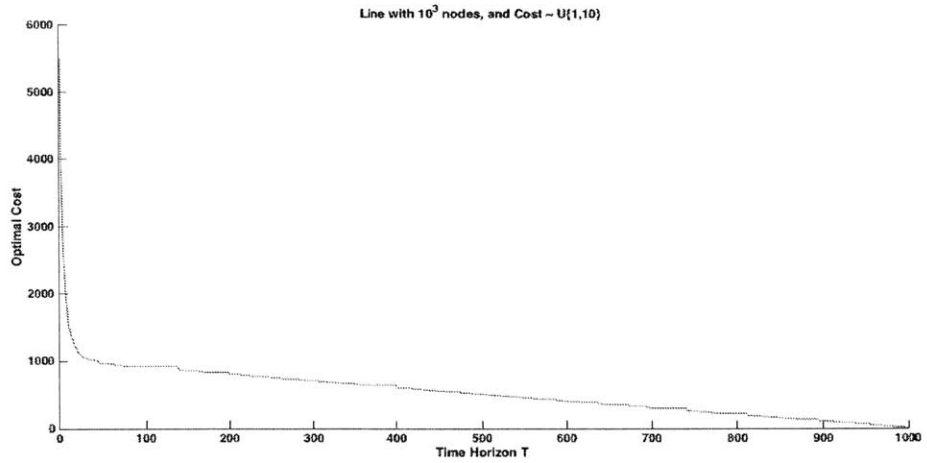


Figure 4-7: Example of an Optimal Cost vs. Time Horizon Plot for 10^3 nodes with node costs drawn uniformly at random in $\{1, \ldots, 10\}$. We observe that the optimal cost decreases sharply as T grows until $T \approx 50$; then the optimal cost seems to decrease linearly as T grows.

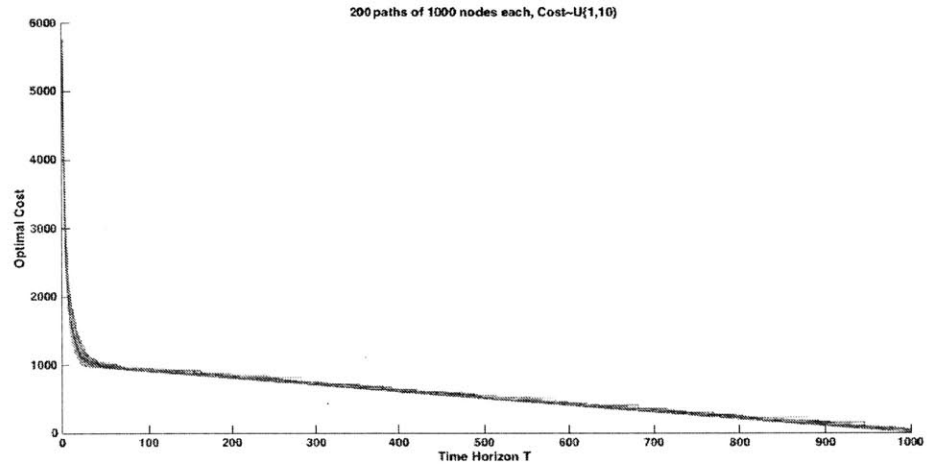


Figure 4-8: 200 Optimal Cost vs. Time Horizon Plot with node costs drawn uniformly at random in $\{1, \ldots, 10\}$. A pattern seems to emerge: the function decreases very fast at the beginning and then goes linearly to the cost of the self-influencing node.

influencing some expensive nodes. Depending on the distribution of the costs along the path, we are left mainly with cheaper nodes after some time (we will define properly this cutoff time as T_c in the next proposition). This implies that the optimal cost will decrease slower for larger values of T.

We can now try to be more quantitative about the optimal cost vs. time horizon curves

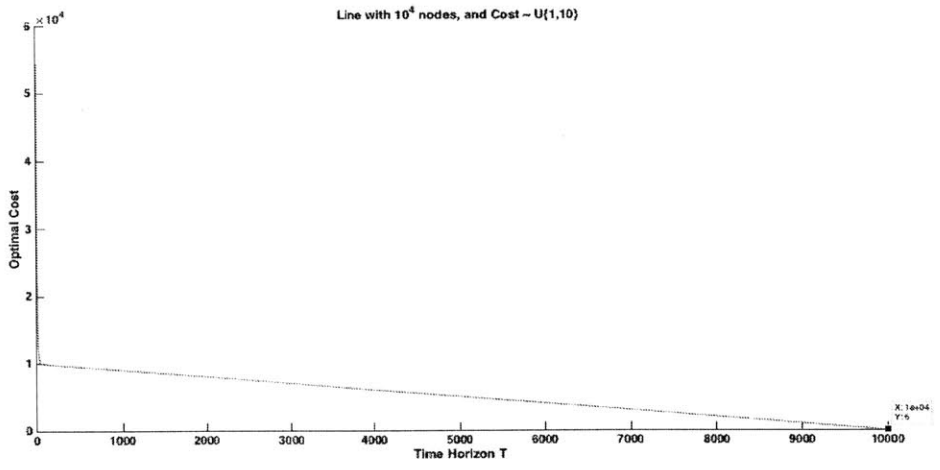


Figure 4-9: Example of an Optimal Cost vs. Time Horizon Plot for 10^4 nodes with node costs drawn uniformly at random in $\{1, \ldots, 10\}$. We conjecture that, for values of T greater than $\log(n)$, and as the number of nodes goes to infinity, the optimal cost function tends to the line $n - (T + 1)$.

(e.g., see Fig.4-8):

We generate a path of n nodes by randomly choosing the costs $\mathbb{C}_i$ of the nodes according to some Probability Mass Function (PMF) $P_{\mathbb{C}}(c)$ defined for positive values of the cost c. For simplicity, we can assume that the cost $\mathbb{C}_i$ ($\mathbb{C}_i \sim P_{\mathbb{C}}$, $i = 1, \ldots, n$ are independent and identically distributed) satisfy $P_{\mathbb{C}}(c) = 0$ for all $c < 1$, and $P_{\mathbb{C}}(1) > 0$. Define $\widehat{\mathbb{C}}(T)$ as the optimal cost at time T.

Approximate Calculation 4.1.16. Under such a construction:

$\mathbb{E}[\widehat{\mathbb{C}}(T = 1)] = n\mathbb{E}[\mathbb{C}] = n\sum_{c=1}^{\infty} cP_{\mathbb{C}}(c)$

$\mathbb{E}[\widehat{\mathbb{C}}(T = n+1)] = \mathbb{E}[\mathbb{C}_1] = \sum_{c=1}^{\infty} cP_{\mathbb{C}}(c)$ (which is the expected cost of the self-influencing node)

Define $c_0 = \min_{i=1,\ldots,n} \mathbb{C}_i$ as the cheapest cost in the path. We assume that the path is long enough (with respect to $1/P_{\mathbb{C}}(1)$), so that we can make the approximation that $c_0 = 1$. Call T_c the earliest time at which the optimal policy influences only nodes with cost c_0 (except possibly for the self-influencing node of the path). T_c corresponds to the time at which the behavior of the curves becomes approximately linear. Then we can approximate $\mathbb{E}[T_c]$ by:

$$\mathbb{E}[T_c] \approx n - \sum_{n_0=1}^{n} \frac{\binom{n}{n_0}}{1-(1-p)^n} p^{n_0}(1-p)^{n-n_0} \sum_{t=1}^{n} (1-(1-p)^t)^{n_0}, \tag{4.5}$$

or

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{n-1} \frac{1-\left(1-p(1-p)^t\right)^n}{1-(1-p)^n} \tag{4.6}$$

where $p = P_{\mathbb{C}}(c_0)$

Justification. First, $\mathbb{E}[\widehat{\mathbb{C}}(T=1)] = \mathbb{E}[\sum_{i=1}^{n} \mathbb{C}_i] = n\mathbb{E}[\mathbb{C}]$ because when $T=1$ we must influence all nodes on the path, so the optimal cost is just the sum of the costs of all nodes. Also, $\mathbb{E}[\widehat{\mathbb{C}}(T=n)] = \mathbb{E}[\mathbb{C}_1]$ because when $T \geq n$ we only need to influence the first node of the path, as its influence has enough time to propagate through the entire path.

Assume that the path is long enough, so that we have $c_0 = 1$. For $i = 1, \ldots, n$, let Y_i be the following indicator random variable:

$$Y_i = \begin{cases} 1, & \text{if } \mathbb{C}_i = c_0, \\ 0, & \text{if } \mathbb{C}_i > c_0. \end{cases}$$

Hence Y_i are IID Bernoulli random variables with parameter $p = P_{\mathbb{C}}(c_0)$. Then, T_c corresponds to the largest interval between two successes (including the index of the first success). The lengths of such intervals are random variables which approximately follow a geometric distribution (ignoring end-horizon effects) with parameter $p = P_{\mathbb{C}}(c_0)$. We define these random variables as: W_j = length of the j^{th} interval. If N_0 is the number of nodes with cost c_0, then given N_0: $T_c = \max_{j=1,\ldots,N_0} W_j$. We make the approximation that the W_j are independent of N_0 and are independent of each other. In that case, for $1 \leq t \leq n$:

$$\mathbb{P}(T_c \leq t | N_0) = \mathbb{P}(\max_{j=1,\ldots,N_0} W_j \leq t) = \prod_{j=1}^{N_0} \mathbb{P}(W_j \leq t) \approx \prod_{j=1}^{N_0} \sum_{k=1}^{\lfloor t \rfloor} p(1-p)^{k-1} = p^{N_0} (\sum_{k=1}^{\lfloor t \rfloor} (1-p)^{k-1})^{N_0}$$

So: $\mathbb{P}(T_c \leq t | N_0) \approx p^{N_0} \left(\frac{1-(1-p)^t}{p} \right)^{N_0} = [1-(1-p)^t]^{N_0}$

Therefore:

$$\mathbb{E}[T_c | N_0] \approx \sum_{t=1}^{n} (1 - \mathbb{P}(T_c \leq t)) = n - \sum_{t=1}^{n} (1-(1-p)^t)^{N_0}$$

Finally, using the Tower Property $\mathbb{E}[T_c] = \mathbb{E}[\mathbb{E}[T_c|N_0]]$, and using the fact that $N_0 \sim Bin(n,p)$ truncated zero (we assumed that there was at least one node with cost c_0), we conclude that:

$$\mathbb{E}[T_c] \approx n - \sum_{n_0=1}^{n} \frac{\binom{n}{n_0}}{1-(1-p)^n} p^{n_0}(1-p)^{n-n_0} \sum_{t=1}^{n} (1-(1-p)^t)^{n_0}$$

This double sum is hard to evaluate. We will now derive a shorter formula by first calculating the cumulative distribution function of T_c:

$$\mathbb{P}(T_c \leq t) = \sum_{n_0=1}^{n} \mathbb{P}(T_c \leq t | N_0 = n_0)\mathbb{P}(N_0 = n_0) \approx \sum_{n_0=1}^{n} [1-(1-p)^t]^{n_0} \frac{\binom{n}{n_0} p^{n_0}(1-p)^{n-n_0}}{1-(1-p)^n},$$

so

$$\mathbb{P}(T_c \leq t) \approx \begin{cases} \frac{\left(1-p(1-p)^t\right)^n - (1-p)^n}{1-(1-p)^n}, & \text{if } t < n, \\ 1, & \text{if } t \geq n. \end{cases} \tag{4.7}$$

and

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{n-1} \left[1 - \frac{\left(1-p(1-p)^t\right)^n - (1-p)^n}{1-(1-p)^n} \right] = \sum_{t=1}^{n-1} \frac{1-\left(1-p(1-p)^t\right)^n}{1-(1-p)^n}.$$

Remark. Recall, from the Descendant Algorithm, that if $j_1, \ldots, j_{k^*}$ are the nodes in the path we have to influence, then the number N_k of consecutive times we need to influence a node j_k:

$$N_k = \min(j_k - j_{k-1}, j_{k+1} - j_{k-1} - T + 1), \forall 1 \leq k \leq k^*.$$

Therefore if the costs of the nodes are generated independently from a PMF $P_{\mathbb{C}}(c)$, the N_k can be approximated by the random variables W_k in the proof above, and will thus approximately have a geometric distribution with mean $1/P_{\mathbb{C}}(c_0)$.

We can also approximate the expression obtained in the Approximate Calculation 4.1.16, for large values of n. Indeed, we have shown that $\mathbb{E}[T_c|N_0] \approx n - \sum_{t=1}^{n}(1-(1-p)^t)^{N_0}$. In the limit where n is large, the strong law of large number tells us that N_0/n converges to p almost surely. Hence we can approximate N_0 by np for large values of n. Thus, for large

values of n:

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{N} \left[1 - (1 - (1-p)^t)^{Np}\right] \tag{4.8}$$

Example 4.1.17. We apply the previous proposition to Figure4-7: here, $n = 1000$, $\mathbb{C} \sim \mathrm{U}\{1, 10\}$, therefore $\mathbb{E}[\mathbb{C}] = 5.5$ and $p = 0.1$. So:

$$\mathbb{E}[\widehat{\mathbb{C}}(T=1)] = n\mathbb{E}[\mathbb{C}] = 5500$$

$$\mathbb{E}[\widehat{\mathbb{C}}(T=n)] = \mathbb{E}[\mathbb{C}_1] = 5.5$$

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{n-1} \frac{1 - \left(1 - p(1-p)^t\right)^n}{1 - (1-p)^n}$$

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{1000-1} \frac{1 - \left(1 - 0.1(1-0.1)^t\right)^{1000}}{1 - (1-0.1)^{1000}} \approx 48.6919$$

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{n} \left[1 - (1 - (1-p)^t)^{np}\right]$$

$$\mathbb{E}[T_c] \approx \sum_{t=1}^{1000} \left[1 - (1 - (1-0.1)^t)^{1000 \times 0.1}\right] \approx 48.7345$$

Note that for $n = 10$ (Figure 4-6) we get: $\mathbb{E}[T_c] \approx 6.3732$ which agrees with the sample path shown (the linear behavior starts at $T = 5$ in Figure 4-6).

The formula derived in Equation (4.8) is not very practical for approximating $\mathbb{E}[T_c]$ when n gets large. We want to understand how fast this function grows with n. In the following, we show that $\mathbb{E}[T_c]$ grows as $O(\log n)$.

For simplicity, let $q = 1-p$: then $n - \mathbb{E}[T_c] \approx \sum_{t=1}^{n}(1-q^t)^{np}$. If $q^t \ll \frac{1}{np}$ (so $t \gg -\frac{\log(np)}{\log(q)}$), then, for $t_0 \gg -\frac{\log(np)}{\log(q)}$ (and assuming $np > 1$), we obtain

$$\sum_{t=1}^{n}(1-q^t)^{np} = \sum_{t=1}^{t_0}(1-q^t)^{np} + \sum_{t=t_0+1}^{n}(1-q^t)^{np} \approx \sum_{t=1}^{t_0}(1-q^t)^{np} + \sum_{t=t_0+1}^{n}(1-npq^t).$$

Thus,

$$\mathbb{E}[T_c] \approx n - \sum_{t=1}^{t_0}(1-q^t)^{np} - \sum_{t=t_0+1}^{n}(1-npq^t) = -\sum_{t=1}^{t_0}(1-q^t)^{np} + t_0 + np\sum_{t=t_0+1}^{n} q^t.$$

Hence,

$$\mathbb{E}[T_c] \approx -\sum_{t=1}^{t_0}(1-q^t)^{np} + npq^{t_0+1}\frac{1-q^{n-t_0}}{1-q} + t_0.$$

Now,

$$\sum_{t=1}^{t_0}(1-q^1)^{np} < \sum_{t=1}^{t_0}(1-q^t)^{np} < \sum_{t=1}^{t_0}(1-q^{t_0})^{np},$$

so that

$$t_0 e^{np\log(1-q)} \approx t_0(1-q)^{np} < \sum_{t=1}^{t_0}(1-q^t)^{np} < t_0(1-q^{t_0})^{np} \approx t_0 e^{-np}$$

is negligible when n is large. Hence,

$$\mathbb{E}[T_c] \approx npq^{t_0+1}\frac{1-q^{n-t_0}}{1-q} + t_0 = npq^{t_0}\frac{q}{1-q} - npq^{n+1}\frac{1}{1-q} + t_0 \approx npq^{t_0}\frac{q}{1-q} + t_0,$$

which gives

$$\mathbb{E}[T_c] \approx npq^{t_0}\frac{1-p}{p} + t_0 = O(1)\frac{1-p}{p} - \frac{\log(np)}{\log(1-p)}.$$

Therefore $\mathbb{E}[T_c]$ scales as $\log(n)$ for large values of n. This means that, for large values of n, if we are willing to pay a cost of $O(n)$ we will be able to influence the whole line in time $O(\log(n))$, while if we wish to incur a cost of $o(n)$ we will be able to influence the whole line in time $O(n)$.

4.2 Analysis for a General Network

In this section, we generalize the results obtained for Line Networks to general networks. We will use the structure of general in-degree one networks to show that the Descendant Algorithm can still be applied to obtain an optimal policy to our optimization problem. We will consider both finite and infinite time-horizon problems. We will first assume that the controller wants to influence every node in the network by time T, and then we will relax this condition by assuming the controller only wants to influence a fraction $\eta \in (0, 1)$ of the nodes by time T. Unless otherwise specified, all networks are assumed to be in-degree one networks throughout this section.

4.2.1 Properties of the Graph Representation

We start by analyzing the structure of general in-degree one networks. We will show that such networks are composed of only two possible types of connected components. We first define:

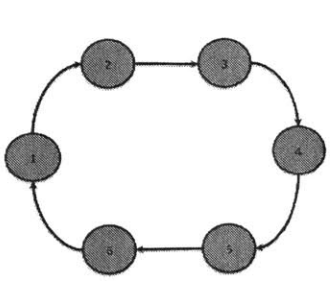
Definition 4.2.1. A **k-Cycle** is a cycle of k nodes $(i_1, \ldots, i_k)$ in which each node influences exactly one of his neighbors, and all edges are oriented in the same direction (Figure 4-10a). We will refer to it as the k-cycle k-$C(i_1, \ldots, i_k)$. For convenience, we assume node i_1 to have the least cost ($c_{i_1} \leq c_{i_j}$ for all $j \in \{2, \ldots, k\}$).

Definition 4.2.2. A **Tree Chain** is a connected set of nodes that contains no cycles, but which contains exactly one self-influencing node i (Figure 4-10c). We will refer to it as the Tree Chain $TC(i)$.

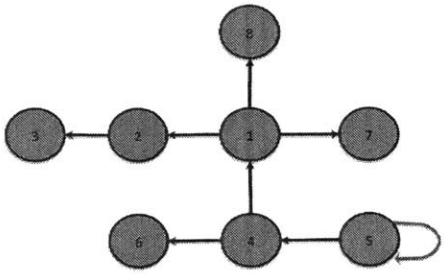
Definition 4.2.3. A **Mixed Chain** is a connected set of nodes that contains a unique cycle of some finite length k: k-$C(i_1, \ldots, i_k)$ (Figure 4-10d). We will refer to it as the mixed chain $MC(i_1, \ldots, i_k)$. For convenience, we assume node i_1 to have the least cost along the cycle ($c_{i_1} \leq c_{i_j}$ for all $j \in \{2, \ldots, k\}$).

Remark. Note that a k-cycle is a special case of a mixed chain.

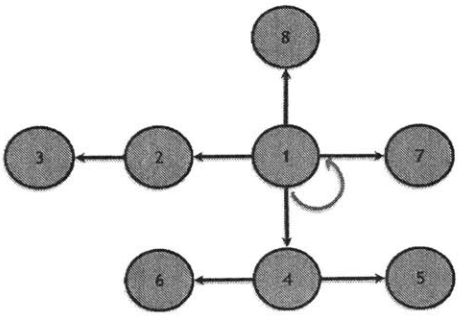
Theorem 4.2.4. *For a general in-degree one network, a connected component is either:*



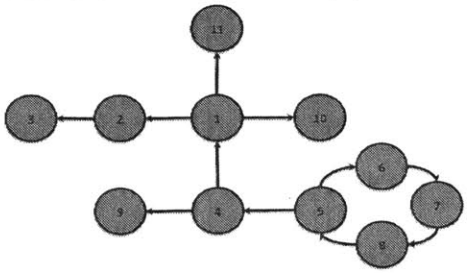
(a) 6-Cycle 6-$C(1,2,3,4,5,6)$.



(b) Tree Chain with 8 nodes $TC(5)$.



(c) Another Tree Chain with 8 nodes $TC(1)$.



(d) Mixed Chain with 11 nodes and a 4-cycle $MC(5,6,7,8)$. The cycle could also be found inside the network (e.g., we can replace node 1 in (c) by a cycle).

Figure 4-10: Types of connected components encountered in the simplified graph of a general network.

1. *a tree chain $TC(i)$, or*

2. *a mixed chain $MC(i_1, \ldots, i_k)$ (which could be a k-cycle k-$C(i_1, \ldots, i_k)$).*

To prove this theorem, we note the following property of the network:

Lemma 4.2.5. *If all nodes of a network have in-degree one, then there must be as many edges as there are nodes.*

Proof. Suppose there are n nodes in the network. Since nodes have in-degree one, we must have one incoming edge per node, hence n edges in total. □

Lemma 4.2.6. *A connected set of k nodes either has a unique cycle (of some finite length k) or a unique node influencing itself.*

Proof. By Lemma 4.2.5, we must have exactly k edges connecting these k nodes. To connect k nodes, we need only $k-1$ edges. However, if we only have $k-1$ edges for k nodes, there must be a node i with in-degree zero. The additional edge can either be added from this node i to itself, or from some node j to this node i. In the former case, we get the unique self-influencing node i. In the latter case, we claim that nodes i and j will be part of a k-cycle:

Because the network with $k-1$ edges is already connected, and node i is the only node with in-degree zero, we get a tree with root i. This implies that all edges flow away from i in the same direction (see Prop. 4.2.7 below). Therefore there must exist a path $(i_1, i_2, \ldots, i_k)$ from node i to node j where $i_1 = i$, $i_k = j$. By adding the edge from node j to node i, we create a k-cycle. □

We can now return to the proof of Theorem 4.2.4:

Proof. (Theorem 4.2.4) From the previous lemmata, we have shown that a connected set of k nodes must have either a unique cycle, or a unique self-influencing node. If this connected set of nodes has a unique self-influencing node, the graph is a Tree Chain. If this connected set of nodes has a unique cycle, the graph is a Mixed Chain. □

We can now generalize the Cascade Property to general in-degree one networks. This property will be very useful in constructing an optimal policy.

Proposition 4.2.7. (The Cascade Property) *If we remove the self-influencing node (resp. k-cycle) of a Tree Chain (resp. Mixed Chain), we obtain a collection of disjoint chains. In each chain, all arrows point in the same direction. In other words, edges either go away from a self-influencing node or from a k-cycle.*

Proof. If an edge is going towards a self-influencing node, then this node has in-degree two which is forbidden. If an edge goes towards a k-cycle, then one node of the k-cycle has in-degree two which is forbidden. □

We now have a complete characterization of the connected components we can encounter in general in-degree-one networks. This will immediately provide an optimal policy for our control problem over such networks, when $T = \infty$ and $\eta = 1$.

Infinite Time Horizon Case ($T = \infty$) With $\eta = 1$: We start with the case of an infinite time horizon ($T = \infty$) where $\eta = 1$ (i.e., we want to fully influence the network). This means that the controller's objective is to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. We assume that $X(1) = (0, \ldots, 0)$. The policy when $X(1) = (X_1(1), \ldots, X_n(1))$ follows from the theorem below and is described in the remark below.

Proposition 4.2.8. *If $T = \infty$ and $\eta = 1$:*

1. *Suppose the controller wants to fully influence a k-cycle $k\text{-}C(i_1, \ldots, i_k)$ (with cheapest node i_1). An optimal policy is to influence node i_1 at times $t = 1, 2, \ldots, k$ (i.e., choose $\alpha_{i_1}(1) = \ldots = \alpha_{i_1}(k) = 1$, and set all other components of $\alpha(t)$ to zero). The total cost of influence for this connected component will be: kc_{i_1}. The whole component will be entirely in the $+1$ state at time $t = k + 1$.*

2. *Suppose the controller wants to fully influence a tree chain $TC(i)$. An optimal policy is to influence node i at time $t = 1$ (i.e., choose $\alpha_i(1) = 1$ and set all other components of $\alpha(t)$ to zero). The total cost of influence for this connected component will be: c_i.*

The whole component will be entirely in the $+1$ state at time $t = q + 1$, where q is the number of nodes in the longest path starting at node i.

3. *Suppose the controller wants to fully influence a mixed chain $MC(i_1, \ldots, i_k)$ (with cheapest node i_1 in the cycle). An optimal policy is to influence the node i_1 at times $t = 1, 2, \ldots, k$ (i.e., choose $\alpha_{i_1}(1) = \ldots = \alpha_{i_1}(k) = 1$, and set all other components of $\alpha(t)$ to zero). The total cost of influence for this connected component will be: kc_{i_1}. The whole component will be entirely in the $+1$ state at time $t = q + 1$, where q is the number of nodes in the longest path (or cycle) starting at node i_1.*

In general, the optimal cost for fully influencing the whole network is given by:

$$\sum_{TC(i)} c_i + \sum_{MC(i_1,\ldots,i_k)} kc_{i_1} \tag{4.9}$$

and the time needed to influence the whole network is equal to:

$$T^* = \max_{\substack{\text{all connected} \\ \text{components}}} \begin{cases} k+1, & \text{if in a } k\text{-cycle } k\text{-}C(i_1, \ldots, i_k), \\ 1+\text{number of nodes in the longest path starting at node } i, & \text{if in a Tree Chain } LC(i), \\ 1+\text{number of nodes in the longest path or cycle starting at node } i_1, & \text{if in a Mixed Chain } MC(i_1, \ldots, i_k). \end{cases} \tag{4.10}$$

Proof. For simplicity, we assume that $X(1) = (0, \ldots, 0)$.

1. Suppose the controller wants to fully influence a k-cycle k-$C(i_1, \ldots, i_k)$. We are considering an infinite time-horizon problem, therefore the controller wants to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. Notice that if the controller only influences one node in the cycle, only one node will be in the $+1$ state at each time. Therefore the controller must perform k influences in total. Clearly, influencing the cheapest node k times will be an optimal solution. Thus, an optimal policy is to influence node i_1 at times $t = 1, \ldots, k$ (i.e., choose $\alpha_{i_1}(1) = \ldots = \alpha_{i_1}(k) = 1$ and set all other components of $\alpha(t)$ to zero).

2. Suppose the controller wants to fully influence a tree chain $TC(i)$. We are considering

an infinite time-horizon problem, therefore the controller wants to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. For node i to reach a $+1$ state, the controller will need to influence node i at some finite time, because node i is only influenced by itself. Once the controller does that, node i will remain indefinitely in the $+1$ sate. Therefore the optimal cost of the control problem is at least equal to c_i. By the Cascade Property (Proposition. 4.2.7), we see that node i will spontaneously influence every path starting at node i in a finite amount of steps. This cascade effect does not require any action from the controller. Therefore this policy has a cost equal to c_i, and our policy must be an optimal policy.

3. Suppose the controller wants to fully influence a mixed chain $MC(i_1, \ldots, i_k)$. We are considering an infinite time-horizon problem, therefore the controller wants to reach the steady state $X(t_0) = (1, \ldots, 1)$ at a finite (but arbitrarily large) time t_0. First, we view the cycle $(i_1, \ldots, i_k)$ as a unique self-influencing node. By part (2), we see that the controller must influence the cycle directly in order to reach the desired steady state. Observe that the whole mixed chain will be in the $+1$ state if and only if the full cycle is in the $+1$ state (by the Cascade Property, Prop. 4.2.7). We now use part (1) to influence the cycle at an optimal cost.

□

For an arbitrary initial state $X(1)$, we need to change our policy only if the self-influencing node, or nodes of the k-cycle are already in the $+1$ state at time $t = 1$. The policies described in the following remark are clearly optimal.

Remark. For an arbitrary initial state $X(1) = (X_1(1), \ldots, X_n(1))$, we can obtain optimal policies in a similar way. For a k-cycle $k\text{-}C(i_1, \ldots, i_k)$ that starts with K nodes in the $+1$ state, we influence node i_1 $(k - K)$ times when node i_1 is in a 0 state: this policy reaches the optimal cost $(k-K)c_{i_1}$. Similarly, for a Mixed Chain $MC(i_1, \ldots, i_k)$ whose cycle starts with K nodes in the $+1$ state, we influence node i_1 $(k - K)$ times when node i_1 is in a 0 state: this policy reaches the optimal cost $(k - K)c_{i_1}$. For a Tree Chain $TC(i)$, if node i starts in the $+1$ state, do nothing; if node i starts in the 0 state, influence node i at $t = 1$ at a cost c_i.

We can now use the previous proposition to solve the infinite time-horizon control problem when $0 < \eta < 1$.

Infinite Time Horizon Case ($T = \infty$) With $0 < \eta < 1$:

Suppose that, in our infinite time-horizon problem, the controller only wants to influence a fraction of the nodes from the network (i.e., $0 < \eta < 1$). We still need to reach a recurrent state in a finite (but arbitrarily large) amount of time. The network is a disjoint collection of tree chains and mixed chains. Since the controller does not need to get all nodes in the $+1$ state, we simply need to influence the cheapest connected components of the network in order to get at least ηn nodes: this is clearly a knapsack problem. For simplicity, assume that $X(1) = \Big(0, \ldots, 0\Big)$.

Proposition 4.2.9. *For simplicity, assume that* $X(1) = \Big(0, \ldots, 0\Big)$. *If* $T = \infty$ *and* $0 < \eta < 1$*:*

1. *Suppose we have a k-cycle* $k\text{-}C(i_1, \ldots, i_k)$ *(with* i_1 *being the cheapest node): to have* m *nodes in the* $+1$ *state, an optimal policy will be to influence node* i_1 m *consecutive times (i.e.,* $\alpha_{i_1}(1) = \ldots = \alpha_{i_1}(m) = 1$ *and set all other components of* $\alpha(t)$ *to zero). This is done at a cost* mc_{i_1} *by time* $t = m + 1$.

2. *Suppose we have a tree chain* $TC(i)$*:*

   Since $T = \infty$*, the controller must influence the node* i *at time* $t = 1$ *(i.e.,* $\alpha_i(1) = 1$ *and set all other components of* $\alpha(t)$ *to zero). Note that this procedure will influence all nodes in the tree chain by the Cascade Property (Proposition 4.2.7). This is done at a cost* c_i*. The whole tree chain will be entirely in the* $+1$ *state by time* $t = q + 1$*, where* q *is the number of nodes in the largest path starting at node* i.

Proof. This proposition clearly follows from Proposition 4.2.8. □

For Mixed Chains of n nodes, when we directly influence m nodes of the cycle, we eventually reach a recurrent state. If the cycle of the Mixed Chain has k nodes, then there will be k recurrent states occurring periodically. We show below that, if we take a time-

average of the number of nodes in the +1 state after reaching a recurrent state, we will have "on average" $n\frac{m}{k}$ nodes in the +1 state in the Mixed Chain.

Proposition 4.2.10. *Suppose we have a mixed chain $MC(i_1, \ldots, i_k)$ of n nodes (with $c_{i_1} \leq c_{i_j}$ for all $j \in \{2, \ldots, k\}$).*

1. *To influence "on average" n/r nodes (for some $s \in \{1, \ldots, n\}$), we can influence node i_1 at $m = \lfloor k/r \rfloor$ consecutive times with a value $\alpha_{i_1}(1) = \ldots = \alpha_{i_1}(m) = 1$, at a cost mc_{i_1}, by time $t = q + 1$ (where q is the highest number of nodes on either side of node i).*

2. *To influence all nodes, we can influence node i_1 at k consecutive times, at cost kc_{i_1}, by time $t = q + 1$ (where q is the number of nodes in the longest path starting at node i).*

Proof. 1. Suppose the Mixed Chain has n nodes, and that we directly influence m nodes in the cycle of length k. Now consider a "branch" of the Mixed Chain: we define a branch as a path of the Mixed Chain that starts from a node of the cycle, and reaches an out-degree zero node of the network without passing by a node of the cycle. Consider a branch of b nodes.

Now draw a configuration of the cycle with m nodes in the +1 state, and $k - m$ nodes in the 0 state. Start from node i_1 in the cycle and count the number of nodes in the +1 state among the set $\{i_1, \ldots, i_b\}$. Go to node i_2 and count the number of nodes in the +1 state among the set $\{i_2, \ldots, i_{b+1}\}$. Continue in that way until you reach node i_k in the cycle, and count the number of nodes in the +1 state among the set $\{i_k, \ldots, i_{b-1}\}$. These sets of b nodes correspond to all the configurations of the branch we can observe in k successive time steps.

Thus in k time steps, we will have counted mb nodes in the +1 state. Therefore, on average, a branch of length b will have mb/k nodes in the +1 state. So we conclude that, on average, the network will have mn/k nodes in the +1 state.

Therefore, for any configuration of the cycle with m nodes (out of k) in the +1 state, we have, on average, mn/k nodes in the +1 state in the network. This means that

by influencing m nodes in the cycle, we get, on average, mn/k nodes in the $+1$ state. The cheapest way to influence m nodes in the cycle is to influence the cheapest node of the cycle m times. Hence the policy proposed here is optimal.

2. This proposition clearly follows from Proposition 4.2.8.

□

Proposition 4.2.11. *We are given a network consisting of a collection of Tree Chains and Mixed Chains. To eventually reach a fraction η of nodes in the $+1$ state, we can follow the procedure outlined in Proposition 4.2.9 and Proposition 4.2.10, and choose the connected components to influence by solving the following knapsack problem.*

$$\text{Let } \alpha_i(t) = \begin{cases} 1, & \text{if we influence node } i \text{ at time } t, \\ 0, & \text{if we don't influence node } i \text{ at time } t. \end{cases}$$

We solve:

$$\begin{aligned} \textit{minimize} \quad & \sum_{TC(i)} \alpha_i(1)c_i + \sum_{MC(i_1,\ldots,i_k)} \Big(\alpha_{i_1}(1) + \ldots + \alpha_{i_1}(k)\Big)c_{i_1} \\ \textit{subject to} \quad & \eta n \leq \sum_{TC(i)} \alpha_i(1)|TC(i)| + \sum_{MC(i_1,\ldots,i_k)} \frac{|MC(i_1,\ldots,i_k)|}{r}\Big(\alpha_{i_1}(1) + \ldots + \alpha_{i_1}(\lfloor k/r \rfloor)\Big), \\ & \alpha_i(t) \in \{0,1\}, \forall i = 1 \ldots n, \forall t = 1, \ldots, k \end{aligned} \tag{4.11}$$

where the sums run over all Tree Chains and all Mixed Chains in the network.

Note that the last estimate of the number of nodes in the mixed chains is approximate (it gives the average number of nodes that will be in the $+1$ state for this mixed chain).

Proof. This proposition clearly follows from Proposition 4.2.9 and Proposition 4.2.10. □

Equation (4.11) involves complicated notation, but the idea behind it is simple: we want to select the cheapest connected components and decide how many nodes we need to influence in each case. This basically boils down to a knapsack problem.

4.2.2 Applications of The Descendant Algortithm

We will now consider finite time-horizon problems, first assuming that $\eta = 1$, and then for a general $\eta \in (0,1)$. In the former case, we will solve the control problem by applying the

Descendant Algorithm (DA). We will also show that the DA can be extended to infinite time-horizon problems. For simplicity, we will assume $X(1) = (0, \ldots, 0)$ in the entire section.

Finite Time Horizon ($T < \infty$) and $\eta = 1$

Note that if T is finite, the proposition still holds when $T^* \leq T$ (where T^* is defined in Equation (4.10) as the smallest time to reach the targeted state under an optimal policy). If $T^* > T$, then the process must be accelerated by influencing more nodes. This way, the entire network will be in the $+1$ state at time T. The following proposition provides an optimal policy:

Proposition 4.2.12. *If $T < \infty$ and $\eta = 1$:*

1. *Suppose the controller wants to fully influence a k-cycle k-$C(i_1, \ldots, i_k)$ (where i_1 is the cheapest node of the cycle). In this case, $T^* = k + 1$. Hence, if $T \geq k + 1$, we just need to apply the optimal policy provided for the infinite time-horizon case. If $T \leq k$, then the controller can influence node i_1 at times $t = 1, \ldots, T - 1$, delete the arrow from i_k to i_1 and apply the Descendant Algorithm on the path $(i_1, \ldots, i_k)$, viewing i_1 as the self-influencing node.*

2. *Suppose the controller wants to fully influence a tree chain $TC(i)$. The controller will apply the Descendant Algorithm to all the paths of $TC(i)$ going from node i to an end of the tree chain (i.e., a node j with out-degree equal to zero). If a node belongs to different paths, the algorithm will tell us how many times we should influence this node for each path considered: we must therefore take the maximum of these values to get the number of times we need to influence this node.*

3. *Suppose the controller wants to fully influence a mixed chain $MC(i_1, \ldots, i_k)$ (where i_1 is the cheapest node of the cycle). In this case, $T^* = k + 1$. Hence, if $T \geq k + 1$, we just need to apply the optimal policy described in the infinite time-horizon case. If $T \leq k$, then the controller can influence node i_1 at times $t = 1, \ldots, T - 1$, delete the*

arrow from i_k to i_1 and apply the Descendant Algorithm on all paths going from i_1 to an end of the mixed chain (i.e., a node j with out-degree equal to zero) viewing i_1 as the self-influencing node. If a node belongs to different paths, the algorithm will tell us how many times we should influence the node for each path considered: we must therefore take the maximum of these values to get the number of times we need to influence this node.

Proof. This proposition follows from the proof of the Descendant Algorithm on a Path. □

Remark. We observe that the Descendant Algorithm can also be applied to this control problem when $T = \infty$.

Finite Time Horizon Case ($T < \infty$) With $0 < \eta < 1$:

If T is finite and $0 < \eta < 1$, the solution is much trickier. Since we do not need to influence the entire network, and the time-horizon is finite, then we may not want to influence directly the self-influencing nodes or nodes in cycles. Things can go very wrong as shown in the case of the line $LC(2)$ in Example 4.1.15. Optimal policies for finite T and $0 < \eta < 1$ would consist of influencing "cheap" nodes at times "close" to T. The formalism we will develop in the next chapter for the Ancestral Algorithm will allow us to solve this problem in a much simpler way. We thus defer the analysis of this case to the next chapter.

Chapter 5

The Dynamic Deterministic Model (DDM)

In this chapter, we will solve our optimization problem for the case of the Dynamic Deterministic Model (DDM). Our approach here will differ from the one we used earlier for the Static Deterministic Problem (SDP). Previously, we were looking at a node and checking which nodes it can influence and at what price: we referred to this method as the "descendant method". This approach is intuitive and easy to implement, however it is much harder to generalize to the DD problem. Furthermore, the Descendant Method requires more work when some nodes in the system are already influenced at time zero.

Recall that the Dynamic Deterministic Problem (DDP) is based on the following time-dependent matrices: the matrix of Influencers $I = [I(t,i)]$ (for $t \in \{1,\ldots,T\}$ and $i \in \{1,\ldots,n\}$) summarizing the dynamics of the network, and the matrix of Costs $C = [c(t,i)]$ (for $t \in \{1,\ldots,T\}$ and $i \in \{1,\ldots,n\}$), corresponding to the cost the controller would incur if it decides to influence a particular node at a given time. The control vector $\alpha(t) = \Big(\alpha_1(t),\ldots,\alpha_i(t)\Big) \in \{0,1\}^n$ has a positive i^{th} component when the controller decides to influence node i at time t. Finally, the state of a node i at time t is given by:

$$X_i(t+1) = \begin{cases} X_{I(t,i)}(t), & \text{if } \alpha_i(t) = 0 \text{ (with cost } = 0)\ , \\ 1, & \text{if } \alpha_i(t) = 1 \text{ (with cost } = c_i(t))\ . \end{cases}$$

To solve the DD optimization problem, we work backwards in time: we will look at a node and check its genealogy, i.e., his influencer one time-step earlier, the influencer of his influencer two time-steps earlier, ... Once we create this ancestral path, we will check which node is the cheapest possible influencer and decide if it is convenient to influence it directly. We will refer to this method as the "ancestral method". Building the ancestral path may appear less intuitive at first as it requires us to think about time and space simultaneously (i.e., who is the influencing node t time-steps earlier), and we must look backwards in time. However, once we have it, we can solve the SDP as easily as the DDP. Furthermore, we will show that the ancestral method is easily applied to the case in which some nodes are already influenced at time zero.

However, we will note that the ancestral method requires, at time $t = 1$, the full knowledge of the evolution of the system. The descendant method approach would be better suited for heuristic policies when there are uncertainties in the future state of the nodes (for example, we could think of a model in which the edges in the network can be switched on and off with probability 1/2).

We will mostly consider finite time-horizon problems and will show that any such problem can be solved using the Ancestral Algorithm. In the case of an infinite time-horizon, we will be able to solve any problem in which the graph dynamics reach a recurrent state at some finite time T_0. Infinite time-horizon problem are not very useful if they do not behave in a tractable way: it would be more suited to work with stochastic models in that case.

To simplify the presentation, we will assume that the last time at which the controller is allowed to act is time $t = T$ (not $t = T - 1$), i.e., that the controller wants to reach its target state by time $t = T + 1$.

In this chapter, we begin in Section 5.1 by presenting the formalism used to set up the Ancestral Algorithm and we describe the algorithm. In Section 5.2, we apply the Ancestral Algorithm to the Static Deterministic System. In Section 5.3, we apply the Ancestral Algorithm to the Dynamic Deterministic System when we want to have at least η nodes (with $0 < \eta < 1$) in state $+1$ by time T. Finally, in Section 5.4, we illustrate through an example the theory developed throughout the chapter.

5.1 The Ancestral Path, the Ancestral Network, and the Ancestral Algorithm

Our goal will be, given an initial state $X(1)$ of the system, to obtain a state $X(T) = (1, \ldots, 1)$ at minimal cost. This will be accomplished by the Ancestral Algorithm. To develop this algorithm, we first construct the Ancestral Network. We will start by defining the Ancestral Path. Recall that $I(t,i)$ is the unique node that will influence node i at time t.

Definition 5.1.1. We define the a^{th} **ancestor** $I^a(t,i)$ **of a node** i **at time** t as the node whose state at time $t-a$ affects the state of node i at time t. Formally:

$I^a(t,i) = I\Big(t-a+1, I(t-a, I(t-a-1, I(\ldots I(t,i))))\Big)$

or recursively:

$I^a(t,i) = I(t-a+1, I^{a-1}(t,i))$, where $I^0(t,i) = i$ and $I^1(t,i) = I(t,i)$

Example 5.1.2. We see that:

$I^1(t,i) = I(t-1+1, I^{1-1}(t,i)) = I(t,i),$

$I^2(t,i) = I(t-2+1, I^{2-1}(t,i)) = I(t-1, I(t,i))$

$I^3(t,i) = I(t-3+1, I^{3-1}(t,i)) = I(t-2, I^2(t,i)) = I(t-2, I(t-1, I(t,i)))$

The notation may seem involved, but the intuition is very simple: if I influence node $I^a(t,i)$ at time $t-a$, then I know that node i will be influenced at time t without any other intervention.

Definition 5.1.3. The **Ancestral Path of a node** i **with respect to time** t consists of the sequence of ancestors: $A(t,i) = (i, I(t,i), I^2(t,i), \ldots, I^{T-1}(t,i))$.

Since we want to get our nodes influenced by time T, we will mostly focus on the sequence $A(T,i)$.

Definition 5.1.4. The **Ancestral Path of a node** i consists of the sequence of ancestors: $A(i) := A(T,i) = (i, I(T,i), I^2(T,i), \ldots, I^{T-1}(T,i))$.

The Ancestral Path of a node is well defined because nodes have in-degrees equal to one.

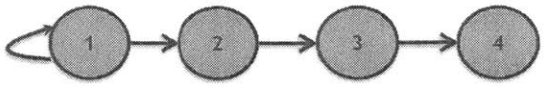


Figure 5-1: Network used in Ex. 5.1.5: we are here describing an SDM, and we assume $T = 5$.

Example 5.1.5. Consider the following SDP: suppose that our network dynamics are constant in time, and consider a path $(1, 2, 3, 4)$ with $T = 5$ (see Fig. 5-1). We assume node 1 to be the self-influencing node. Then:

$A(1) = (1, 1, 1, 1, 1)$

$A(2) = (2, 1, 1, 1, 1)$

$A(3) = (3, 2, 1, 1, 1)$

$A(4) = (4, 3, 2, 1, 1)$

We will later represent these Ancestral Paths in the Ancestral Network. The Ancestral Network of Fig. 5-1 is shown in Fig. 5-2.

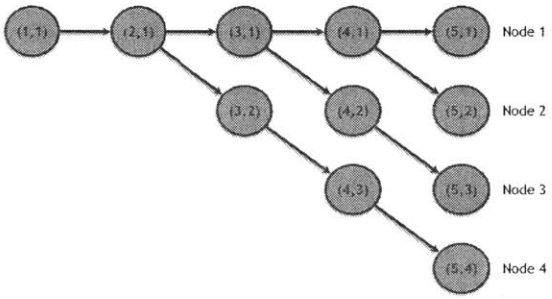


Figure 5-2: Ancestral Network of Fig. 5-1: this is constructed using the Ancestral Paths calculated in Ex. 5.1.5. The first index at each node of the Ancestral Network indicates time, while the second index indicates the corresponding node in the original network.

We can now define the Ancestral Network:

Definition 5.1.6. The **Ancestral Network** is a directed graph with the following properties:

1. Each node in the Ancestral Network represents a pair (t, i) corresponding to node i at time t in the original network. We only include pairs (t, i) that appear in the ancestral path of at least one node of the original network.

2. We include a directed edge from a pair (t, i) to another $(t+1, j)$ if and only if node j's direct ancestor at time $t+1$ is node i.

The Ancestral Network displays the "genealogy" of the nodes in the network. It also indicates nodes that share common ancestors. Some basic properties of the Ancestral Network are derived below.

Proposition 5.1.7. *We have:*

1. *A node of the Ancestral Network has in-degree equal to one.*

2. *The Ancestral Network has no cycles.*

3. *All paths starting at any node (t, i) of the Ancestral Network will have the same length; furthermore, if we start at a node with $t = 1$, any path of the Ancestral Network will have length equal to T.*

In particular, the Ancestral Network is a forest of directed trees.

Proof. .

1. A node i of the original network has a unique ancestor $I(t, i)$ at any time t. Therefore node (t, i) of the Ancestral Network will have a unique incoming edge.

2. Edges of the Ancestral Network always point towards a node of the form (t, i) to a node of the form $(t+1, j)$ (for some node i and j of the original network). Hence, edges of the Ancestral Network point "forward in time". A cycle would require edges going backward in time, which is not possible.

3. Edges of the Ancestral Network point "forward in time", so a node (t, i) will have exactly $(T - t + 1)$ descendants along each possible path starting at (t, i). Therefore, each of these paths will have length $(T - t + 1)$. In particular, paths starting at nodes of the form $(1, i)$ have length T.

□

Notice that when trees in the Ancestral Network are disjoint, their nodes do not share any common ancestors. So, for disjoint trees, we can solve the optimization problem on each ancestral tree independently.

5.1.1 The Ancestral Algorithm

Assume that $\eta = 1$ in this subsection; the case $0 < \eta < 1$ will be treated later. We are given the I matrix and the C matrix. From the I matrix, we construct the Ancestral Paths $A(i)$ of each node i of the network (total run-time $= O(nT)$). We construct the Ancestral Network based on the A matrix, and associate with each node (t, i) its cost $c(i, t)$. In general, the Ancestral Network consists of a forest of disjoint directed trees with nodes of in-degree one (except for the root of the tree). In the Ancestral Network, each tree of the forest is composed of one or multiple paths of length T sourced at a node $S = (t = 1, i_0)$ (where i_0 is a node of the original network).

We can formulate our optimization problem in terms of the Ancestral Network. We want all nodes of the original network to be in the $+1$ state at time T. These nodes correspond to the leaves (nodes of out-degree zero) of the Ancestral Network. To influence any one of these nodes (T, i), we must influence at least one node along the path going from S to (T, i). Therefore we must choose a subset of nodes from the Ancestral Network (a "cover") such that every leaf has at least one ancestor chosen. In this case, we say that the chosen ancestors constitute a subset that **covers** all leaves of the Ancestral Network. The cost of such a cover is equal to the sum of the costs of the nodes in the cover. We want to cover all n leaves of the Ancestral Network by using the cheapest covering subset. This is a set covering problem, but because of the special structure, it admits an efficient solution.

This covering problem is solved by the following dynamic program, which we call the Ancestral Algorithm:

ANCESTRAL ALGORITHM:

Assume $\eta = 1$, and suppose we are already given the Ancestral Network. The following algorithm tells us which nodes of the Ancestral Network we should influence

at the optimal cost.

For all nodes (t,i) of the Ancestral Network, define $\mathscr{D}(t,i)$ as the set of descendants of node (t,i): this is the set of nodes (t',i') such that there is a path from (t,i) to (t',i'). In particular, the set $\{(t+1,i') \in \mathscr{D}(t,i)\}$ corresponds to the immediate descendants of node (t,i) in the Ancestral Network. We also define the **value of a node (t,i) of the Ancestral Network** as :

$$V(T,i) := c(T,i) \tag{5.1}$$

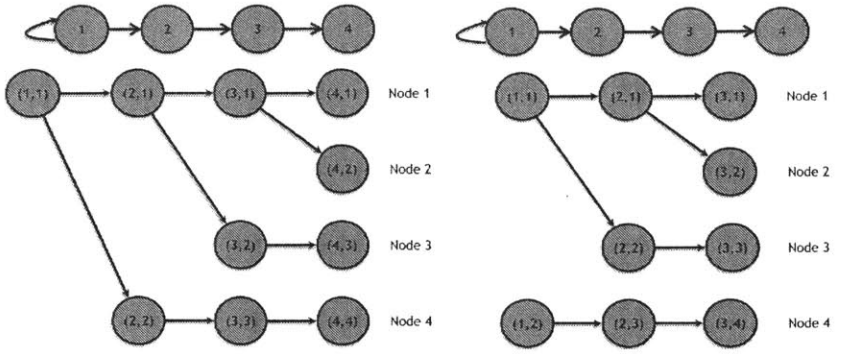
$$V(t,i) := \min\left\{c(t,i)\,,\ \sum_{(t+1,i')\in\mathscr{D}(t,i)} V(t+1,i')\right\}. \tag{5.2}$$

Then the value $V(t,i)$ of node (t,i) is defined as the smallest cost needed to have all its final descendants in the $+1$ state (i.e., at time T). We already know that $V(T,i) = c(T,i)\,, \forall (T,i) \in \mathscr{N}$. Then starting from the leaves of the Ancestral Network, it is possible to calculate $V(t,i)$ for all nodes (t,i) of the Ancestral Network using the recursion in (5.2).

The total run-time of the algorithm is of order $O(nT)$.

5.2 Application of the Ancestral Algorithm to the Static Deterministic System

For Linear Chains $LC(1)$ in the Static Deterministic System (for an example, see Fig. 5-3a), then, in the Ancestral Network, there always exist a tree with a path of the form $((1,1),(2,1),\ldots,(T,1))$. Note that $S=(1,1)$ will be the source of this directed tree of the Ancestral Network. Since we consider the static case, the original network does not evolve with time, therefore $c(t,i) = c_i$ is constant in t. Note that $V(t,1) = \min\{c_1, V(t+1,1) + V(t+1,2)\}$ and $V(T,1) = c_1$. This implies that $V(t,1) = c_1$ for all $t \geq 1$, therefore it is optimal to influence node 1 at time $t=1$. When $T < n$ (for an example, see Fig. 5-3b), the Ancestral Network will also contain disjoint paths which are also disjoint from the tree



(a) Ancestral Network when $T = 4$. (b) Ancestral Network when $T = 3$.

Figure 5-3: We consider here a Linear Chain of 4 nodes. We plot the Ancestral Network of the Linear Chain for different time horizons.

of source S. Over these paths, the Ancestral Algorithm simply picks the cheapest node.

For k-cycles (for an example, see Fig. 5-4), the costs are constant, so we pick the cheapest one (call it node $i = 1$) and use it as the source $S = (1, 1)$. Assume that $T \geq k$. The Ancestral Network of a k-cycle will look like this: $((1,1),(2,2),\ldots,(k,k),(k+1,1),\ldots,(2k,k),(2k+1,1),\ldots,(T,.))$ for the an ancestral tree (not necessarily the one corresponding to node 1), $((1,2),(2,3),\ldots,(k,1),(k+1,2),\ldots,(2k,1),(2k+1,1),\ldots,(T,.))$ for another ancestral tree, ... until a k-th ancestral tree. Hence the cheapest nodes will always be $(1,1)$ in the first tree, $(k,1)$ in the second tree, $(k-1,1)$ in the third tree, ... and $(2,1)$ in the k-th tree. These are exactly the nodes to influence at the given times, i.e., we must influence node $i = 1$ at times $t = 1, 2, \ldots, k$.

For k-cycles, if $T < k$ (for an example, see Fig. 5-5), we cannot always access the cheapest node of the cycle, so we must pick the cheapest accessible node. All accessible nodes are shown in the Ancestral Network.

Case of infinite time-horizon $T = \infty$: in the Static Deterministic System, we eventually reach a recurrent state, so truncate the time-horizon to $T = n$. Note that for our general deterministic system, if the system reaches a recurrent state at some time T_0, we can truncate our time-horizon to $T = T_0 + n$. Actually we can truncate the time even more by taking

$$T = (T_0 + \textit{number of nodes in the longest path from source in Ancestral Network}).$$

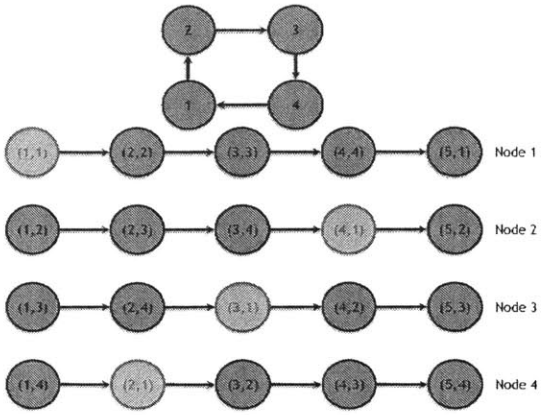


Figure 5-4: A 4-cycle network ($T = 5$) with its Ancestral Network: note that the Ancestral Network consists of four disjoint paths as expected from the periodicity of the original network. We labeled by node 1 the cheapest node of the 4-cycle. The red nodes in the Ancestral Network correspond to the ones picked by the Ancestral Algorithm.

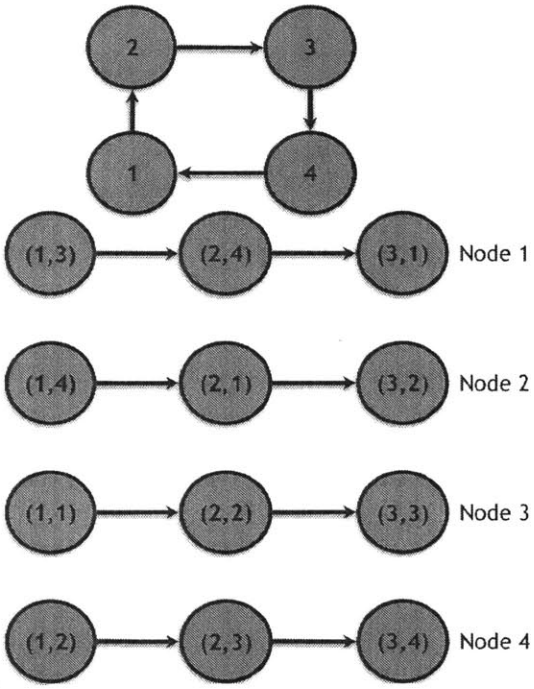


Figure 5-5: A 4-cycle network ($T = 3$) with its Ancestral Network: note that the Ancestral Network consists of four disjoint paths as expected from the periodicity of the original network. We labeled by node 1 the cheapest node of the 4-cycle.

The Ancestral Algorithm applied to the Static Deterministic System can be written out as follows:

ANCESTRAL ALGORITHM ON A PATH FOR THE STATIC DETERMINISTIC MODEL FOR $\eta = 1$:

Without loss of generality, call this path $(1, 2, 3, \ldots, n)$ where 1 is the self-influencing node, and i influences $i+1$ for all $1 \leq i \leq n-1$ (see Fig.4-4). We want to influence the whole chain within a time T (T can be infinite) and at minimal cost.

1. If for node i we have $T+1 \leq i \leq n$, then construct the line containing nodes (t_j, j): $\{(1, i-T+1), (2, i-T+2), \ldots, (T, i)\}$ with their corresponding costs $c(t_j, j) = c_j$. Take the node with cheapest cost (and take the one with smallest time in case of ties).

2. If for node i we have $1 \leq i \leq T$, then construct the line containing nodes (t_j, j): $\{(1,1), (2,1), \ldots, (T-i+1, 1), (T-i+2, 2), \ldots, (T, i)\}$. We pick node $(1,1)$ to influence directly.

This algorithm is extended to any Static Deterministic graphs (not only paths) as follows. Take any node i in the original network:

a. if none of the previous T nodes are self-influencing nodes, apply rule 1 above.

b. otherwise, if there is a self-influencing node (say node k) among the previous T nodes, apply rule 2 above with k as the reference node (i.e., take k as "1").

We now see the power of our Ancestral Algorithm: even if the formalism is more complicated than the one of the Descendant Algorithm, we see that this algorithm can be applied directly to all problems with $\eta = 1$.

5.3 Application of the Ancestral Algorithm to the Case $\eta \in (0,1)$

Our Ancestral Network is also very powerful because it can also be used for any problem with $\eta \in (0,1)$. However, in this case, our solution will involve solving an integer program.

Consider now the Ancestral Network with the costs C associated with it. Let $\mathcal{N}$ be the set of all nodes (t,i) present in the Ancestral Network. Furthermore, let $A(i)$ be the ancestral path of node (T,i) (including node (T,i) according to our earlier definition). We also define $\alpha(t,i)$ to be the following control variable

$$\alpha(t,i) = \begin{cases} 1, & \text{if we influence node } i \text{ at time } t, \\ 0, & \text{otherwise.} \end{cases}$$

The solution is given by solving the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{(t,i)\in\mathcal{N}} \alpha(t,i)c(t,i) \\ \text{subject to} \quad & \sum_{(T,i)\in\mathcal{N}} Y(T,i) \geq n\eta \\ & \forall (T,i) \in \mathcal{N} : 0 \leq Y(T,i) \leq \sum_{(t',i')\in A(i)} \alpha(t',i') \\ & \forall (t,i) \in \mathcal{N} : \alpha(t,i) \in \{0,1\}, \text{ and } Y(T,i) \in \{0,1\} \end{aligned} \tag{5.3}$$

The integer program given in Equation (5.3) will solve our optimization problem, as the objective is to minimize the cost incurred by influencing nodes from the Ancestral Network. The last constraint follows from the definition of $\alpha(t,i)$. The first constraint will guarantee that at least $n\eta$ nodes of the original network will be in the $+1$ state at time T: indeed, $Y(T,i)$ tells us exactly how many leaves of the Ancestral Network will be in the $+1$ state at time T. The only tricky part is to make sure we are not counting the same node multiple times. The second constraint discards this case: if all the $\alpha(t',i') = 0$ for $(t',i') \in A(i)$, then no nodes are in the $+1$ state along the ancestral path, so the leaf (T,i) cannot be in the $+1$ state at time T. If at least one $\alpha(t',i') = 1$ for some $(t',i') \in A(i)$, then the second constraint gives us the freedom to choose $Y(T,i)$ to be 0 or 1: the objective function and

the first constraint will make us choose 1.

The optimization problem in Equation (5.3) provides a general solution to any General Deterministic problem (for any $\eta \in (0,1]$). It also holds when $\eta = 1$ as we will show below. For infinite time-horizon problems, if the system reaches a recurrent state at some time T_0, we can truncate our time-horizon to $T = (T_0+n)$ where n is the number of nodes in the original network (it is actually enough to set $T = (T_0+$*length of largest path from source in Ancestral Network*$)$) and apply Equation (5.3).

In the case $\eta = 1$, the Ancestral Algorithm works because we take the cheapest nodes that can influence the most nodes. We can obtain the same solution as the one given by the Ancestral Algorithm by solving:

$$\begin{aligned} \text{minimize} \quad & \sum_{(t,i)\in\mathcal{N}} \alpha(t,i)c(t,i) \\ \text{subject to} \quad & \sum_{(T,i)\in\mathcal{N}} Y(T,i) = n \\ & \forall (T,i) \in \mathcal{N} : 0 \leq Y(T,i) \leq \sum_{(t',i')\in A(i)} \alpha(t',i') \\ & \forall (t,i) \in \mathcal{N} : \alpha(t,i) \in \{0,1\}, \text{ and } Y(T,i) \in \{0,1\} \end{aligned} \tag{5.4}$$

Equation (5.4) holds only when the Ancestral Network is composed by a single tree. If the Ancestral Network is a forest of disjoint trees, we need to use a knapsack-type integer program.

An equivalent formulation of the problem in 5.3 is given below (note that the first constraint is quadratic):

$$\begin{aligned} \text{minimize} \quad & \sum_{(t,i)\in\mathcal{N}} \alpha(t,i)c(t,i) \\ \text{subject to} \quad & \sum_{(t,i)\in\mathcal{N}} \alpha(t,i)\Big(\gamma(t,i) - \sum_{(t',i')\in\mathcal{D}(t,i)} \alpha(t',i')\gamma(t',i')\Big) \geq n\eta \\ & \alpha(t,i) \in \{0,1\}, \forall (t,i) \in \mathcal{N} \end{aligned} \tag{5.5}$$

We can simplify the optimization by applying the three following rules: for a node (t,i), set $\alpha(t,i) = 0$ if

1. $c(t,i)$ is equal to or higher than the sum of the costs of his direct descendants.

2. $c(t,i)$ is equal to or higher than the sum of the costs of ηn of his descendants at time T.

3. $c(t,i)$ is equal to or higher than the cost of one of his ancestor whose α has not yet been set to zero using the first two rules.

As we will show through an example in Section 5.4, these rules can sometimes reduce the complexity of the optimization problem. As can be seen from the double-sum in Equation (5.5), we can simplify the problem considerably when we can set $\alpha = 0$ for nodes with small values of t.

5.3.1 A Dynamic Programming Solution to the $0 < \eta < 1$ case

Assume $0 < \eta < 1$, and suppose we are already given the Ancestral Network. Our goal is to derive an algorithm that will tell us which nodes of the Ancestral Network we should influence at the optimal cost. In this section, we will use a Dynamic Programming approach of polynomial complexity.

For all nodes (t,i) of the Ancestral Network, define $\mathscr{D}(t,i)$ as the set of descendants of node (t,i): this is the set of nodes (t',i') such that there is a path from (t,i) to (t',i'). In particular, the set $\{(t+1,i') \in \mathscr{D}(t,i)\}$ corresponds to the immediate descendants of node (t,i) in the Ancestral Network. We also define the **k-value $V(t,i,k_i)$ of a node (t,i) of the Ancestral Network** as :

$$V(T,i,k_i) := k_i \times c(T,i), \text{ where } k_i \in \{0,1\} \tag{5.6}$$

$$V(t,i,k_i) := \min\left\{ c(t,i), \min_{\substack{k_{i'} \text{ s.t. } (t+1,i') \in \mathscr{D}(t,i), \\ \text{and } V(t+1,i',k_{i'}) \text{ exists}, \\ \sum_{(t+1,i') \in \mathscr{D}(t,i)} k_{i'} \geq k_i}} \left(\sum_{(t+1,i') \in \mathscr{D}(t,i)} V(t+1,i',k_{i'}) \right) \right\} \tag{5.7}$$

$$\tag{5.8}$$

When $k_i = 1$, the value $V(t,i,k_i)$ of node (t,i) represents the smallest cost needed to have all its final descendants in the $+1$ state. When $k_i = 0$, then $V(t,i,k_i) = 0$ as we decide

not to have any of the final descendants of node (t,i) in the $+1$ state. We also impose the following condition on $V(t,i,k_i)$: if $V(t,i,k) = V(t,i,k')$ for some $k' > k$, then delete the value $V(t,i,k)$ and keep only $V(t,i,k')$. This is what we mean by "$V(t,i,k_i)$ exists".

Then starting from the leaves of the Ancestral Network, it is possible to calculate $V(t,i,k)$ for all nodes (t,i) of the Ancestral Network for each feasible value of k. Then the optimal cost is:

$$V_{\text{opt}} = \min_{\substack{k \geq \eta n, \text{ and} \\ V(1,i_0,k) \text{ exists}}} V(1,i_0,k),$$

where $(1,i_0)$ is the root of the tree considered. Once we know the optimal cost, we find an optimal policy just by keeping track of our minimizing selections at each DP iteration.

Note that the inner minimization

$$\min_{\substack{k_{i'} \text{ s.t. } (t+1,i') \in \mathscr{D}(t,i), \\ \text{and } V(t+1,i',k_{i'}) \text{ exists}, \\ \sum_{(t+1,i') \in \mathscr{D}(t,i)} k_{i'} \geq k_i}} \left(\sum_{(t+1,i') \in \mathscr{D}(t,i)} V(t+1,i',k_{i'}) \right)$$

can be solved in polynomial time using the following dynamic program:

For all $i \in \{1,\ldots,n\}$, let $W_{i,t}(k')$ be the minimal cost the controller incurs if he aims to influence a total of k' nodes among the set $\{i,\ldots,n\}$ at time $t+1$. Then:

$$W_{n,t}(k') := V(t+1,n,k') \tag{5.9}$$

$$W_{i-1,t}(k') := \min_{k=1,\ldots,k'} \Big[W_{i,t}(k'-k) + V(t+1,i-1,k) \Big]. \tag{5.10}$$

It is convenient to let $V(t+1,n,k')$ be infinite if k' is higher than the possible number of nodes $(t+1,n)$ can influence.

When the Ancestral Network consists of a single tree, we use the recursion in (5.10) which can be implemented in $O(n^3)$ time.

If the Ancestral Network is a forest of trees with roots i_1, i_2, ..., i_q. Then the optimal cost for the forest is given by the following knapsack problem:

$$V_{\text{opt}} = \min_{\substack{k_1,\ldots,k_q \text{ s.t.} \\ V(1,i_j,k_j) \text{ exists,} \\ \text{and } \sum_{j=1}^{q} k_j \geq \eta n}} \sum_{j=1}^{q} V(1, i_j, k_j). \tag{5.11}$$

This knapsack problem admits a polynomial-time algorithm of complexity $O(n \times \eta n \times nT) = O(n^3 T)$.

5.4 Illustrative Example

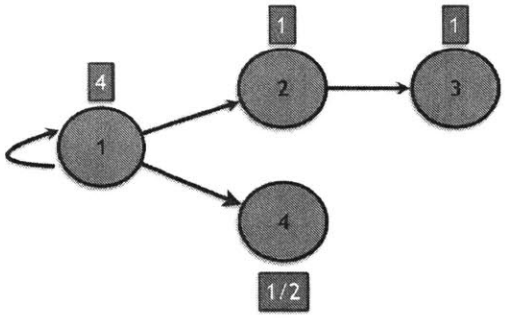
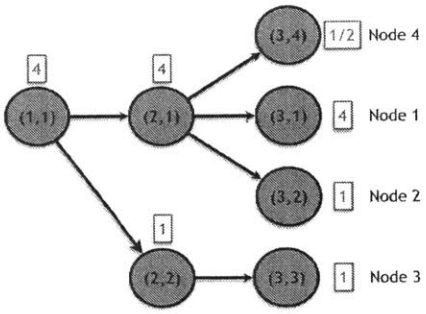


Figure 5-6: This network is assumed to be the same for times $t = 1, 2, 3$ (this corresponds to a Simple Deterministic System). We assume $T = 3$, and the costs C of each node are given in the red boxes.

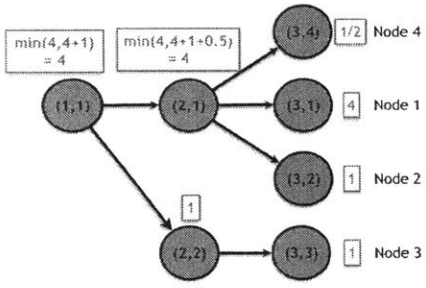
In this section we will analyze the network depicted in Fig. 5-6. For simplicity, we will use a Simple Deterministic network (i.e., this network is the same at times $t = 1, 2, 3$). For our General Deterministic Model, the same procedure applies.

We first build the Ancestral Network from the given network (Fig. 5-7a): in the Ancestral Network, each node represents the couple (t_i, i), where t_i is the time at which node i is an ancestor of the node considered. This is the only step that differs slightly in the case of a General Deterministic model because we need to pay attention the time evolution of the given network. We also include the costs of each node of the Ancestral Network.

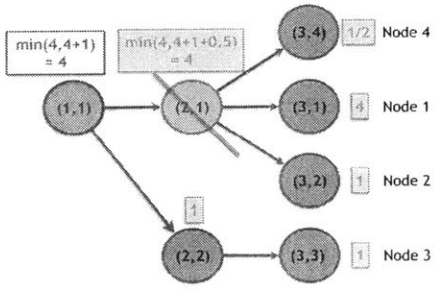
In the first step of the Ancestral Algorithm (Fig. 5-7b), we solve the recursion in (5.2). Starting from the leaves, we assign value $V(T, i) = c_i$ leaf (T, i). Then we move to the ancestor of each leaf and, using the minimization in (5.2), we obtain the values $V(T-1, i)$



(a) Ancestral Network for Fig. 5-6: the cost of the nodes are reported in the pink boxes.



(b) First iteration of the Ancestral Algorithm: we compute the values of each node (t, i) in the green boxes.



(c) Second iteration of the Ancestral Algorithm: we mark all nodes whose value are equal to their cost. We then delete all descendants of marked nodes.

Figure 5-7: Ancestral Network for Fig. 5-6 and its evolution through the Ancestral Algorithm: nodes are represented by the couple (t_i, i).

for node $(T-1, i)$. We continue in that fashion until we reach the source $(1, 1)$.

In the second step of the Ancestral Algorithm (Fig. 5-7c), we mark all nodes whose value $V(t, i)$ is equal to c_i. We then delete all descendants of a marked node. The remaining marked nodes are the ones we should influence.

Therefore the Ancestral Algorithm picks node $(1, 1)$ in the Ancestral Network, so we must influence node 1 at time $t = 1$.

Now suppose we want to influence $\eta = 50\%$ of the nodes of the network. We use the Ancestral Network, and associate to it the costs of the nodes as shown in Fig. 5-8.

Now, $\eta n = 2$, and $\mathcal{N} = \{(1, 1), (2, 1), (2, 2), (3, 1), (3, 2), (3, 3), (3, 4)\}$ with respective

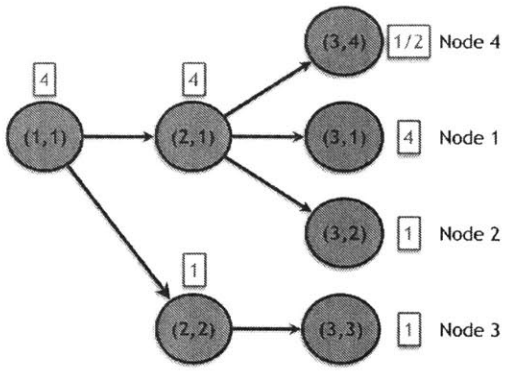


Figure 5-8: Ancestral Network for Fig. 5-6: the original cost of each node is reported in the pink boxes.

costs $C = \{4, 4, 1, 4, 1, 1, 0.5\}$. Applying Equation (5.5), we need to solve:

$$\begin{aligned}
\text{minimize} \quad & 4\alpha(1,1) + 4\alpha(2,1) + \alpha(2,2) + 4\alpha(3,1) + \alpha(3,2) + \alpha(3,3) + 0.5\alpha(3,4) \\
\text{subject to} \quad & \alpha(1,1)\Big(4 - 3\alpha(2,1) - \alpha(2,2) - \alpha(3,1) - \alpha(3,2) - \alpha(3,3) - \alpha(3,4)\Big) \\
& +\alpha(2,1)\Big(3 - \alpha(3,1) - \alpha(3,2) - \alpha(3,4)\Big) \\
& +\alpha(2,2)\Big(1 - \alpha(3,3)\Big) \\
& +\alpha(3,1) + \alpha(3,2) + \alpha(3,3) + \alpha(3,4) = 2 \\
& \alpha(t,i) \in \{0,1\}, \forall (t,i) \in \mathcal{N}
\end{aligned} \tag{5.12}$$

We will now simplify the problem using the three rules outlines in the previous section:

1. Rule 1 sets $\alpha(2,1) = 0$.
2. Rule 2 sets $\alpha(1,1) = 0$.
3. Rule 3 sets $\alpha(3,3) = 0$.

So the optimization problem reduces to the following Integer Program:

$$\begin{aligned}
\text{minimize} \quad & \alpha(2,2) + 4\alpha(3,1) + \alpha(3,2) + 0.5\alpha(3,4) \\
\text{subject to} \quad & \alpha(2,2) + \alpha(3,1) + \alpha(3,2) + \alpha(3,4) = 2 \\
& \alpha(t,i) \in \{0,1\}, \forall (t,i) \in \mathcal{N}
\end{aligned} \tag{5.13}$$

Using the constraint, we get: $\alpha(2,2) + \alpha(3,2) = 2 - \alpha(3,1) - \alpha(3,4)$. Hence we need to:

$$\begin{aligned} \text{minimize} \quad & 2 + 3\alpha(3,1) - 0.5\alpha(3,4) \\ \text{subject to} \quad & \alpha(3,1) \in \{0,1\} \\ & \alpha(3,4) \in \{0,1\} \end{aligned} \tag{5.14}$$

Thus $\alpha(3,1) = 0$ and $\alpha(3,4) = 1$. So we must have $\alpha(2,2) + \alpha(3,2) = 2 - 0 - 1 = 1$. Hence $\alpha(3,4) = 1$ and either $\alpha(2,2) = 1$ or $\alpha(3,2) = 1$ are two optimal solutions to the optimization problem. We see that, in this example, our three rules have simplified the problem a lot !

By inspection, we can also see that there are exactly three optimal solutions from Fig. 5-8: we can influence node 4 at time $t = 3$, and either node 2 at $t = 2$ or $t = 3$, or node 3 at $t = 3$. Hence: $\alpha(3,4) = 1$ and either $\alpha(2,2) = 1$, or $\alpha(3,2) = 1$, or $\alpha(3,3) = 1$.

5.5 Summary of the Results

In this section, we summarize the main results from Chapters 4 and 5. In this first part of the thesis, we considered in-degree one networks over which nodes interact according to deterministic dynamics. An external controller wants to influence the nodes in order to get a population sharing the $+1$ opinion, by some time T. Each time the controller decides to influence a node, it will incur a certain cost. Our goal is to design a policy that the external controller can follow in order to attain its goal with the smallest possible cost.

In Chapter 4, we assumed that the network (of n nodes) did not evolve with time and we were able to construct the Descendant Algorithm to achieve, in linear time $O(n)$, the controller's goal. In Chapter 5, we allowed the network (of n nodes) to evolve with time in a deterministic way, and we were able to construct the Ancestral Algorithm to achieve, in linear time $O(n)$, the controller's goal.

If the external controller wants to influence the nodes in order to get a fraction η (for some $0 < \eta < 1$) of 0 and 1 opinions among the population, by some time T, then we formulated a Dynamic Program which achieves the controller's goal in polynomial time $O(n^3)$ if the Ancestral Network contains a single tree. When the Ancestral Network contains more than a tree (which happens when $k-$cycles are present in the original network), we

basically need to solve a knapsack problem.

Part II

Static versus Dynamic Policies for the Voter Model

Chapter 6

Introduction

The previous part of the thesis focused on deterministic processes taking place over a given network. In contrast, the second part of the thesis focuses on a particular type of stochastic process taking place over a network: the classical and well studied "Voter Model" [3, 8].

The Voter Model can be described as follows (for a detailed survey, Liggett [12] is a good reference). We start with a network of N nodes in which each node has an opinion (which we denote 0 or 1 for simplicity) and a fixed set of neighbors. At each time step, a node updates its opinion by randomly selecting one of its neighbors and adopting the opinion of the neighbor selected. In this thesis we will only consider the case where the graph of the network is the $d-$dimensional integer torus $\mathbb{Z}_n^d$. Following the work of Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14], as well as Fagnani [7], we first consider static models in which we allow nodes of the network to be "stubborn" (they are commonly referred to as "stubborn nodes" or "stubborn agents"): this means that such nodes maintain their initial opinion during the whole process.

6.1 Scope and Main Objectives

In this part of the thesis, we assume that we can place a set $\mathscr{B}$ of B stubborn nodes (also called stubborn nodes) over the network. All stubborn nodes have the same opinion "1". We know in this case [14] that the network will eventually reach a $1-$consensus: we mean by this that all non-stubborn nodes in the network will be in the $1-$state after some finite

time. Our goal is to measure the expected time until the network reaches the 1−consensus.

We first consider a static setting in which we initially position our B stubborn nodes over the lattice and compute the expected time the system requires to reach consensus. We then consider a dynamic setting in which we are allowed to move our B stubborn nodes at each time step, as a function of the current state of the network. Given a number B of stubborn nodes, we want to answer the following questions: "how fast can we reach consensus in the static setting ?" and "can we do better in the dynamic setting ?".

6.2 Related Work and Contributions

The Voter Model is a classical type of Markov process which was introduced independently by Clifford and Sudbury [3], and Holley and Liggett [8]. The main questions to be answered are usually: "what is the probability that the process reaches a consensus ?", "if a consensus is reached, what is the probability of having a consensus with all nodes in the 1−state ?", and "what is the expected time to reach a consensus ?".

Holley and Liggett [8] have proved that a consensus will always be reached over the infinite lattice $\mathbb{Z}^d$ when $d = 1$ or 2, but consensus is not necessarily reached in higher dimensions. However, on finite networks a consensus is always reached. Cox [4] has later shown that for a Voter Model over a d−dimensional integer torus $\mathbb{Z}_n^d$ which starts from a product measure with probability $p \in (0,1)$ of starting in the $+1$ state, we expect consensus to be reached in time $O(c_p n^2)$ when $d = 1$, $O(c_p n^2 \log n)$ when $d = 2$, and $O(c_p n^d)$ when $d \geq 3$, where $c_p = -p\log p - (1-p)\log(1-p)$. Note that in this thesis, we will not initialize our lattice randomly, and we allow some nodes to be stubborn.

Indeed, we will focus on a generalization introduced by Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14], and Fagnani [7], in which we allow some nodes to maintain their original opinion throughout the whole process: these nodes are called "stubborn nodes" or "stubborn agents". Clearly, if all stubborn nodes are in the same state (say the 1−state), then the network will reach a 1−consensus with probability one [14]. Using a Theorem from Aldous and Fill [1], Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14] have shown that, when stubborn nodes were added in the network, the expected time to reach consensus can

be upper bounded (up to a log factor depending on the size of the network) by the longest expected time a symmetric random walk would take to hit a stubborn node.

Using the results mentioned above, the main goals of this part of the thesis are as follows. First, we want to find a static placement of our B nodes that minimizes the expected time to reach the 1−consensus. Then, we allow our stubborn nodes to move inside the network, and check if there exists a sequence of placements of the stubborn nodes in the dynamic setting which leads to a shorter expected time to reach consensus than an optimal placement of the stubborn nodes in the static setting.

6.3 Outline of Part II

The rest of this part of the thesis is organized as follows. In Chapter 7 we provide some background on the Voter Model, and introduce the "dual Voter Model" approach, a classical tool often used to compute consensus times. Then, in Chapter 8 we restrict our analysis to the Voter Model over a d−dimensional integer torus, and we construct a static placement of stubborn nodes that minimizes the expected time needed to set all nodes in the state 1. Finally, in Chapter 9, we study the same model but allow ourselves to move our set of stubborn nodes during the experiment. In this last chapter, we want to understand whether there exist dynamic policies providing strictly smaller consensus times than optimal static policies: we will actually show that dynamic policies do not perform significantly better than static policies when $d \geq 2$.

Chapter 7

Background on the Voter Model

In this chapter, we provide some background information about the Voter Model. In particular, we present the classical dual approach to the Voter Model. This approach is very useful: it provides an elegant correspondence between consensus times for a Voter process, and coalescence times for simple random walks. This framework will be much simpler to work with and will be used extensively in Chapter 8.

We first start by describing the Voter process and apply it to the model we will be using. We then introduce the classical dual process, and present a Proposition from Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14] which will be heavily used in Chapter 8.

7.1 The Voter Process for our Model

In this thesis, we consider the Voter Model over a $d-$dimensional integer torus $\mathbb{Z}_n^d$ (which we also refer to as a $d-$dimensional periodic lattice). Such a lattice contains $N = n^d$ nodes. We always denote the total number of nodes in the lattice by N, and the number of nodes on each side of the lattice by n. At each instant, a node will be in one of two states, 0 or 1.

Nodes in the network update their state according to independent Poisson processes of rate $\nu = 1$. We say that a Poisson clock ticks whenever the Poisson process registers an arrival (Poisson clock and Poisson process will be used interchangeably here). We will assume each node to have its own Poisson clock of rate 1, and all Poisson clocks are independent from each other. When a node's clock ticks, the node will choose one of his neighbors

uniformly at random and will adopt the state of that chosen neighbor.

In addition, we possess a set $\mathscr{B}$ of B "stubborn nodes": these nodes are special as they do not update their state during the process and keep their original state throughout the whole experiment. We assume that all stubborn nodes in our problem are in the $+1$ state. In Chapter 8, we will place the set of stubborn nodes over the lattice at the beginning of the process and wait for the system to reach consensus. In Chapter 9, we will be able to modify the positions of the stubborn nodes during the run of the experiment. Our goal is to make the network reach a $1-$consensus: we mean by this that we want all nodes in the network to be in the $+1$ state after some finite time.

It is already known that the Voter model always reaches a consensus over finite lattices (e.g., see Cox [4]). In particular, in the presence of stubborn nodes which all share the same state (say, $+1$), the network will always reach a $1-$consensus in finite time (see Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14]). Since a $1-$consensus is reached with probability 1, we are interested in finding *how fast* the $1-$consensus is reached. As discussed in Chapters 8 and 9, this will depend on the chosen placement of stubborn nodes over the lattice.

In this part of the thesis, we assume that all non-stubborn nodes are initially in the sate 0. Note that a network initialized with some non-stubborn nodes already in the $+1$ state cannot reach a $1-$consensus slower (in expectation) than a network initialized with all non-stubborn nodes in the 0 state, given a same placement of $1-$stubborn nodes. Hence considering all non-stubborn nodes initially in the 0 state corresponds to a "worst case scenario".

7.2 The Dual Process

To compute consensus times, it is often easier to look at the process backwards in time: this classical approach is referred to as the *dual of the Voter process*, or as the *coalescing random walk process*. For a more detailed discussion of the dual process, see Aldous and Fill [1] (Chapter 14.3), or Durett [6] (Chapter 6.9). The dual approach has been used for the Voter Model with stubborn nodes in Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14].

As presented in Aldous and Fill [1] (Chapter 14.3), in the dual approach, we consider a particle at each node of the lattice. The particles all perform independent symmetric random walks over the lattice. If two particles meet at a node, they *coalesce* i.e., they merge and move as a single symmetric random walk. If a particle lands at a stubborn node, it stays there indefinitely.

Let T_{hit} be the time until all random walks hit a stubborn node, and T_{voter} be the time until the Voter process reaches consensus. As discussed in Aldous and Fill [1] (Chapter 14.3), these random variables have the same distribution. Intuitively, if we look at the Voter process backwards in time, we will see the dual process. In fact, in the dual process we are tracking the origin of the current state of the node.

Using the dual approach, Yildiz, Acemoglu, Ozdaglar, Saberi and Scaglione [14] prove the following proposition:

Proposition 7.2.1. *Given a set $\mathscr{B}$ of B stubborn nodes in the $+1$ state, the expected time to reach consensus is bounded above by*

$$\mathbb{E}[T_{consensus}] \leq e \log(2 + N - B) \max_{i \notin \mathscr{B}} \mathbb{E}[T_i],$$

where $\mathbb{E}[T_i]$ is the expected time a random walk initialized at node i first hits the set $\mathscr{B}$.

It can be shown that $\mathbb{E}[T_i]$ is bounded above by $O(N^3)$ for general graphs (e.g., see Aldous and Fill [1]). Therefore consensus is expected to be reached by time $O(N^3 \log N)$ in general. Our goal is to get much sharper bounds in the case of $d-$dimensional integer tori, and to also capture the dependence on B, as B increases.

For the rest of the thesis, we will use the following notation: We define τ^* as the expected time to reach consensus in the Voter process. We define τ as the expected hitting time from one point in the lattice to the set of stubborn nodes in the worst case over all stubborn nodes. If $\mathscr{B}$ is the set of stubborn nodes then $\tau = \max_{i \notin \mathscr{B}} \mathbb{E}_i[T_{\mathscr{B}}]$, where $\mathbb{E}_i[T_{\mathscr{B}}]$ is the expected time a random walk initialized at node i first hits the set $\mathscr{B}$.

Using Proposition 7.2.1, we immediately get the following corollary:

Corollary 7.2.2. *Given a set $\mathscr{B}$ of B stubborn nodes in the $+1$ state:*

$$\tau \leq \tau^* \lesssim \tau \log(N - B)$$

With this corollary in mind, we will simplify our analysis in Chapter 8 by computing τ instead of τ^*. This will allow us to use well understood tools from Markov Processes and the random walk literature (see Aldous and Fill [1], Levin book). When the stubborn nodes are allowed to move (see Chapter 9), the dual process will not be helpful as we would need to compute the expected time until a random walk hits a moving set.

Chapter 8

Convergence Time to Consensus for a Static Policy

In this chapter, we study the following static policy: given a budget B of stubborn nodes with opinion $+1$, we place these nodes over a $d-$dimensional lattice $\mathbb{Z}_{N^{1/d}}^d$ of N nodes at the start of the experiment and then we try to bound the expected time needed for such a network to reach consensus.

This chapter is structured as follows: after having introduced the model used here, we will consider random walks over $\mathbb{Z}^d$ which will be helpful to develop general lower bounds for the desired consensus times. We will then study the effect of placing all the budget on the boundary of the lattice, as well as "spreading" all the budget over the lattice. This will allow us to construct an optimal placement for stubborn nodes which minimizes the expected time needed to reach consensus. Finally, we will attempt to rederive a result from Levin, Peres and Wilmer [11] which we use heavily through the chapter.

8.1 Model Description

We begin by describing the process studied in this chapter. As mentioned in Chapters 6 and 7, we will mainly focus on square lattices with periodic boundary conditions (i.e., tori) of dimension d with N nodes.

We possess a budget B of stubborn nodes in the $+1$ state. We allow this budget to

depend on N. Although we will obtain results for general budgets B, we typically start by considering budgets of the type $B \sim O(1)$ (this corresponds to a "very small" budget), budgets of the type $B \sim O(N^{\frac{d-1}{d}})$ (this corresponds to having a budget comparable to the size of a full face of the $d-$dimensional lattice), and budgets of the type $B \sim O(N)$ (this corresponds to having a budget comparable to the size of the full lattice). We thus position these stubborn nodes on the lattice and bound the expected time $\tau^*(N)$ for the process to reach consensus. Recall that nodes in the lattice update their opinion by selecting the opinion of one of their neighbors uniformly at random, and that the times at which a node updates his opinion occurs when the rate-1 Poisson process of this node registers an arrival.

As we will show later, the expected time to reach consensus can change significantly if we modify the initial placement of the stubborn nodes. We typically consider two types of placements for our stubborn nodes: we either place all our stubborn nodes on the boundary of the lattice, or we spread the stubborn nodes uniformly over the lattice.

Definition 8.1.1. Consider a d-dimensional lattice with N nodes and a budget B of stubborn nodes. Assume that the quantity $s = \left(\frac{N}{B}\right)^{1/d}$ is an integer. We say that we spread the budget B of stubborn nodes uniformly over the lattice if the positions of the stubborn nodes form a subgrid of the lattice with spacing s.

If $\left(\frac{N}{B}\right)^{1/d}$ is not a integer, we find some s such that $s = \left(\frac{N}{B}\right)^{1/d} + o(1)$ and place the stubborn nodes over a subgrid of spacing s.

Given a budget B, we have roughly $B^{1/d}$ stubborn nodes on each edge of the lattice. We want these $B^{1/d}$ nodes to be spaced by a distance s. An edge of the lattice has size $N^{1/d}$, therefore we require $s \times B^{1/d} = N^{1/d}$. We solve for s and get $s = \left(\frac{N}{B}\right)^{1/d}$ (ignoring the fact that the result may not be an integer).

8.2 Main Results

We summarize here the main results we will obtain in this chapter. We define τ^* as the expected time to reach consensus in the Voter process. We define τ as the expected hitting time from one point in the lattice to the set of stubborn nodes in the worst case over all

stubborn nodes. If $\mathscr{B}$ is the set of stubborn nodes then $\tau = \max_{i \notin \mathscr{B}} \mathbb{E}_i[T_{\mathscr{B}}]$, where $\mathbb{E}_i[T_{\mathscr{B}}]$ is the expected time a random walk initialized at node i first hits the set $\mathscr{B}$.

Remark. Throughout the chapter, we always consider the **worst case scenario** for hitting times: this means that we consider the largest expected hitting time to a set over all starting node in the lattice.

In the chapter, we compute the expected time it takes a single random walk (in the worst case scenario) to hit a stubborn node. We use Corollary 7.2.2 to deduce the expected time to reach consensus by multiplying our result by a $\log(N-B)$ factor. We are given a budget B of stubborn nodes which we need to place over a $d-$dimensional periodic lattice of N nodes.

If we spread our budget evenly over the lattice, then (see Section 8.6):

$$\begin{cases} (\frac{N}{B})^2 \lesssim \tau^* \lesssim (\frac{N}{B})^2 \log(N-B), & \text{if } d=1, \\ \left(\frac{N}{B}\right) \log\left(\frac{N}{B}\right) \lesssim \tau^* \lesssim \left(\frac{N}{B}\right) \log\left(\frac{N}{B}\right) \log(N-B), & \text{if } d=2, \\ \left(\frac{N}{B}\right) \lesssim \tau^* \lesssim \left(\frac{N}{B}\right) \log(N-B), & \text{if } d \geq 3. \end{cases} \tag{8.1}$$

If $B = \Omega\left(N^{\frac{d-1}{d}}\right)$, and if we place our budget over the boundary of the lattice as well as over equally spaced internal slices of the lattice, then (see Section 8.5):

$$\left(\frac{N}{B}\right)^2 \lesssim \tau^* \lesssim \left(\frac{N}{B}\right)^2 \log(N-B) \text{ , when } d \geq 1. \tag{8.2}$$

If $B = o\left(N^{\frac{d-1}{d}}\right)$, and if we place our budget over the faces of a small cube of side $B^{\frac{1}{d-1}}$ in the lattice, then we *conjecture* the following upper bounds (see Section 8.5.4):

$$\begin{cases} B^2 \vee (N-B)^2 \lesssim \tau^* \lesssim \left(B^2 \vee (N-B)^2\right) \log(N-B), & \text{if } d=1, \\ B^2 \vee N \log\left(\frac{N}{B^2}\right) \lesssim \tau^* \lesssim \left(B^2 \vee N \log\left(\frac{N}{B^2}\right)\right) \log(N-B), & \text{if } d=2, \\ B^{\frac{2}{d-1}} \vee \frac{N}{B} B^{\frac{1}{d-1}} \lesssim \tau^* \lesssim \left(B^{\frac{2}{d-1}} \vee \frac{N}{B} B^{\frac{1}{d-1}}\right) \log(N-B), & \text{if } d \geq 3. \end{cases} \tag{8.3}$$

To obtain the above results on τ^*, we bound the hitting time τ and use Corollary 7.2.2 to bound τ^*. We will show in Section 8.7 that spreading the budget of stubborn nodes

uniformly over the lattice is actually an optimal policy for $d \geq 3$ (as it produces the smallest value of τ). Indeed, we will show in Proposition 8.4.1 that the lower bound $(N/B) \lesssim \tau$ is tight for $d \geq 3$ (as it is reached when we spread the budget uniformly over the lattice). The optimal placement of stubborn nodes when $d = 2$ will be left open, but we will conjecture that the expected hitting time τ scales as $(N/B)\log(N/B)$ if we place our stubborn nodes in an optimal way. When $d = 1$, we also prove that spreading the budget is an optimal policy.

We see that, for $d \geq 2$, placing the budget over the boundary of the lattice is much worse that spread the budget evenly over the lattice; the latter is order optimal and we will see in Chapter 9 that it is close to optimal when compared to dynamic placements.

8.3 Symmetric Random Walks on a Lattice of Dimension d

Before diving into hitting times for more complicated settings, we start by developing some easy results for a much simpler and well known setting. In this section, we consider a symmetric random walk on $\mathbb{Z}^d$: we are interested in bounding the expected time for a random walk initialized at the origin to escape from a $d-$dimensional hypercube of side l centered at the origin.

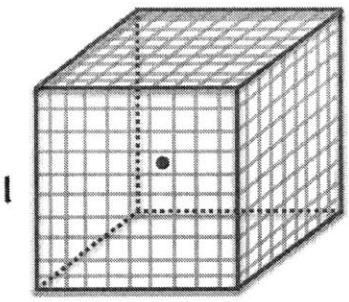


Figure 8-1: $3-$dimensional cube of size l in $\mathbb{Z}^3$ over which we will run a random walk initialized at center (red node). We prove in Lemma 8.3.1 that the expected time for the random walk to hit the boundary of the cube is of order $\Theta(l^2)$.

We begin by defining the symmetric random walk on $\mathbb{Z}^d$. For each $i \in \{1, \ldots, d\}$, let $\mathbf{e}_i$ be the unit vector whose only nonzero component is the i^{th} one. Let $\{X_k\}_{k=0}^{\infty}$ be i.i.d. random variables such that $\mathbb{P}(X_k = \mathbf{e}_i) = \frac{1}{2d}$ and $\mathbb{P}(X_k = -\mathbf{e}_i) = \frac{1}{2d}$ for each $i \in \{1, \ldots, d\}$,

and let $X_0 = \mathbf{0}$ be the origin. We define the random walk $\mathbf{S_{k+1}} = \mathbf{S_k} + X_{k+1} = \sum_{i=0}^{k+1} X_i$ initiated at the origin $\mathbf{S_0} = \mathbf{0}$.

Lemma 8.3.1. *For any $d \geq 1$, the random walk $\{\mathbf{S_k}\}_{k=0}^{\infty}$ defined above will exit a $d-$dimensional square of side l centered at the origin in time $\Theta(l^2)$, for a fixed d.*

Proof. We define our stopping time

$$\tau_l = \min\{k \geq 1 : \text{ one component of } \mathbf{S_k} \text{ is } \pm \frac{l}{2}\}.$$

τ_l corresponds to the first time our random walk reaches the edge of the $d-$dimensional square of side l centered at the origin. We consider the random walk $\mathbf{S_k} = (S_k^1, \ldots, S_k^d)$ defined above, which is initialized at node $(0, \ldots, 0)$. The walk halts as soon as S_k^i reaches the value $l/2$ for some $i \in \{1, \ldots, d\}$.

To get the lower bound, we argue as follows. Define $Y_k = \|\mathbf{S_k}\|^2 - k$, where $\|.\|$ is the euclidean norm. We first show that Y_k is a martingale. Note that $\mathbb{E}\left[\|\mathbf{S_k}\|^2\right] = \mathbb{E}\left[\mathbf{S_k} \cdot \mathbf{S_k}\right] = \sum_{i=1}^{k}\sum_{j=1}^{k} \mathbb{E}[X_i \cdot X_j]$, so by independence of the X_i (and since $\mathbb{E}[X_i] = 0, \|X_i\| = 1$) we get that $\mathbb{E}\left[\|\mathbf{S_k}\|^2\right] = \sum_{i=1}^{k} \mathbb{E}\left[\|X_i\|^2\right] = k$. Hence, $\mathbb{E}[Y_k] = \mathbb{E}\left[\|\mathbf{S_k}\|^2\right] - k = 0$ which is bounded for all k. We finish by computing

$$\begin{aligned}
\mathbb{E}\left[Y_{k+1}|Y_k\right] &= \mathbb{E}\left[\mathbf{S_{k+1}} \cdot \mathbf{S_{k+1}}|Y_k\right] - k - 1 \\
&= \mathbb{E}\left[(\mathbf{S_k} + X_{k+1}) \cdot (\mathbf{S_k} + X_{k+1})|Y_k\right] - k - 1 \\
&= \mathbb{E}\left[\|\mathbf{S_k}\|^2 + 2\mathbf{S_k} \cdot X_{k+1} + \|X_{k+1}\|^2 \Big| Y_k\right] - k - 1 \\
&= Y_k + 2\mathbb{E}\left[\mathbf{S_k} \cdot X_{k+1}|Y_k\right] \\
&= Y_k + 2\mathbb{E}\left[\mathbf{S_k}|Y_k\right] \cdot \mathbb{E}\left[X_{k+1}\right], \text{ as } X_{k+1} \text{ is independent of both } \mathbf{S_k} \text{ and } Y_k \\
&= Y_k + 2\mathbb{E}\left[\mathbf{S_k}|Y_k\right] \cdot \mathbf{0} \\
&= Y_k.
\end{aligned}$$

Note that $\frac{l}{2} \leq \|\mathbf{S_k}\| \leq l\frac{\sqrt{d}}{2}$. Applying Doob's Optional Stopping Theorem, we get $\mathbb{E}[\tau_l] = \mathbb{E}\left[\|\mathbf{S}_{\tau_l}\|^2\right] \geq (\frac{l}{2})^2 = \frac{l^2}{4}$. Therefore $\mathbb{E}[\tau_l] \geq \frac{l^2}{4}$, hence $\mathbb{E}[\tau_l] = \Omega(l^2/4)$.

Applying Doob's Optional Stopping Theorem for the upper bound, we get $\mathbb{E}[\tau_l] =$

$\mathbb{E}\left[\|\mathbf{S}_{\tau_l}\|^2\right] \leq (l\frac{\sqrt{d}}{2})^2 = \frac{l^2 d}{4}$. Therefore $\mathbb{E}[\tau_l] \leq \frac{l^2 d}{4}$, hence $\mathbb{E}[\tau_l] = O(l^2 d/4)$. When d is fixed, we conclude that $\mathbb{E}[\tau_l] = \Theta(l^2)$.

□

This simple lemma will be useful in the following sections as it helps us to prove simple lower bounds for more general hitting times of random walks over lattices (Proposition 8.4.1), and for analyzing the expected time needed to reach consensus for the voter process when all stubborn nodes are placed on the boundary (results in Section 8.5). The case $d = 1$, in which we consider the hitting time for a random walk over a cycle, will be used in Lemma 8.6.1, and Theorem 8.6.2.

8.4 A General Lower Bound for Hitting Times on a Lattice

In this section, we will consider a general budget B (which can possibly depend on N) allocated arbitrarily over a $d-$dimensional lattice of N nodes with periodic boundary conditions. Considering a symmetric $d-$dimensional random walk starting from any non stubborn node of the lattice, we provide a lower bound for the first time that the random walk hits a stubborn node.

We begin by stating and proving a proposition which directly follows from Lemma 8.3.1.

Proposition 8.4.1. *We are given an arbitrary budget B (which can possibly depend on N). Let X_k be a symmetric $d-$dimensional random walk over the periodic lattice, starting from any non stubborn node of the lattice. Let τ be the maximum expected first time that the random walk hits a stubborn node, over all starting point of the random walk. Then $E[\tau] = \Omega\left(\left(\frac{N}{B}\right)^{\frac{2}{d}}\right)$, with d fixed.*

Proof. Let s be the spacing in the case of a uniform spreading: $s = \left(\frac{N}{B}\right)^{1/d}$. Observe that for a general placement of the nodes there must exist a non-stubborn node in the lattice which is at a Euclidean distance greater or equal to $s/2$ from its closest stubborn node. We can therefore lower bound the expected time the random walk initiated at this node would take to hit a stubborn node by computing the time it takes for this random walk to exit

a square of side s. From Lemma. 8.3.1, we obtain $\mathbb{E}[\tau] = \Omega(s^2)$, which gives the desired lower bound. □

Even though this lower bound is clearly tight for $d = 1$ (because of the geometry of a line), it is not tight enough for higher dimensions. We can get a better lower bound using more sophisticated tools from Aldous and Fill [1]. Let $\mathscr{B}$ be the set of stubborn nodes, let X_k be a symmetric $d-$dimensional random walk over the $d-$dimensional periodic lattice, and define

$$T_{\mathscr{B}}^{+} = \min\{k \geq 1 | X_k \in \mathscr{B}\}\,, \text{ and}$$

$$T_{\mathscr{B}} = \min\{k \geq 0 | X_k \in \mathscr{B}\}\,.$$

$T_{\mathscr{B}}^{+}$ is the first return time to the set of stubborns, and $T_{\mathscr{B}}$ the first hitting time to the set of stubborns ($T_{\mathscr{B}}$ and $T_{\mathscr{B}}^{+}$ are equal unless $X_0 \in \mathscr{B}$). We also define π_i as the stationary distribution of the Markov chain at the state i, and $\pi(\mathscr{B}) = \sum_{i \in \mathscr{B}} \pi_i$. Also, $\mathbb{E}_{\pi_{\mathscr{B}}}[.]$ corresponds to taking an expected value when the Markov chain is initialized from the stationary distribution over $\mathscr{B}$. In Chapter 2 of Aldous and Fill [1], the authors prove "Kac's formula":

Lemma 8.4.2. *(Kac's Formula) Using the notations introduced above:* $\mathbb{E}_{\pi_{\mathscr{B}}}\left[T_{\mathscr{B}}^{+}\right] = \frac{1}{\pi(\mathscr{B})}$.

Using Kac's formula, we will be able to prove a better lower bound for the hitting time over an arbitrary set of stubborn nodes $\mathscr{B}$.

Proposition 8.4.3. *Let τ_i be the expected hitting time of the random walk, starting at node i, over the usual $d-$dimensional lattice on the set of stubborn nodes $\mathscr{B}$. Define τ as $\max_{i \notin \mathscr{B}} \tau_i$. Assume the lattice has N nodes, and $|\mathscr{B}| = B$. Then $\mathbb{E}[\tau] = \Omega(N/B)$.*

Proof. First, note that $\pi_i = 1/N$ for any node i in the lattice, therefore $\pi(\mathscr{B}) = \sum_{i \in \mathscr{B}} \pi_i = \frac{B}{N}$. From Kac's formula, we get $\mathbb{E}_{\pi_{\mathscr{B}}}\left[T_{\mathscr{B}}^{+}\right] = \frac{1}{\pi(\mathscr{B})} = N/B$.

We now want to show that $1 + \mathbb{E}[\tau] \geq \mathbb{E}_{\pi_{\mathscr{B}}}\left[T^+_{\mathscr{B}}\right]$. We have:

$$\begin{aligned}
\mathbb{E}_{\pi_{\mathscr{B}}}\left[T^+_{\mathscr{B}}\right] &= \mathbb{P}(\mathscr{B} \to \mathscr{B} \text{ transition}) + \mathbb{P}(\mathscr{B} \to \mathscr{B}^c \text{ transition})\mathbb{E}_{\mathscr{B}^c}[T_{\mathscr{B}}] \\
&\leq 1 + \mathbb{E}_{\mathscr{B}^c}[T_{\mathscr{B}}] \\
&\leq 1 + \sum_{i \notin \mathscr{B}} \tau_i \mathbb{P}(\mathscr{B} \to i \text{ transition}) \\
&\leq 1 + \max_{i \notin \mathscr{B}} \tau_i = 1 + \mathbb{E}[\tau],
\end{aligned}$$

as τ considers the worst case scenario. Therefore $\mathbb{E}[\tau] \geq \frac{N}{B} - 1$, and so $\mathbb{E}[\tau] = \Omega(N/B)$. □

This proposition will be very useful to show that spreading the budget of stubborn nodes uniformly over the lattice is actually an optimal policy for $d \geq 3$. Indeed, we will show that the lower bound (N/B) we have just derived is tight for $d \geq 3$ (as it is reached when we spread the budget uniformly over the lattice), while the lower bound $(N/B)^{2/d}$ obtained in Proposition 8.4.1 is reached for $d = 1$. The optimal placement of stubborn nodes when $d = 2$ will be left open, but we will conjecture that the expected hitting time τ scales as $(N/B)\log(N/B)$ if we place our stubborn nodes in an optimal way.

8.5 Effect of Placing all Resources on the Boundary

In this section we consider placing all our budget B over the boundary of the lattice. We first study the cases $B \sim O(N), O(N^{\frac{d-1}{d}})$ and $O(1)$, before providing upper and lower bounds for more general budgets.

8.5.1 Linear Budget

Given a linear budget $B = O(N)$, we can place all resources on the boundary and the rest inside the lattice. We assume that we possess a budget $B = \alpha N$ for some constant $\alpha \in (0, 1]$.

An order optimal placement of the nodes would be to position stubborn nodes over all the faces of the lattice and then spread the rest of the budget $O(N(\alpha - N^{-1/d})) = O(\alpha N)$ uniformly inside (Fig. 8-2a). Indeed, with such a placement a non-stubborn node will be contained in a hypercube of side $O((1/\alpha)^{1/d})$ in which we have a stubborn node at each

corner. By periodicity of the lattice, it is similar to computing the hitting time for a random walk over the periodic lattice $\mathbb{Z}^d_{(\frac{1}{\alpha})^{1/d}}$. There is no dependency on N here, so $\tau = \Theta(1)$ (with a possible dependency on d). This placement is thus order optimal.

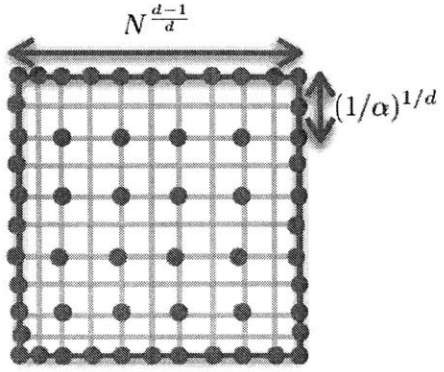
We can also consider a sub-optimal placement of the nodes: start by placing the stubborn nodes over all faces of the lattice, and then cover as many adjacent parallel internal faces as possible (Fig. 8-2b). We obtain a lattice filled with stubborn nodes but with a hole of side $O(N^{\frac{d-1}{d}})$. As shown in Lemma. 8.3.1, $\tau = \Theta(N^{2\frac{d-1}{d}})$ which is clearly sub-optimal.

Note in the last case however that, if instead of piling up all the stubborn nodes in adjacent parallel faces we space the stubborn faces by a distance of order $O((1/\alpha))$ (see Figure 8-2c), then a random walk hits the stubborn set in constant time $\Theta(1)$ (with a possible dependency on d though).

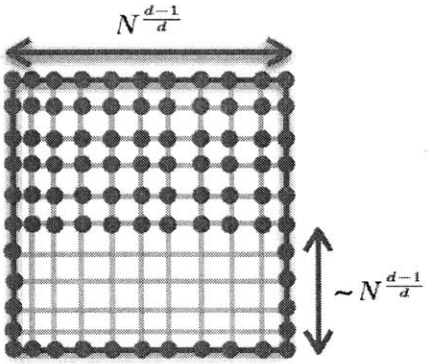
8.5.2 Stubborn Boundary Budget

Given a budget of the form $B \sim \Theta(N^{\frac{d-1}{d}})$, we can place stubborn nodes over all the faces of the lattice. Then using Lemma. 8.3.1, we immediately obtain that, for any fixed d, $\tau = \Theta(N^{2/d}) = \Theta((\frac{N}{B})^2)$ because the side of our lattice has size $N^{1/d}$.

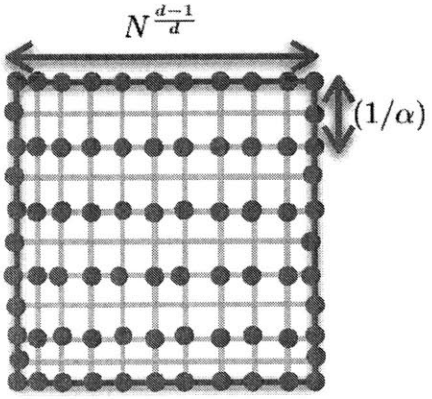
Remark. Note that we do not need to cover all the faces of the lattice: if we cover only one face, the random walk will hit the stubborn set equally fast. Indeed, by symmetry we can consider the first face as a single state "1", and all internal slices which are parallel to this face will be represented by states "2", ..., "$N^{1/d}$". We define a random walk Y_k over this new graph: we get a lazy birth-death chain in which $\mathbb{P}(Y_{k+1} = j | Y_k = i) = \frac{1}{4}$, if $|j - i| = 1$, and $\mathbb{P}(Y_{k+1} = i | Y_k = i) = \frac{1}{2}$ (assume state $N^{1/d} + 1$ and state 1 are identical). We can even modify this chain to be absorbing at state 1: $\mathbb{P}(Y_{k+1} = 1 | Y_k = 1) = 1$. A random walk over this lazy birth-death chain will hit the absorbing state in time $\Theta(N^{2/d})$ as well. Indeed, this can be seen by looking at a one-dimensional random walk on $\mathbb{Z}$ defined by $S_n = \sum_{i=0}^{n} X_i$, with $X_0 = 0$ and $X_i \in \{-1, 0, 1\}$ such that $\mathbb{P}(X_i = 1) = \mathbb{P}(X_i = -1) = \frac{1}{2}\mathbb{P}(X_i = 0)$. In this case, we consider the martingale $Y_n = S_n^2 - \frac{n}{2}$ and the stopping time $T_N = \min\{n \geq 1 : |S_n| = N^{1/d}\}$. Doob's theorem implies that $\mathbb{E}[T_N] \sim N^{2/d}$ (the idea is very similar to the proof of Lemma 8.3.1).



(a) An order optimal placement of stubborn nodes.



(b) A sub-optimal placement of stubborn nodes.



(c) An other order optimal placement of stubborn nodes.

Figure 8-2: Three different placements of roughly $N/2$ stubborn nodes over the lattice. In (a) a random walk hits the set of stubborn nodes in constant time (i.e., the hitting time is independent of N). In (b) the hitting time is of order $\Theta(N^{2\frac{d-1}{d}})$. In (c) the hitting time is of order $\Theta(1)$.

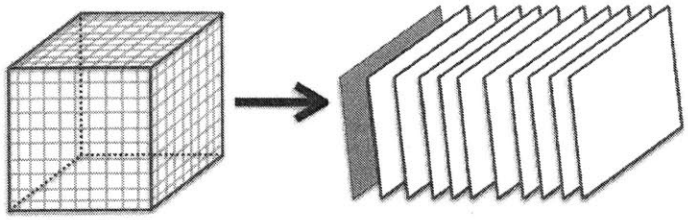


Figure 8-3: We illustrate an easy way to decompose the lattice if only one face (in green) is covered by stubborn nodes.

8.5.3 Constant Budget

We now consider a budget B independent of the number of nodes in the lattice (i.e., $B \sim O(1)$, for a fixed d). For example, this can be pictured as placing the stubborn nodes at the corners of the lattice. We will see in Section 8.6 that, for a lattice of side $N^{1/d}$ with stubborn nodes at its corners, the hitting time is:

$$\tau = \begin{cases} \Theta(N^2), & \text{, if } d = 1, \\ \Theta(N \log N), & \text{, if } d = 2, \\ \Theta(N), & \text{, if } d \geq 3. \end{cases}$$

8.5.4 General Budget

We finish this section by considering a general budget placed over the boundary of the lattice. We consider two cases: in the first one, we assume that our budget is at least as large as the size of the boundary (i.e., we are able to fully cover the boundaries of the lattice with stubborn nodes). In the second case, we look at budgets that are smaller than the size of the boundary (i.e., we are not able to fully cover the boundaries of the lattice with stubborn nodes).

Case1: Large Budget $B = \Omega(N^{\frac{d-1}{d}})$

If the budget is $B = \Omega(N^{\frac{d-1}{d}})$ then we can place the stubborn nodes over all faces of the lattice, and add more nodes inside the lattice. Already, with a boundary fully covered by stubborn nodes, we have $\tau = O(N^{2/d})$. Assume that the budget is of the form $B = \alpha N^{\frac{d-1}{d}}$

for some α such that $1 \lesssim \alpha \lesssim N^{1/d}$. Then we can reduce the hitting time τ by covering α internal slices of the lattice separated by a distance $\frac{N^{1/d}}{\alpha}$ from each other (see Figure 8-4). We have thus divided our lattice into many sublattices of side $\frac{N^{1/d}}{\alpha}$. Therefore this placement of stubborn nodes produces a hitting time $\tau = \Theta\left(\left(\frac{N}{B}\right)^2\right)$. In the next section, we will prove in Theorem 8.6.5 that spreading the resources uniformly over the lattice can reduce the hitting time to $\tau = \Theta\left(\frac{N}{B}\right)$ when $d \geq 3$: therefore spreading the resources over the lattice is strictly better than grouping resources over faces of the lattice.

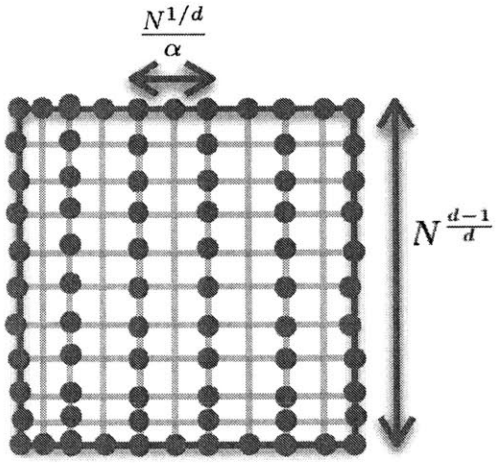


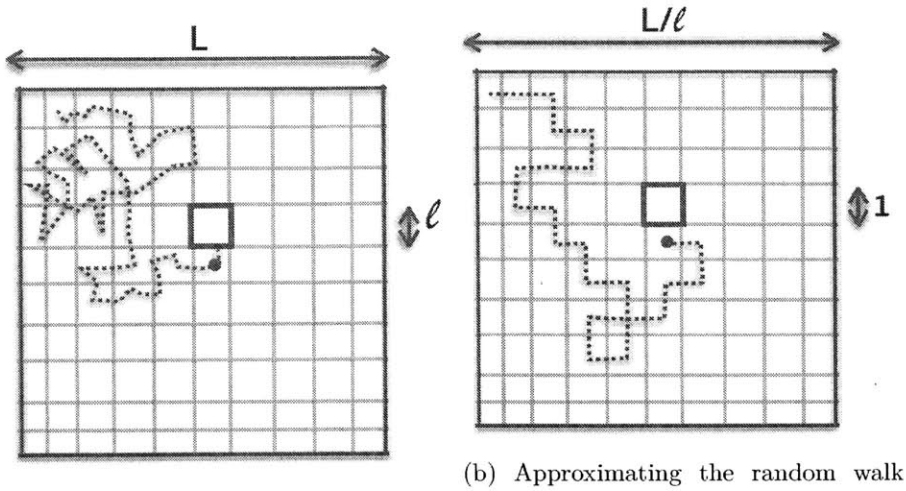
Figure 8-4: We illustrate a way to place the stubborn nodes for a budget of $B = \Omega(N^{\frac{d-1}{d}})$ nodes. In this case, $\tau = \Theta\left(\left(\frac{N}{B}\right)^2\right)$.

Case 2: Small Budget $B = o(N^{\frac{d-1}{d}})$

When the budget is smaller than the size of the boundary, we do not have a rigorous result. Instead we present an argument that results in the estimate:

$$\tau \lesssim \begin{cases} B^2 \vee (N-B)^2, & \text{if } d = 1, \\ B^2 \vee N \log\left(\frac{N}{B^2}\right), & \text{if } d = 2, \\ B^{\frac{2}{d-1}} \vee \frac{N}{B} B^{\frac{1}{d-1}}, & \text{if } d \geq 3. \end{cases} \tag{8.4}$$

If the budget is $B = o(N^{\frac{d-1}{d}})$, then we can place all the budget over the faces of some smaller cube of side $l = B^{\frac{1}{d-1}}$. We now attempt to estimate an upper bound for the hitting



(a) Original random walk over the lattice of side L. This random walk halts as soon as it hits the green cube of side l.

(b) Approximating the random walk over the lattice by a delayed symmetric random walk on a new lattice of side $L' = L/l$.

Figure 8-5: In (a), we consider a random walk over a periodic lattice of side L which halts as soon as it hits the cube of side l. In (b), we approximate this random walk by a symmetric random walk over a lattice of side $L' = L/l$ halting when the walk hits the unit cube.

time to the cube. Define S_t as the original random walk over the lattice of side $L = N^{1/d}$, as represented in Fig. 8-5a. S_t halts as soon as it hits the green cube of side l. Next, we define a random walk S'_t over a new lattice of side $L' = L/l$ which halts as soon as it hits a cube of side 1. We couple S_t and S'_t as follows: we partition the original lattice into cubes of side l as shown in Fig. 8-5a; while S_t is inside such a cube, the walk S'_t stays at the center of the corresponding cube in its own lattice space. As soon as S_t crosses a boundary, S'_t moves by one unit in the direction of the crossing S_n performed. S'_n then performs a delayed random walk over the primed lattice: the expected time between successive jumps in S'_t is of order $O(l^2)$ (it is the expected time for S_t to exit the cube of side l). Unfortunately the primed random walk is no longer symmetric because, at each jump, S'_t has a much higher probability to move back at its previous position. If we approximate S'_t as a symmetric random walk (by assuming l to be small compared to L), this reduces the problem to computing the expected hitting time of a symmetric random walk to a point: as we will see in the next section (Proposition 8.6.3), this scales as L'^d when $d \geq 3$ and $L'^2 \log L'$ if $d = 2$. Based on

this heuristic argument, we are led to conjecture that:

$$\begin{aligned}
\tau &\approx \begin{cases} O\left(l^2 \times L'^2 \log L'\right), & \text{if } d = 2, \\ O\left(l^2 \times L'^d\right), & \text{if } d \geq 3. \end{cases} \\
&\approx \begin{cases} O\left(L^2 \log\left(\frac{L}{l}\right)\right), & \text{if } d = 2, \\ O\left(\frac{L^d}{l^{d-2}}\right), & \text{if } d \geq 3. \end{cases} \\
&\approx \begin{cases} O\left(N \log\left(\frac{N}{B^2}\right)\right), & \text{if } d = 2, \\ O\left(\frac{N}{B} B^{\frac{1}{d-1}}\right), & \text{if } d \geq 3. \end{cases}
\end{aligned}$$

Note that if the green square gets bigger, then we must also consider the hitting time of a random walk inside the green square: this takes time $O(B^{\frac{2}{d-1}})$. Thus

$$\tau \approx \begin{cases} O\left(B^2 \vee N \log\left(\frac{N}{B^2}\right)\right), & \text{if } d = 2, \\ O\left(B^{\frac{2}{d-1}} \vee \frac{N}{B} B^{\frac{1}{d-1}}\right), & \text{if } d \geq 3. \end{cases} \tag{8.5}$$

where $a \vee b$ is defined as $\max(a, b)$. This bound has not been rigorously derived; in Section 8.8 we derive the same upper bounds by performing a continuous approximation to the random walk. Note that if $d = 1$, then $\tau = \Theta\left(B^2 \vee (N - B)^2\right)$.

8.6 Effect of Spreading all Resources Uniformly Inside the Lattice

In this section we consider spreading all our budget B uniformly inside the lattice. The spacing used will be $s = \left(\frac{N}{B}\right)^{1/d} = N^{\frac{1}{d^2}}$ (ignoring the possible $o(1)$ correction which insures the integrality of s). We first study the cases $B \sim \Theta(N), \Theta(N^{\frac{d-1}{d}})$ and $\Theta(1)$, before providing upper and lower bounds for more general budgets.

8.6.1 Linear Budget

If we use a budget $B \sim \Theta(N)$, then obviously $\mathbb{E}[\tau] = \Theta(1)$. Note that, if we use the result of the following subsection, we can prove that for $B = \frac{N}{\alpha}$ (for some $\alpha > 1$) and d fixed, we

have

$$\tau = \begin{cases} \Theta(\alpha^2), & \text{if } d = 1, \\ \Theta(\alpha \log \alpha), & \text{if } d = 2, \\ \Theta(\alpha), & \text{if } d \geq 3. \end{cases}$$

8.6.2 Stubborn Boundary Budget

We will now bound the expected hitting time for a budget $B = \Theta(N^{\frac{d-1}{d}})$ when we spread all the stubborn nodes uniformly over the lattice. Recall that the value of the spacing $s = \Theta\left(\left(\frac{N}{B}\right)^{1/d}\right)$. In this case, by applying the lower bound theorem (Proposition 8.4.1), we obtain a lower bound of: $\tau = \Omega\left(N^{\frac{2}{d2}}\right)$. By applying the finer lower bound theorem (Proposition 8.4.3), we see that $\tau = \Omega\left(N^{\frac{1}{d}}\right)$.

Lemma 8.6.1. *Set $d = 2$, let $B \sim \Theta(\sqrt{N})$ and spread all the stubborn nodes over the lattice so that the spacing between stubborn nodes will be $s = \Theta(N^{\frac{1}{4}})$. Then $s^2 \lesssim \tau \lesssim s^3$, i.e. $N^{\frac{1}{2}} \lesssim \tau \lesssim N^{\frac{3}{4}}$.*

Proof. (Sketch) We consider the backwards random walk $\mathbf{X_t} = (X_t, Y_t)$ initialized at node (x_0, y_0). The walk halts as soon as X_t or Y_t reach a stubborn node. As the boundary conditions are periodic, it is sufficient to analyze the halting time of a random walk over one square of side $s = N^{\frac{1}{4}}$ with one stubborn node at each corner.

Viewing $\mathbf{X_t}$ as a product chain, we see that X_t and Y_t are generating a random walk over a one-dimensional line of size s. We immediately get the lower bound $\tau(s) = \Omega(s^2)$ as each one-dimensional random walk takes time $\Omega(s^2)$ to reach a stubborn node at the endpoints, and we need both random walks to hit their target simultaneously.

To get the upper bound, we argue as follows. We can first simplify the analysis by viewing X_t and Y_t as random walks on a cycle $\mathbb{Z}_s$ of length s which halt at soon as they simultaneously hit node zero. We know X_t reaches zero in time $O(s^2)$. By Levin and Peres [ref:mixing times textbook -5.3.1], the mixing time for a random walk over a cycle of length s is of order $O(s^2)$. This result is also proved by Aldous and Finn [relaxation time τ_2 for the cycle is $O(s^2)$]. This means that, after time $O(s^2)$, Y_t is uniformly randomized over the cycle, so Y_t is at node zero with probability $\frac{1}{s}$. So we expect about s trials for Y_t to reach

node zero. Hence $\tau(s) = O(s^2) \times O(s) = O(s^3)$. □

Theorem 8.6.2. *For $d \geq 1$, let $B \sim O(N^{\frac{d-1}{d}})$ and spread all the stubborn nodes over the lattice so that the spacing between stubborn nodes will be $s = \Theta\left(N^{\frac{1}{d^2}}\right)$. Then $\tau(s) = O(s^{d+1})$. Furthermore, $N^{\frac{1}{d}} \lesssim \tau(N) \lesssim N^{\frac{1}{d}} \times N^{\frac{1}{d^2}}$.*

Proof. We consider again the backwards random walk $\mathbf{X_t} = (X_t^1, \ldots, X_t^d)$ initialized at node $(x_0^1, \ldots, x_0^d)$. The walk halts as soon as an X_t^i reaches a stubborn node. As the boundary conditions are periodic, it is sufficient to analyze the halting time of a random walk over one $d-$dimensional lattice of side $s = N^{\frac{1}{d^2}}$ with one stubborn node at each corner.

Viewing $\mathbf{X_t} = (X_t^1, \ldots, X_t^d)$ as a product chain, we see that the X_t^i are generating a random walk over a one-dimensional line of size s. We immediately get the lower bound $\tau(s) = \Omega(s^2)$ as each one-dimensional random walk takes time $\Omega(s^2)$ to reach a stubborn node at the endpoints. and we need all random walks to hit their target simultaneously. Expressing this lower bound as a function of N, we get $s^2 = N^{\frac{2}{d^2}}$. To compute a better lower bound in the entire lattice, we use the finer lower bound theorem (Proposition 8.4.3) we derived earlier. This yields a lower bound of $N/B = N^{\frac{1}{d}}$.

To get the upper bound, we argue as follows. We can first simplify the analysis by viewing the X_t^i as a random walk on a cycle $\mathbb{Z}_s$ of length s which halts at soon as all random walks $X_t^1, \ldots, X_t^d$ simultaneously hit node zero. We know X_t^1 reaches zero in time $O(s^2)$. By Levin and Peres [ref:mixing times textbook -5.3.1], the mixing time for a random walk over a cycle of length s is of order $O(s^2)$. This result is also proved by Aldous and Finn [relaxation time τ_2 for the cycle is $O(s^2)$]. This means that after time $O(s^2)$, the other X_t^i are uniformly randomized over the cycle, so all the remaining X_t^i are at node zero with probability $\frac{1}{s^{d-1}}$. So we expect about s^{d-1} trials for the remaining X_t^i to reach node zero. Hence $\tau(s) = O(s^2) \times O(s^{d-1}) = O(s^{d+1})$.

□

Until now, we were able to show that

$$1 \lesssim \frac{\tau}{N^{1/d}} \lesssim N^{1/d^2}.$$

It is actually possible to find a tighter asymptotic bound. Such a theorem can be found in Chapter 10 of Levin, Peres and Wilmer [11] (Chapter 10):

Proposition 8.6.3. *(Levin, Peres and Wilmer [11]) Let x and y be two points at distance $k \geq 1$ in the torus $\mathbb{Z}_l^d$. Let τ_{xy} be the expected time of first visit to y, starting from x. Then*

$$\tau_{xy} = \begin{cases} \Theta(l^2 \log k), & \text{if } d = 2, \\ \Theta(l^d), & \text{, independently of } k \text{ if } d \geq 3. \end{cases}$$

Corollary 8.6.4. *Consider a random walk over a lattice $\mathbb{Z}_{N^{1/d}}^d$ of N nodes with $B = O(N^{\frac{d-1}{d}})$ stubborn nodes evenly spread over the lattice. Then, in the worst case (over initial state) scenario, the expected time at which the random walk hits a stubborn node for the first time is given by:*

$$\tau = \begin{cases} \Theta(N^2), & \text{if } d = 1, \\ \Theta(\sqrt{N} \log N), & \text{if } d = 2, \\ \Theta(N^{1/d}), & \text{if } d \geq 3. \end{cases}$$

Proof. This immediately follows from Proposition 8.6.3 by taking $l = s$, i.e., $l = N^{1/d^2}$ and by setting $k = l$ to get the worst case scenario. Note that Proposition 8.6.3 is about hitting a single node. Since we spread our budget evenly over the lattice, then studying the whole lattice of N nodes with B stubborn nodes evenly spread is equivalent to looking at the small periodic lattice of side s with a single stubborn node at its corner. □

If we compare our previous bounds to the ones derived by Levin, Peres and Wilmer [11], we see that our lower bound was actually providing the correct asymptotic behavior when $d \geq 3$, and our upper bound was providing the correct asymptotic behavior when $d = 1$. The case $d = 2$ which is in between is the trickiest.

8.6.3 Constant Budget

When we have a constant budget $B \sim O(1)$, we can bound the expected hitting time τ using Peres' theorem and get:

$$\tau = \begin{cases} \Theta(N^2), & \text{if } d = 1, \\ \Theta(N \log N), & \text{if } d = 2, \\ \Theta(N), & \text{if } d \geq 3. \end{cases}$$

8.6.4 General Budget Evenly Spread over the Lattice

In this section, we consider the case of a general budget B, when we spread all the stubborn nodes evenly along the lattice. We therefore partition our lattice into squares of side $s = \Theta\left(\left(\frac{N}{B}\right)^{1/d}\right)$ with a stubborn node at each corner. We can therefore apply results of the previous section to obtain:

Theorem 8.6.5. *In dimension $d \geq 1$, if we spread our budget B of stubborn nodes evenly along the lattice, then*

$$\frac{N}{B} \lesssim \tau(N) \lesssim \left(\frac{N}{B}\right)^{1+1/d}.$$

Using the bounds from Peres, we get

$$\tau = \begin{cases} \Theta((\frac{N}{B})^2), & \textit{if } d = 1, \\ \Theta(\frac{N}{B} \log \frac{N}{B}), & \textit{if } d = 2, \\ \Theta(\frac{N}{B}), & \textit{if } d \geq 3. \end{cases}$$

Proof. Partition the lattice into squares of side $s = \Theta\left(\left(\frac{N}{B}\right)^{1/d}\right)$ with a stubborn node at each corner and apply the bounds we derived in the Theorem 8.6.2 and Corollary 8.6.4. □

8.7 Optimal Placement of Stubborn nodes

Now that we have bounded the expected hitting times for uniformly spread stubborn nodes, we can show that this placement is actually optimal for $d \neq 2$ and conjecture that it still is

optimal when $d = 2$. In this section, we say that a placement of stubborn nodes is optimal if it minimizes the worst case expected hitting time from a non-stubborn node to the set of stubborn nodes.

Lemma 8.7.1. *Let $d = 1$. Given a budget B of stubborn nodes, an order optimal placement (which minimizes the worst case scenario, over initial state, of the expected hitting time) for the stubborn nodes consists in spreading them evenly at a distance $s = O(N/B)$ of each other. Therefore the optimal hitting time is $\tau = (N/B)^2$.*

Proof. If we spread evenly the budget over the line, stubborn nodes will be at a distance $O(N/B)$ of each other, therefore the hitting time will be equal to $(N/B)^2$. We proved in Theorem 8.6.5 that $\tau = \Omega\left(\left(\frac{N}{B}\right)^{\frac{2}{d}}\right) = \Omega\left(\left(\frac{N}{B}\right)^2\right)$. So the hitting time cannot be lower than $(N/B)^2$ in one dimension. We conclude that this placement is indeed optimal. □

Note. We can also prove the optimality of the evenly spread placement when $d = 1$ by arguing in the following way. Suppose that we place the stubborn nodes along the line at distances $l_1, l_2, \ldots, l_{B-1}$ of each other (where $0 \leq l_k \leq N$). Then $\tau = \max_{i=1\ldots B-1} \tau_i$, where τ_i is the expected hitting time over the segment of length l_i which has stubborn nodes i and $i+1$ for endpoints. We know that $\tau_i = l_i^2$, therefore $\tau = \max_{i=1\ldots B-1} l_i^2$ for any placement of the stubborn nodes over the line. We conclude that the optimal placement of the stubborn nodes i.e., the placement that minimizes the expected hitting time, should minimize the largest possible distance between consecutive stubborn nodes. This implies that an optimal placement consists of spreading evenly all the stubborn nodes over the line.

We can now provide the main result of the chapter:

Theorem 8.7.2. *Let $d \geq 1$ and $d \neq 2$. Given a budget B of stubborn nodes, an order optimal placement for the stubborn nodes over the lattice $Z^d_{N^{1/d}}$ consists in spreading them evenly at a distance $s = O((N/B)^{1/d})$ of each other. The expected hitting time for a random walk is given by:*

$$\tau = \begin{cases} \Theta((\frac{N}{B})^2), & \text{if } d = 1, \\ \Theta(\frac{N}{B}), & \text{if } d \geq 3. \end{cases}$$

Proof. We have proved the theorem for $d = 1$. Now for $d \geq 3$, we already know that the hitting time is bounded below by (N/B) (Proposition 8.4.3). Since the lower bound is reached when we spread the resources evenly over the lattice (Theorem 8.6.5), we conclude that such a policy is indeed optimal. □

Conjecture 8.7.3. *We believe that for $d = 2$, spreading the resources evenly over the lattice will be an order optimal policy. We already know from Theorem 8.6.5 that the hitting time for such a random walk would be, in expectation, $\tau = \Theta(\frac{N}{B} \log \frac{N}{B})$.*

The main idea is to avoid wasting resources, by avoiding to create large clumps of stubborn nodes. Indeed, a square of side l filled with stubborn nodes will have the same effect as an empty square of side l with stubborn nodes only on the boundary, as long as l is not too large (the time l^2 it takes to hit the boundary of the square from the inside of the square should be smaller than the time it takes to hit the boundary of the square from the outside of the square). Indeed, in that case a filled square uses the $d-$th power of the budget for a comparable hitting time.

8.8 Another Attempt at Computing the Hitting Time to a Point

In this section, we will once again consider the hitting time to a point for a symmetric random walk over a $d-$dimensional lattice of size s (with s^d nodes).

Fact. *For $d = 2$, suppose we have one stubborn node in a corner of the $2-$dimensional periodic lattice of side s. Then $\tau(s) \sim O(s^2 \log(s^2))$.*

This result is already known, and can be found in Levin, Peres and Wilmer [11] (see Theorem 8.6.3). The proof in [11] relies on circuit analogies, and relies on Pólya urn processes. In the following, we want to offer a plausible argument for this fact by performing a continuous approximation of the problem and by solving PDEs with appropriate boundary conditions. The "proof" attempted below is not rigorous and we will highlight the missing steps.

Proof. Attempt. We will first derive a recursion equation in the lattice space for the values of the hitting times to the targeted point (initialized at various nodes in the lattice). Then we will look at a continuous approximation to these equations. This step is often performed and can be rigorously justified. So far, we would have to solve a PDE over a square. Due to the symmetry of the problem, it is much easier to solve the PDE over a disk: this is exactly where the non-rigorous part of the proof lies. We would need to prove that, a Brownian motion initialized on the boundary of a square would hit a "small" target in its center at roughly the same time as a Brownian motion initialized on the boundary of a disk of comparable size, in orders of magnitude of the lattice size s.

Step 1: recursive relation

As before, we focus on a random walk over a periodic square of side s which has a stubborn node at each corner. Define $T(k,k')$ to be the expected time to reach a corner if the random walk is initialized at node (k,k'). By periodicity of the boundaries, we see that $T(k,k') = T(-k,k') = T(d-k,k')$ and by symmetry we see that $T(k,k') = T(k',k)$.

We obtain the following recursive equation:

$$\begin{aligned} T(k,k') &= 1 + \frac{1}{4}\Big(T(k-1,k') + T(k+1,k') + T(k,k'-1) + T(k,k'+1)\Big) \\ T(0,0) &= 0 = T(d,0) = T(0,d) = T(d,d) \end{aligned}$$

$$\begin{aligned} T(k,k') &= T(-k,k') = T(d-k,k') \\ T(k,k') &= T(k',k) \end{aligned}$$

We are interested in computing $T_{mid} = T(\lfloor \frac{d}{2} \rfloor, \lfloor \frac{d}{2} \rfloor)$ to obtain an upper bound for the expected time to reach the stubborn node if we start in the middle of a square. This recursion is very hard to solve.

Step 2: continuous approximation

We will convert the square lattice of side s to a disk of unit area. To do this conversion, we must scale distances as follows: the lattice has area s^2 and the disk has area 1, therefore 1 unit in the lattice corresponds to $\frac{1}{s}$ units on the disk. To scale time correctly, recall that we expect to reach a boundary in time s^2 on the lattice, while on the disk the boundary is reached in unit time, therefore 1 time unit on the lattice process corresponds to $1/s^s$ time units on the disk. To avoid confusion, we will use the notations $\hat{T}(\hat{x},\hat{y})$ or $\hat{T}(\hat{r})$ over the disk (where $0 \leq \hat{r} = \hat{x}^2 + \hat{y}^2 \leq 1$). Thus we have the following conversions:

$$\hat{x} = x/s, \hat{y} = y/s, \text{ and}$$

$$\hat{T}(\hat{x},\hat{y}) = \frac{1}{s^2} T(x/s, y/s).$$

Over the disk, we can view the process as a diffusion by approximating the recursion equations derived above.

$$T(k,k') = 1 + \frac{1}{4}\Big(T(k-1,k') + T(k+1,k') + T(k,k'-1) + T(k,k'+1)\Big)$$

becomes

$$\begin{aligned}
\hat{T}(\hat{x},\hat{y}) &= \frac{1}{s^2} T\left(\frac{x}{s}, \frac{y}{s}\right) \\
&= \frac{1}{s^2} + \frac{1}{4s^2}\left(T\left(\frac{x-1}{s}, \frac{y}{s}\right) + T\left(\frac{x+1}{s}, \frac{y}{s}\right) + T\left(\frac{x}{s}, \frac{y-1}{s}\right) + T\left(\frac{x}{s}, \frac{y+1}{s}\right)\right) \\
&= \frac{1}{s^2} + \frac{1}{4}\left(\hat{T}\left(\hat{x} - \frac{1}{s}, \hat{y}\right) + \hat{T}\left(\hat{x} + \frac{1}{s}, \hat{y}\right) + \hat{T}\left(\hat{x}, \hat{y} - \frac{1}{s}\right) + \hat{T}\left(\hat{x}, \hat{y} + \frac{1}{s}\right)\right).
\end{aligned}$$

We thus obtain:

$$-\frac{1}{s^2} =$$

$$= \frac{1}{4}\left(\hat{T}\left(\hat{x}-\frac{1}{s},\hat{y}\right)+\hat{T}\left(\hat{x}+\frac{1}{s},\hat{y}\right)-2\hat{T}\left(\hat{x},\hat{y}\right)+\hat{T}\left(\hat{x},\hat{y}-\frac{1}{s}\right)+\hat{T}\left(\hat{x},\hat{y}+\frac{1}{s}\right)-2\hat{T}\left(\hat{x},\hat{y}\right)\right)$$

$$= \frac{1}{4s^2}\left(\frac{\hat{T}\left(\hat{x}-\frac{1}{s},\hat{y}\right)+\hat{T}\left(\hat{x}+\frac{1}{s},\hat{y}\right)-2\hat{T}\left(\hat{x},\hat{y}\right)}{1/s^2}+\frac{\hat{T}\left(\hat{x},\hat{y}-\frac{1}{s}\right)+\hat{T}\left(\hat{x},\hat{y}+\frac{1}{s}\right)-2\hat{T}\left(\hat{x},\hat{y}\right)}{1/s^2}\right).$$

Rearranging the terms, we write

$$-4 = \frac{\hat{T}\left(\hat{x}-\frac{1}{s},\hat{y}\right)+\hat{T}\left(\hat{x}+\frac{1}{s},\hat{y}\right)-2\hat{T}\left(\hat{x},\hat{y}\right)}{1/s^2}+\frac{\hat{T}\left(\hat{x},\hat{y}-\frac{1}{s}\right)+\hat{T}\left(\hat{x},\hat{y}+\frac{1}{s}\right)-2\hat{T}\left(\hat{x},\hat{y}\right)}{1/s^2}.$$

Taking the limit as s goes to infinity, we get:

$$\frac{\partial^2}{\partial\hat{x}^2}\hat{T}(\hat{x},\hat{y})+\frac{\partial^2}{\partial\hat{y}^2}\hat{T}(\hat{x},\hat{y})=-4 \tag{8.6}$$

Writing the Laplacian in polar coordinates and using spherical symmetry, we get the following ODE for $\hat{T}(\hat{r})$: $\hat{T}''(\hat{r})+\frac{1}{\hat{r}}\hat{T}'(\hat{r})=-4$, which implies that

$$\hat{T}(\hat{r})=-\hat{r}^2+\alpha\log(\hat{r})+\beta, \tag{8.7}$$

for some constants α,β.

On a disk of radius $\hat{r}_0<1$, we have $\hat{T}(\hat{r}_0)=0$, implying that

$$\beta=\hat{r}_0^2-\alpha\log(\hat{r}_0).$$

The $\hat{r}_0$ we will use will be of order $1/s$ since $T(r)=0$ for $r<1$ over the lattice.

We know that $\hat{T}(1)=dr^2+\frac{1}{2}\hat{T}(1)+\frac{1}{4}\hat{T}(1+dr)+\frac{1}{4}\hat{T}(1-dr)$ on the boundary. Due to spherical symmetry and by periodicity of the boundary conditions, we observe that $\hat{T}(1+dr)=\hat{T}(1-dr)$. Therefore $\hat{T}(1)=dr^2+\frac{1}{2}\hat{T}(1)+\frac{1}{2}\hat{T}(1-dr)$, i.e. $\hat{T}(1)-\hat{T}(1-dr)=2dr^2$. This implies that $\hat{T}'(1)=\lim_{dr\to 0}2dr=0$. We have $\hat{T}'(\hat{r})=-2\hat{r}+\alpha/\hat{r}$, therefore $0=\hat{T}'(1)=-2+\alpha$ so $\alpha=2$ and $\beta=\hat{r}_0^2-2\log(\hat{r}_0)$. We conclude that:

$$\hat{T}(\hat{r})=(\hat{r}_0^2-\hat{r}^2)+\log(\frac{\hat{r}^2}{\hat{r}_0^2}). \tag{8.8}$$

Step 3: conversion to the discrete space

If we discretize the result we see that $T(r) = s^2\hat{T}(r/s)$, therefore:

$$\begin{aligned} T(r) &= (s^2\hat{r}_0^2 - r^2) + s^2\log(\left(\frac{r}{s\hat{r}_0}\right)^2) && (8.9)\\ &= (r_0 - r^2) + s^2\log\left(\left(\frac{r}{r_0}\right)^2\right) && (8.10)\\ &= (1 - r^2) + s^2\log(r^2) && (8.11) \end{aligned}$$

since $r_0 \sim 1$.

The right hand side is of order $O\left(r^2 + s^2\log(r^2)\right)$. The highest value of T is reached at the middle of the lattice (so $r \approx s/2$). In that case:

$$T = \Theta(s^2\log(s^2)).$$

We conclude that the longest time needed to hit a point on a $2-$dimensional lattice is

$$T = \Theta(s^2\log(s^2)).$$

Now if we consider the hitting time to a square of side r_0 instead of a point, we get

$$T = \Theta\left(s^2\log\left(\left(\frac{s}{r_0}\right)^2\right)\right).$$

□

Note. By the same argument, in dimensions $d \geq 3$, we obtain: $\Delta\hat{T}(\vec{\hat{x}}) = -2d$, leading to

$$\hat{T}(\hat{r}) = -\hat{r}^2 + \alpha\frac{\hat{r}^{2-d}}{2-d} + \beta.$$

Using the same boundary conditions we obtain

$$\hat{T}(\hat{r}) = (\hat{r}_0^2 - \hat{r}^2) + \frac{2}{2-d}(\hat{r}_0^{2-d} - \hat{r}^{2-d}).$$

Converting the problem back to the original discrete lattice:

$$T(s) = (r_0^2 - s^2) + \frac{2}{2-d} s^d (s^{2-d} - r_0^{2-d}).$$

We conclude that:

$$T(s) = \frac{2}{2-d} s^d \left(\frac{1}{r_0^{d-2}} - \frac{1}{s^{d-2}}\right) - (s^2 - r_0^2). \tag{8.12}$$

Using the fact that $r_0 = 1$:

$$T(s) = \frac{2}{2-d}(s^d - s^2) - (s^2 - 1). \tag{8.13}$$

We therefore recover the result $T(s) = \Theta(s^d)$. We also observe that, for a more general $r_0 < s$, we have $T(s) = \Theta\left(\frac{s^d}{r_0^{d-2} - s^2}\right)$. In this case, if we consider the $d-$dimensional lattice of side $s = N^{1/d}$ with a budget B placed on the faces of a cube of side $r_0 = B^{\frac{1}{d-1}}$, we get

$$T(s) = \begin{cases} \Theta((\frac{N}{B})^2), & \text{if } d = 1, \\ \Theta\left(B^2 \vee N \log \frac{N}{B^{\frac{d}{d-1}}}\right), & \text{if } d = 2, \\ \Theta\left(B^{\frac{2}{d-1}} \vee \left(\frac{N}{B^{\frac{d-2}{d-1}}} - N^{2/d}\right)\right), & \text{if } d \geq 3. \end{cases}$$

where the first B^2 term comes from the hitting time inside the square, and where $a \vee b$ is defined as $\max(a, b)$.

$$T(s) = \begin{cases} \Theta((\frac{N}{B})^2), & \text{if } d = 1, \\ \Theta\left(B^2 \vee N \log \frac{N}{B^2}\right), & \text{if } d = 2, \\ \Theta\left(B^{\frac{2}{d-1}} \vee \frac{N}{B} B^{\frac{1}{d-1}}\right), & \text{if } d \geq 3. \end{cases} \tag{8.14}$$

Recall that B can take values between 1 and $N^{\frac{d-1}{d}}$: then when B is actually equal to $N^{\frac{d-1}{d}}$ (which corresponds to covering all faces of the lattice with stubborn nodes), the term $B^{\frac{2}{d-1}}$ dominates and yields $N^{\frac{2}{d}}$ in agreement with our results in Section 8.5.2. Furthermore, the result we conjecture here is consistent with the upper bound we estimated in Section 8.5.4 (Equation (8.5)).

Chapter 9

Convergence Time to Consensus for a Dynamic Policy

In Chapter 8, we considered a static model in which we placed B stubborn nodes with opinion $+1$ over a $d-$dimensional lattice $\mathbb{Z}^d_{N^{1/d}}$ of N nodes and waited for the network to reach a $1-$consensus. In this chapter, we consider a dynamic variation of the problem studied in Chapter 8: we are once again given a budget of B stubborn nodes with opinion $+1$ and a $d-$dimensional lattice $\mathbb{Z}^d_{N^{1/d}}$ of N nodes, but this time we are allowed to change the position of the stubborn nodes over the lattice during the run of the experiment. We want to bound the expected time needed for such a network to reach a $1-$consensus. Our ultimate goal is to see if we can significantly improve the convergence time to consensus by adopting dynamic policies instead of static policies.

This chapter is structured as follows: after having introduced the model used here, we consider dynamic policies for budgets at least as large as the size of the boundary of the lattice, and also dynamic policies for budgets smaller than the size of the boundary of the lattice. We then consider the case of a general budget, and we conclude the chapter by comparing the performance of static and dynamic policies.

9.1 The Model

We use the same model as in Chapter 8. We possess a budget B of stubborn nodes in the $+1$ state, and position these stubborn nodes on a $d-$dimensional lattice $\mathbb{Z}^d_{N^{1/d}}$ of N nodes. What is different is that we are allowed to reposition all our stubborn nodes over the lattice each time a single Poisson clock (independent of the Poisson clocks of the nodes in the lattice) of rate $\nu = 1$ ticks. Our goal is to bound the expected time τ^* for the network to reach a $1-$consensus. We assume throughout the chapter that all nodes are in the 0 state at time $t = 0$.

If a node was stubborn at time t, and if we decide not to select it as a stubborn node at a later time t', then the node will be in state $+1$ at time t' but will resume updating his opinion according to the Voter process (this means that the node is no longer protected and can adopt state 0 if one of its neighbors with opinion 0 influences him after time t').

On the other hand, if a non-stubborn node is in state 0 at time t and if we decide to make it stubborn at time t, the node will immediately adopt the opinion $+1$ at time t. Thus, in our dynamic model, we can essentially switch nodes from a 0 state to a $+1$ state at an average rate of B nodes per unit time.

In this Chapter, we study two cases: we first consider budgets B at least at large as the size of the boundary of the lattice (i.e., $B = \Omega\left(N^{\frac{d-1}{d}}\right)$) in Section 9.2, and then budgets B smaller than the size of the boundary of the lattice (i.e., $B = o\left(N^{\frac{d-1}{d}}\right)$) in Section 9.3. For simplicity, we will refer to the former case as the "large budget case", and the latter case as the "small budget case" throughout the whole chapter. We will consider the case of a general budget in Section 9.4.

Note that, in contrast with Chapter 8, we will directly use the Voter process (instead of using the dual process). This is because, when stubborn nodes are mobile, the dual formulation becomes much harder to analyze.

9.2 The Large Budget Case:

We begin by considering what we refer to as "large budgets", i.e., budgets B at least as large as the size of the boundary of the lattice. In this case, $B = \Omega\left(N^{\frac{d-1}{d}}\right)$. We describe below a policy according to which the network can reach a $1-$consensus in time $\Theta(N/B)$:

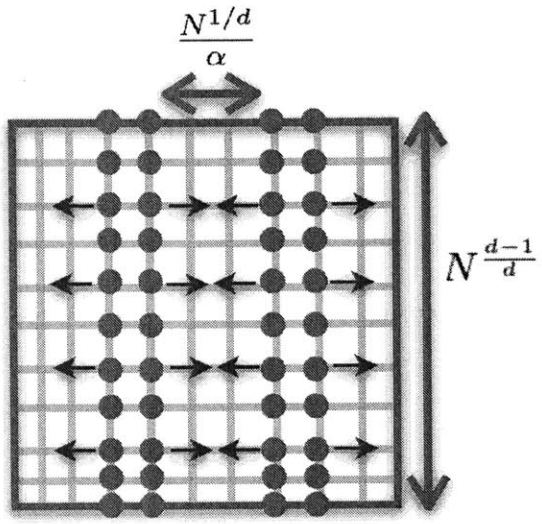


Figure 9-1: Our large-budget dynamic policy applied to a $2-$dimensional lattice of N nodes. Consensus is expected to be reached in time $\tau^* = \Theta(N/B)$.

A DYNAMIC POLICY FOR LARGE BUDGETS:

As $B = \Omega\left(N^{\frac{d-1}{d}}\right)$, we can assume that $B = \alpha \times N^{\frac{d-1}{d}}$ for some $\alpha = \Omega(1)$. As shown in Figure 9-1, we place our stubborn nodes over α slices (of $N^{\frac{d-1}{d}}$ nodes) of the lattice that are spaced at a distance $\frac{N^{1/d}}{\alpha}$ of each other. For simplicity, we actually cover two successive slices instead of one each time: this means that we have α pairs of adjacent slices that are spaced at a distance $\frac{N^{1/d}}{\alpha}$ of each other (see Figure 9-1). The policy then consists of moving the right slices to the right, and the left slices to the left, as shown in Figure 9-1.

In the following theorem we derive the expected time to consensus, when using the dynamic policy outlined above:

Theorem 9.2.1. *Given a budget* $B = \Omega\left(N^{\frac{d-1}{d}}\right)$, *the* $d-$*dimensional lattice* $\mathbb{Z}^d_{N^{1/d}}$ *of* N

nodes can reach a $1-$*consensus in time* $\tau^* = \Theta\left(\frac{N}{B}\right)$ *when using the dynamic policy for large budgets described above.*

Proof. Using the dynamic policy for large budgets described above, we observe that nodes can only transition from state 0 to state $+1$ (as the moving slices protect the nodes which have been set to the $+1$ state). Therefore we expect a $1-$consensus to be reached in time

$$\tau^* = \Theta\left(\frac{N^{1/d}}{\alpha}\right) = \Theta\left(\frac{N^{1/d}}{B/N^{1-(1/d)}}\right) = \Theta\left(\frac{N}{B}\right).$$

□

We will prove in Theorem 9.4.2 that the "dynamic policy for large budgets" is actually an order-optimal policy.

9.3 The Small Budget Case:

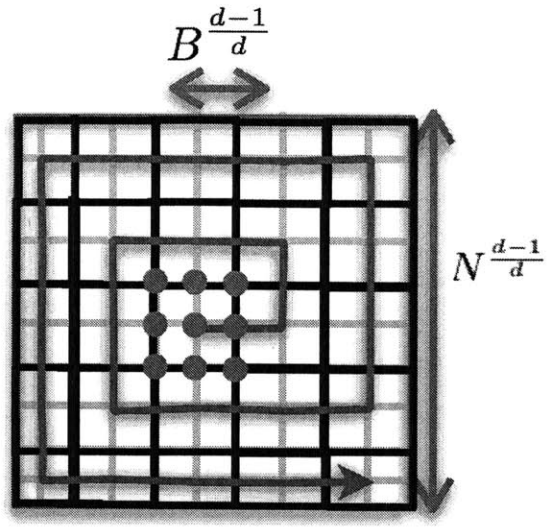


Figure 9-2: Our small-budget dynamic policy applied to a $2-$dimensional lattice of N nodes. Consensus is expected to be reached in time $\tau^* = O(N/B)$.

We now consider what we refer to as "small budgets", i.e., budgets B smaller than the size of the boundary of the lattice. In this case, $B = o\left(N^{\frac{d-1}{d}}\right)$. This case seems harder

to analyze as we cannot protect the set of +1 non-stubborn nodes anymore. We describe below a policy according to which the network can reach a 1−consensus in time $\Theta(N/B)$:

A DYNAMIC POLICY FOR SMALL BUDGETS:

Divide the lattice into (N/B) cubes of side $l = B^{1/d}$. We place our budget $B = o\left(N^{\frac{d-1}{d}}\right)$ over one of these cubes of side l: call this cube C_0. We then move our stubborn cube over neighboring cubes in the direction of a spiral centered at C_0, as shown in Figure 9-2. We also try to place stubborn nodes over +1 nodes that have returned to a 0 state along the way.

We will show in Theorem 9.4.2 that the "dynamic policy for small budgets" leads to a 1−consensus in expected time $\tau^* = \Theta\left(\frac{N}{B}\right)$, and that the policy is actually an order-optimal policy.

9.4 The General Budget Case:

We are now ready to prove the main result of the chapter: we show here that, for an order-optimal policy, $\tau^* = \Theta\left(\frac{N}{B}\right)$. Furthermore, we show that, in dimension $d \geq 2$, dynamic policies do not perform significantly better than static policies.

We begin by defining "non-static" policies.

Definition 9.4.1. We say that a policy is *non-static* if the policy is dynamic, and has the following property: if the controller's clock ticks at time t, and if there are k nodes with opinion 0 just before time t, then the set of stubborn nodes just after time t must include $\min\{k, B\}$ nodes whose opinion was 0 before time t.

We can now state and prove the main result of the chapter:

Theorem 9.4.2. *Given an arbitrary budget B, the $d-$dimensional lattice $\mathbb{Z}^d_{N^{1/d}}$ of N nodes can reach a $1-$consensus in time $\tau^* = \Theta\left(\frac{N}{B}\right)$ when using any non-static policy.*

Proof. Let $\{X_i\}_{i=1}^{\infty}$ be i.i.d. Exponential random variable with parameter 1. Define $K_i = \sum_{j=1}^{i} X_j$: then the random variable K_i represents the time at which the controller's Poisson

clock will tick for the i^{th} time. Define N_{K_i} as the number of nodes in the $+1$ state immediately after time K_i (so that this number also reflects the action the controller performed at time K_i). Observe that, while $N_{K_i} \leq N - B$, we have the following inequality:

$$B \leq \mathbb{E}[N_{K_{i+1}} - N_{K_i} | N_{K_i}] \leq B(2d+1)\,. \tag{9.1}$$

Indeed, in between two clock ticks, i.e., in the time interval $[K_i, K_{i+1})$, we expect:

1. Pairs of non stubborn nodes to influence each other with the same probability (as the lattice is a regular graph): in this case, the expected number of non-stubborn nodes with the $+1$ opinion that have not been influenced by a stubborn node should be constant.

2. Stubborn nodes can influence non stubborn nodes: in that case, the expected number of non-stubborn nodes with the $+1$ opinion that have been influenced by a stubborn node should lie in the interval $(0, 2dB)$.

3. The policy we use is non-static, therefore B new nodes with opinion 0 just before time K_{i+1} will take the $+1$ opinion just after time K_{i+1}.

Define $N'_{K_i} = N_{K_i} - BK_i$ and $N''_{K_i} = N_{K_i} - B(2d+1)K_i$. We show below that, while $N_{K_i} \leq N - B$, N'_{K_i} is a sub-martingale, and N''_{K_i} is a super-martingale. Indeed, using Inequality (9.1), we get:

$$\begin{aligned}
\mathbb{E}[N'_{K_{i+1}} | N'_{K_i}] &= \mathbb{E}[N_{K_{i+1}} - BK_{i+1} | N'_{K_i}] \\
&\geq \mathbb{E}[N_{K_i} + B - B(K_i + X_{i+1}) | N'_{K_i}] \\
&= \mathbb{E}[N'_{K_i} | N'_{K_i}] + B(1 - \mathbb{E}[X_{i+1}]) = N'_{K_i}\,,
\end{aligned}$$

as X_{i+1} is independent from N'_{K_i} and has mean 1; also,

$$\begin{aligned}
\mathbb{E}[N''_{K_{i+1}} | N''_{K_i}] &= \mathbb{E}[N_{K_{i+1}} - 2dBK_{i+1} | N''_{K_i}] \\
&\leq \mathbb{E}[N_{K_i} + 2dB - 2dB(K_i + X_{i+1}) | N''_{K_i}] \\
&= \mathbb{E}[N''_{K_i} | N''_{K_i}] + 2dB(1 - \mathbb{E}[X_{i+1}]) = N''_{K_i}\,,
\end{aligned}$$

as X_{i+1} is independent from N''_{K_i} and has mean 1.

Define the stopping time K_I as the smallest time K_I such that $N_{K_I} > N - B$. Applying Doob's Optional Stopping theorem to the sub-martingale N'_{K_i}, we get:

$$0 = \mathbb{E}[N'_0] \leq \mathbb{E}[N'_{K_I}] = \mathbb{E}[N_{K_I}] - B\mathbb{E}[K_I].$$

Therefore,

$$\mathbb{E}[K_I] \leq \frac{\mathbb{E}[N_{K_I}]}{B} \leq \frac{N}{B}.$$

Applying Doob's Optional Stopping theorem to the super-martingale N''_{K_i}, we get:

$$0 = \mathbb{E}[N''_0] \geq \mathbb{E}[N''_{K_I}] = \mathbb{E}[N_{K_I}] - B(\bar{d}+1)\mathbb{E}[K_I].$$

Therefore,

$$\mathbb{E}[K_I] \geq \frac{\mathbb{E}[N_{K_I}]}{2dB} > \frac{N-B}{2dB}.$$

Therefore,

$$\left(\frac{N}{B} - 1\right)\frac{1}{2d} \leq \mathbb{E}[K_I] \leq \frac{N}{B}, \tag{9.2}$$

and so $\mathbb{E}[K_I] = \Theta(N/B)$.

Now, between time K_I and time K_{I+1} (actually just before time K_{I+1}), the expected number of nodes in the $+1$ state is expected to remain greater than $(N-B)$. Then, at time K_{I+1} the controller will choose the remaining nodes as stubborn nodes, and the network will have reached a $1-$consensus. Therefore our expected consensus time τ^* should be equal to $1 + \mathbb{E}[K_{I+1}]$. We conclude that $\tau^* = \Theta\left(\frac{N}{B}\right)$ when using any non-static policy. □

Notice that Theorem 9.4.2 also holds for arbitrary "regular networks" (i.e., networks whose underlying graph is a regular graph).

Theorem 9.4.3. *Given an arbitrary budget B, a regular network of N nodes of degree r can reach a $1-$consensus in time $\tau^* = \Theta\left(\frac{N}{B}\right)$ when using any non-static policy.*

Proof. This theorem follows from the proof of Theorem 9.4.2, by taking replacing $2d$ by r. Since r is taken to be constant, the result follows. □

As a corollary of Theorem 9.4.2, we see that the "dynamic policy for small budgets" is actually order-optimal (and it also applies to budgets larger than the size of the boundary). We also see that the "dynamic policy for large budgets" is also order-optimal.

9.5 Is There an Advantage to being Dynamic ?

In this section, we compare the results obtained in Chapter 8 and Chapter 9.

If $d = 1$, we saw in Chapter 8 that static policies could reach a $1-$consensus in time τ^* such that:

$$\left(\frac{N}{B}\right)^2 \lesssim \tau^* \lesssim \left(\frac{N}{B}\right)^2 \log\left(N - B\right).$$

In Chapter 9, we can use the "dynamic policy for large budgets" (here we can use it even for small budgets too) to reach consensus in time

$$\tau^* = \Theta\left(\frac{N}{B}\right).$$

We clearly see that a dynamic policy can perform significantly better than a static policy when $d = 1$.

If $d = 2$, we saw in Chapter 8 that, by uniformly spreading the budget over the lattice, we could reach a $1-$consensus in time τ^* such that:

$$\left(\frac{N}{B}\right)\log\left(\frac{N}{B}\right) \lesssim \tau^* \lesssim \left(\frac{N}{B}\right)\log\left(\frac{N}{B}\right)\log\left(N - B\right).$$

We have proved that an order optimal policy should lead to an expected time to consensus that falls inside these bounds. In Chapter 9, we can use the "dynamic policy for large budgets" when $B = \Omega\left(N^{\frac{d-1}{d}}\right)$, or the "dynamic policy for small budgets" when $B = o\left(N^{\frac{d-1}{d}}\right)$ to reach consensus in time

$$\tau^* = \Theta\left(\frac{N}{B}\right).$$

From Theorem 9.4.2, we know that these dynamic policies are order-optimal: this implies

that a dynamic policy, for an arbitrary budget B, cannot perform better than a static policy by more than a $(\log N)^2$ factor.

If $d \geq 3$, we saw in Chapter 8 that, by uniformly spreading the budget over the lattice, we could reach a $1-$consensus in time τ^* such that:

$$\left(\frac{N}{B}\right) \lesssim \tau^* \lesssim \left(\frac{N}{B}\right) \log\left(N - B\right).$$

We have proved that an order optimal policy should lead to an expected time to consensus that falls inside these bounds. In Chapter 9, we can use the "dynamic policy for large budgets" when $B = \Omega\left(N^{\frac{d-1}{d}}\right)$, or the "dynamic policy for small budgets" when $B = o\left(N^{\frac{d-1}{d}}\right)$ to reach consensus in time

$$\tau^* = \Theta\left(\frac{N}{B}\right).$$

From Theorem 9.4.2, we know that these dynamic policies are order-optimal: this implies that a dynamic policy, for an arbitrary budget B, cannot perform better than a static policy by more than a $\log N$ factor.

In conclusion, we have proved the following theorem:

Theorem 9.5.1. *We are given an arbitrary budget B and a $d-$dimensional lattice $\mathbb{Z}^d_{N^{1/d}}$ of N nodes. Let τ_s and τ_d represent the expected time to reach a $1-$consensus for, respectively, an optimal static policy and an optimal dynamic policy. Then:*

1. *If $d = 1$, a dynamic policy can perform significantly better than a static policy: $\tau_s = (\tau_d)^2$.*
2. *If $d = 2$, a dynamic policy cannot perform better than a static policy by more than a $(\log N)^2$ factor: $\frac{\tau_s}{\tau_d} \lesssim (\log N)^2$.*
3. *If $d \geq 3$, a dynamic policy cannot perform better than a static policy by more than a $\log N$ factor: $\frac{\tau_s}{\tau_d} \lesssim \log N$.*

We conclude that, in dimension $d \geq 2$, dynamic policies do not perform significantly better than static policies.

s

Chapter 10

Conclusion and Future Prospects

The general goal of this thesis was to study opinion control policies in two different scenarios. In the first part of the thesis, we considered a broad class of deterministic dynamics governing the interactions inside a network, and we designed a policy that a controller can follow in order to spread an opinion inside a network with the smallest possible cost: this lead to the development of the *Descendant Algorithm* and the *Ancestral Algorithm*. In the second part of the thesis, we focused on the classical "Voter Model" (over networks whose underlying graph is the $d-$dimensional integer torus $\mathbb{Z}_n^d$), and we designed policies that minimize the expected time until the network reaches a consensus. In the second part, we considered the case where stubborn nodes were fixed over the lattice, and the case where the controller is allowed to move the stubborn nodes during the experiment. We ended this part of the thesis by showing that, in dimension $d \geq 2$, dynamic policies do not perform significantly better than static policies. However, in dimension $d = 1$, optimal dynamic policies perform much better than optimal static policies.

In the following, we discuss a few remaining open problems:

Part I: A Deterministic Model

In this first part of the thesis, we have considered networks with deterministic dynamics, and have developed the *Descendant Algorithm* and the *Ancestral Algorithm* to spread an opinion inside the network with the smallest possible cost. We have also assumed that the controller has full knowledge of the entire evolution of the network at the start of the

experiment. It would be interesting to generalize both algorithms and apply them to cases where the controller only knows how the network evolves over a time window comprised of the next K steps , or where the network evolves in a stochastic way; for example, we can build a model in which each edge in the network has probability p of appearing and probability $(1-p)$ of disappearing at each time step.

Part 2: Static Versus Dynamic Policies for the Voter Model

In this second part of the thesis, we focused on the Voter Model over networks whose underlying graph is the $d-$dimensional integer torus $\mathbb{Z}_n^d$. We controlled the network by placing B stubborn nodes over the torus. We first studied the case where we place the stubborn nodes at the start of the experiment and let the system evolve spontaneously from there (this is the "static case"), and then studied the case where we are allowed to change the position of our stubborn nodes over the torus during the run of the experiment (this is the "dynamic case").

1. **Static Case:** it would be interesting to show that spreading the budget uniformly over the torus is actually an order optimal policy in dimension $d = 2$ (we have proved that it was order optimal for $d = 1$ and $d \geq 3$). Another open problem consists in getting tighter bounds for the expected time to reach consensus τ^*: in this work we are bounding τ^* up to a factor of $\log(N - B)$.

2. **Dynamic Case:** an open problem consists in determining whether dynamic policies are significantly better than static policies in other network topologies. In particular, we can consider general regular graphs (in this case, we have proved that $\tau^* = \Theta\left(\frac{N}{B}\right)$ for optimal dynamic policies), and try to bound τ^* for optimal static policies.

Bibliography

[1] David Aldous and James Allen Fill. Reversible markov chains and random walks on graphs, 2002. Unfinished monograph, recompiled 2014, available at `http://www.stat.berkeley.edu/~aldous/RWG/book.html`.

[2] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *Proceedings of the 3rd International Conference on Internet and Network Economics*, WINE'07, pages 306–311, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] Peter Clifford and Aidan Sudbury. A model for spatial conflict. *Biometrika*, 60(3):581–588, 1973.

[4] J. T. Cox. Coalescing random walks and voter model consensus times on the torus in $\mathbb{Z}^d$. *Ann. Probab.*, 17(4):1333–1366, 10 1989.

[5] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 57–66, New York, NY, USA, 2001. ACM.

[6] Rick Durrett. *Random Graph Dynamics (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, New York, NY, USA, 2006.

[7] Fabio Fagnani. Consensus dynamics over networks. 2014. Technical paper available at `http://www-sop.inria.fr/members/Giovanni.Neglia/complexnetworks14/14winter_school_complex_networks_consensus%20dynamics_notes.pdf`.

[8] Richard A. Holley and Thomas M. Liggett. Ergodic theorems for weakly interacting infinite systems and the voter model. *Ann. Probab.*, 3(4):643–663, 08 1975.

[9] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[10] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Proceedings of the 32Nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 1127–1138, Berlin, Heidelberg, 2005. Springer-Verlag.

[11] D.A. Levin, Y. Peres, and E.L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Soc.

[12] Thomas Milton Liggett. *Stochastic interacting systems : contact, voter, and exclusion processes.* Grundlehren der mathematischen Wissenschaften. Springer-Verlag, Berlin, New York, 1999.

[13] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 61–70, New York, NY, USA, 2002. ACM.

[14] Ercan Yildiz, Asuman Ozdaglar, Daron Acemoglu, Amin Saberi, and Anna Scaglione. Binary opinion dynamics with stubborn agents. *ACM Trans. Econ. Comput.*, 1(4):19:1–19:30, December 2013.