Computability and Complexity of opinion dynamics

Computabilidad y Complejidad de la dinámica de opiniones

Bachelor's Thesis (Trabajo de Fin de Grado)
Academic year: 2022 - 2023

Author
Martín Gómez Abejón

Directors
Ismael Rodríguez Laguna
Fernando Rubio Diez

Doble Grado en Ingeniería Informática - Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

Contents

Abstract

In this work, we discuss and prove several theoretical results related to computability and complexity properties of an opinion dynamics model we define. We also try to find low-complexity heuristics and algorithms which let us obtain approximate solutions of an **NP**-complete problem based on our model, and run simulations to determine to what extent these heuristics are effective. Although we conclude that most practical problems related to our model are undecidable in the infinite case and **PSPACE**-complete or **NP**-complete in the finite case, the obtained low-complexity algorithms are effective at solving instances whose structure and properties are similar to those of real-world examples.

Keywords

Complexity, approximability, **PSPACE**-completeness, undecidability, heuristics, greedy algorithms, genetic algorithms.

Resumen

En este trabajo, se abordan y demuestran varios resultados teóricos relacionados con las propiedades de computabilidad y complejidad de un modelo de dinámica de opiniones que definimos. También intentamos encontrar heurísticas y algoritmos de baja complejidad que nos permitan obtener soluciones aproximadas de un problema **NP**-completo basado en nuestro modelo, y ejecutamos simulaciones para determinar en qué medida estas heurísticas son efectivas. A pesar de que concluimos que la mayoría de problemas prácticos relacionados con nuestro modelo son indecidibles en el caso infinito y **PSPACE**-completos o **NP**-completos en el caso finito, los algoritmos de baja complejidad obtenidos son efectivos resolviendo instancias cuya estructura y propiedades son parecidas a las de ejemplos del mundo real.

Palabras clave

Complejidad, aproximabilidad, **PSPACE**-completitud, indecibilidad, heurísticas, algoritmos voraces, algoritmos genéticos.

1 Introduction

In this work we use graphs to model opinion dynamics, discuss the complexity of the model we use and estimate to what extent real-world graphs which represent acquaintances can be processed by efficient enough algorithms to obtain relevant information.

Studying opinion dynamics has many applications in modern society, since modifying the opinion of large groups of people can modify their behaviour and have important consequences, such as an increase in sales, an unexpected election result or a behavioural change. Therefore, a lot of resources are spent every year by various entities in order to change people's opinion on a certain subject. Information and techniques for making this task easier are very valuable. This work provides certain insight into influencing opinion dynamics, so it can be directly used, helping to achieve better results.

As part of this work, we have analysed how difficult this task is. We have also proposed, tested and compared various techniques for changing people's opinion using limited resources. These techniques can be applied in real-world cases if enough information is known about the group of people whose opinion needs to be changed. Performance results for all proposed techniques have been empirically tested by using graphs of acquaintances whose design makes them very similar to real-world graphs of acquaintances. This is done to ensure empirical conclusions can be applied, since these conclusions might be different if the used graphs are not similar to real-world instances.

In order to follow what has been done in this work, the reader needs to be familiar with certain relevant theoretical notions and research related to this topic. In this first section, we briefly discuss this necessary background. The references we use are self-contained and appropriate for readers who are not familiarised with any of the topics. We deliberately choose not to discuss concepts which are part of the syllabus of any Bachelor's degree in Computer Science, such as the classes **P** and **NP**. All this omitted theory can be found in [11]. It can also be found in [4], although the contents of [4] are much broader than those of [11].

The theoretical results we obtain are rather discouraging; problems related to infinite opinion graphs are not decidable, and for finite opinion graphs most interesting problems are either **NP**-complete or **PSPACE**-complete. However, by using greedy algorithms and genetic strategies, we are able to obtain good solutions for a problem which is **NP**-complete in general. Therefore, solving problems related to opinion dynamics in real-world contexts is feasible, even if the obtained solutions might not be optimal.

1.1 Theoretical framework

We now discuss all non-standard prerequisites we need for this work. This prerequisites consist of three major blocks. One of them covers certain aspects of advanced complexity and computability theory, while the latter two are related to how opinion dynamics are modelled with graphs, and which is the best approach to obtain realistic graphs which are topologically similar to real-world networks of acquaintances.

1.1.1 Space complexity

We briefly explain the notions we need to know about space complexity in order to study how complex opinion dynamics are. The concepts we discuss have been extracted from [4].

Space complexity is not studied as part of a standard Bachelor's degree. Just like with time complexity, we can define classes of problems which can be solved if a certain amount of space is given. Namely, we have the following definitions.

Definition 1 (Space-bounded computation)**.** *A language L, or its corresponding problem P, is said to belong to $\boldsymbol{SPACE}(s(n))$ if there is a constant C and a deterministic Turing machine M which decides L, thus solving P, such that not more than $C \cdot s(n)$ different cells are ever visited by M's head when the length of the input equals n.*

We similarly define the class $\boldsymbol{NSPACE}$; $L \in \boldsymbol{NSPACE}(s(n))$ if there is a constant C and a non-deterministic Turing machine M which decides L, such that not more than $C \cdot s(n)$ different cells are ever visited by M's head when the length of the input equals n, for any set of non-deterministic choices taken by M.

Time and space complexity are related to each other, since a Turing machine which runs for t cycles cannot visit more than t cells, and the number of different possible global configurations of a Turing machine M which does not visit more than c cells is bounded by $K_1 \cdot K_2^c$, for certain K_1, K_2 which depend on M.

Theorem 1 (Time and space complexity)**.** *The following set inclusions hold.*

$$\boldsymbol{DTIME}(s(n)) \subseteq \boldsymbol{SPACE}(s(n)) \subseteq \boldsymbol{NSPACE}(s(n)) \subseteq \boldsymbol{DTIME}\left(2^{O(s(n))}\right)$$

A rigorous proof of this result can be found in [4]. The most relevant space complexity classes in this work are

$$\mathbf{PSPACE} \stackrel{\text{def}}{=} \bigcup_{c \in \mathbb{N}} \mathbf{SPACE}(n^c) \text{ and } \mathbf{NPSPACE} \stackrel{\text{def}}{=} \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}(n^c)$$

which intuitively consists of all problems which can be solved using a polynomial amount of space. Just like for **NP**, there exists a notion of *completeness*, which we now define. Just like in [4], we use the symbol $\leq_p$ for polynomial-time Karp reducibility.

Definition 2 (**PSPACE**-completeness)**.** *A language L' is said to be $\boldsymbol{PSPACE}$-hard if and only if $\forall L \in \boldsymbol{PSPACE}$, $L \leq_p L'$. If L' is $\boldsymbol{PSPACE}$-hard and $L' \in \boldsymbol{PSPACE}$, it is said to be $\boldsymbol{PSPACE}$-complete.*

It can be proved that the language

$$L_{\text{accsp}} \stackrel{\text{def}}{=} \{(M, w, 1^n) \mid M \text{ accepts } w \text{ in space } n\} \tag{1}$$

is **PSPACE**-complete, where M represents a *variable* semi-infinite tape Turing machine. Of course, it is indispensable that there exists a well-defined encoding which can be used to represent every possible M in a string. It is standard to assume that this encoding makes it possible to use polynomial-time algorithms for encoding and decoding for use in a universal Turing machine; for certain encodings for which infinitely many words represent

Turing machines whose numbers of states are not bounded by a multiple of a power of the length of the strings, the existence of these polynomial-time algorithms cannot be guaranteed. We use the same construction and assumptions as in [4]; polynomial-time algorithms exist, since Turing machines are encoded by listing the table of their transition functions.

We make some additional remarks which are relevant for this work. It is almost immediate to prove L_{accsp} is **PSPACE**-complete if one is familiar enough with **PSPACE**. Nevertheless, it is not a satisfying **PSPACE**-complete language, since it does not give any intuition about what a **PSPACE**-hard problem looks like. A better example of a **PSPACE**-complete problem is the language of all quantified Boolean formulas which are true. A quantified Boolean formula has the structure

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, x_2, \dots, x_n), \tag{2}$$

where $\forall i \in \mathbb{N}_n, Q_i \in \{\forall, \exists\}$ and $\varphi(x_1, x_2, \dots, x_n)$ is a propositional logic formula on variables $x_1, x_2, \dots, x_n$. A proof of the **PSPACE**-completeness of this problem can be found in [4].

This **PSPACE**-complete problem provides more insight; since the quantifiers can appear in any order and alternate as frequently as needed, it is possible to use formulas like (2) to encode (possibly non-deterministic) two-player games. Each $\forall$ represents a choice taken by the opponent, or a random event which takes place in the game, while each $\exists$ represents a voluntary choice taken by the player. One of these formulas is true if and only if there exists a strategy which leads to victory[1].

We finally state one last property without proving it; **PSPACE** = **NPSPACE**. It is a direct consequence of Theorem 1 from [16], which states that if $s(n) \geq \log(n)$, then $\mathbf{NSPACE}(s(n)) \subseteq \mathbf{SPACE}(s(n)^2)$.

1.1.2 A standard model used for opinion dynamics

Many different models have been used to try to model human interactions and how a person's opinion is determined depending on their entourage and their own initial opinion. The vast majority of these models are continuous stochastic processes in which an individual opinion is updated using a stochastic transition function which takes the initial opinion of a person and its interactions with others into account.

A very used model for this purpose is the voter model from [13]. This model is a continuous time Markov process in which nodes in a lattice ($\mathbb{Z}^d$) interact with each other locally. Nodes can agree (1) or disagree (0), and the transition rates for each node are given by a function $c(x, \eta)$, where x is a position in $\mathbb{Z}^d$ and $\eta \in \{0, 1\}^{\mathbb{Z}^d}$ is the current state of the whole lattice. The higher c is for a specific node x at a given point in time, the more likely it is that the opinion of x changes. c is assumed to have the following properties.

(1) Let η^1 and η^0 be the states in which every node agrees or disagrees, respectively.

[1] If (2) is thought of as a game, victory is encoded in $\varphi(x_1, x_2, \dots, x_n)$, and the game which is played goes as follows: from left to right, a player determines the value of all $\exists$ quantifiers, while the opponent determines the value of all $\forall$ quantifiers. The ultimate goal for the player is to make $\varphi(x_1, x_2, \dots, x_n)$ true, while the opponent tries to make $\varphi(x_1, x_2, \dots, x_n)$ false.

Then, $\forall\ x \in \mathbb{Z}^d, c(x, \eta^1) = c(x, \eta^0) = 0$. This means nodes keep having the same opinion if no other node disagrees with them.

(2) Opinions are symmetric; transition rates are the same if the opinions of all nodes are *flipped*. Quantitatively, $c(x, \eta^\alpha) = c(x, \eta^\beta)$ if $\forall\ y \in \mathbb{Z}^d, \eta^\alpha(y) + \eta^\beta(y) = 1$.

(3) Transition rates for a specific node x are higher when nodes *uniformly* disagree more with x. The word uniformly is very important in this property; since it is not known to what extent each individual node influences x, nothing can be said about the global change of the transition rate of x if some other nodes start to disagree with x, while others start to agree. This property can be expressed using two equivalent equations.

$$\text{If } \forall y \in \mathbb{Z}^d, \eta^\alpha(y) \leq \eta^\beta(y) \text{ and } \eta^\alpha(x) = \eta^\beta(x) = 0, \text{ then } c(x, \eta^\alpha) \leq c(x, \eta^\beta).$$
$$\text{If } \forall y \in \mathbb{Z}^d, \eta^\alpha(y) \leq \eta^\beta(y) \text{ and } \eta^\alpha(x) = \eta^\beta(x) = 1, \text{ then } c(x, \eta^\alpha) \geq c(x, \eta^\beta).$$

(4) The transition rates are invariant under translations in $\mathbb{Z}^d$; if the opinions of all nodes are translated using a vector v, the transition rates can be obtained by translating the original transition rates using v as well.

Despite the name of this model, it was originally not conceived for modelling opinion dynamics. However, it has been shown that modified versions of this model can accurately describe opinion dynamics in real-world contexts (see [8]). However, the original model from [13] has an issue; the graph topology used (a lattice) and the properties of transition rates, especially being invariant under translation, make the model very different from real-world social networks, as we will see. For instance, in [8], the graph of interactions which is considered is very different from the original graph proposed in [13].

In this work, we have analysed the complexity of a modified voter model; instead of being a continuous process, it is discrete. Also, it is deterministic, unlike almost any other voter model or variant thereof which has previously been considered by other authors. Certain deterministic models have been considered by certain authors, such as in [7], but they are very different from the model we consider in this work. We analyse the complexity and expressiveness of this deterministic model, which has never been done before in this context, to the best of our knowledge. Due to the law of large numbers, it can be assumed that a deterministic model is capable of simulating global properties of opinion dynamics.

1.1.3 Real-world social graphs

The voter model discussed in [13] and its variants have been used in many different network topologies. Examples of this are [8], as well as [17], [14] and [5]. Most authors do not consider networks which are similar to real-life social interaction graphs. In [8], this is done successfully, but not all features of real-life graphs, such as average shortest path length and degree distribution (see below), are considered. In [17], the considered graphs are very different from real-life, as they are not sparse and their clustering coefficients (see below) are different from real-life coefficients like those obtained in [3]. In this work, we consider all relevant properties of social networks when modelling them. We now discuss the most relevant properties.

Although the dynamics of many voter models and other games have been considered for lattices (apart from the voter model, there are other examples in other contexts, such as [6] and [15], in which not only $\mathbb{Z}^n$ is considered), it has been shown by many authors (see, for example, [3] and [9]) that lattices are very different from real-life social networks. In [3], many different examples are studied and three important properties are deduced.

(1) The average shortest path length is very small, and grows logarithmically compared to the number of nodes in the network. Random graphs (whose edges are determined randomly and independently) have this property. Graphs which have this property are known as small-world networks.

(2) The clustering coefficient is relatively high compared to random graphs. This coefficient is obtained for each node by considering its neighbours and calculating the proportion of edges between them that exist. When the value of this coefficient is 1, the neighbours form a complete graph, and when it is low, it is highly unlikely that neighbours know each other. The average value of these coefficients is the clustering coefficient of the graph.

(3) The distribution of node degrees follows a power law. This was studied in [3] and related papers. In order to obtain such a network, edges need to be added by giving more probability to nodes which are already very connected. A model which takes this approach is the Barabási-Albert model discussed in [3].

It has been shown in [18], as well as in [3], that combining different types of edges corresponding to lattices and more global edges, chosen randomly or depending on the degree of the nodes, yields networks which satisfy all of the properties discussed above. There exist other models or properties which characterise real-life social networks, which have been discussed in papers like [9] and [12], but these other approaches are very similar to the one taken in [3].

1.2 Objectives and methodology

In this work, we identify and prove the most important complexity properties of a deterministic model we define. This model is based on the voter model from [13], but it is modified by making it deterministic and discrete, and modifying its topology by considering lattices of blocks of nodes, instead of lattices of nodes. We consider each block to be a community which is influenced by nearby communities. It is true that the average shortest path length of this model remains high, as in the original voter model. Nevertheless, it is useful for proving worst-case complexity properties, and we will later modify it to make it more realistic when analysing its behaviour in real-world conditions.

Despite the worst-case complexity properties we obtain, we try to simulate real-world examples of this model and use heuristic algorithms in order to evaluate to what extent these can give satisfying answers to problems whose exact solutions cannot be obtained easily. For this part of the work, we consider graphs which are as similar to real-world social graphs as possible using the properties discussed above, and consider different heuristic algorithms to solve problems related to these graphs. Additional information about the methodology used for these simulations and algorithms can be found in Section 4.

1.3 Notation used in this work

We use certain standard and non-standard notation in this work. Notation is usually introduced in each part of this work, and the reader is reminded certain aspects when deemed appropriate. Most of the times, this is the best approach. Nevertheless, there are certain non-standard conventions we use very frequently, which is why we explain them here.

We use $\mathbb{N}$ for the set of all natural numbers. In this work, $0 \notin \mathbb{N}$ for convenience. When necessary, we use $\mathbb{Z}_{\geq 0} = \mathbb{N} \cup \{0\}$ when it is more convenient. We also use the notation $\mathbb{N}_s = \{1, 2, \ldots, s\}$ for the set of all natural numbers starting from 1 until (and including) s, only when $s \geq 1$.

When working with graphs, we use the standard notation $G = (V, E)$, where V is the set of vertices and E is the set of edges. We use the words vertex and node interchangeably. Abuses of notation are common in this context; for example, it might be stated that a node belongs to a graph, although it is more correct to state that a node belongs to the set of nodes of the graph.

2 Definition of the general problem

2.1 General definition of opinion graphs

We will now define the most general graph and problem we are going to cover. In this text, we will mostly study more constrained versions of this problem.

The opinion graphs we intend to model consist of nodes and edges. Each node represents a person, who can only be for (1) or against (0) something. Although this approach might seem simplistic, it is the one taken in [13] and all other related texts we have mentioned, and avoids making graphs extremely complicated, while ensuring they are complex enough for modelling most real-life situations without loss of generality.

Our model makes two non-standard assumptions compared to the models discussed in [13]; while the voter model assumes time is continuous and the recalculation of the opinions of nodes is modelled using continuous stochastic processes as time passes, we assume opinions are recalculated periodically, deterministically and simultaneously by all nodes in our model. These opinion updates are given by a function which is applied locally[2]. This kind of opinion updates reduce the complexity of the model. However, it is still quite general in our view, since randomness arising from individual stochastic and continuous transitions has no significant global effect when the number of nodes and the time scale are large enough, provided that the interactions we consider model opinion dynamics accurately[3]. It can be assumed that the continuous and stochastic transitions which take place from time t to time $t+1$ can be summarised and approximated by using a single discrete and deterministic transition.

The opinion graphs we consider are directed graphs $G = (V, E)$ whose edges have invariant weights in $[0, 1]$. In order to prove certain complexity results, we require all weights to be rational. In the graph, vertices represent people and edges represent influence links between two specific people. The number of vertices these models have is always countable, so without loss of generality they can be indexed by $\mathbb{N}$ or a smaller subset and referred to as v_α, where α is the index. Given a vertex v_i, its influential nodes are those from which an edge goes to v_i. This set of nodes will be denoted by $\text{infl}(v_i)$. Multiple edges which share both source v_i and destination v_j cannot coexist. Nevertheless, it is possible to have an edge whose source and destination are the same node[4].

Let us define more useful notation. When an instance of the problem has two vertices v_i, v_j and an edge whose source node is v_i and whose destination node is v_j, we refer to this edge as e_{ij}. This notation is well defined, since no two edges can have the same source node and the same destination node. Given an edge e_{ij}, we will use the notation $|e_{ij}|$ to denote its weight.

[2]This means only the states of nearby nodes determine the new state of a node when a recalculation takes place.

[3]When trying to model human interactions, we need to take into account that the effect of a regular individual on the general opinion of society or part of it is negligible. Certain individuals' opinions which do not behave as expected (this phenomenon is modelled by using randomness in other models) compensate each other or create a global effect which can be modelled by changing certain global parameters.

[4]These reflexive edges are used for modelling how *adamant* a node is; the higher the weight these reflexive edges have, the more conservative their corresponding nodes will be when faced to alternative opinions their influential nodes have.

Due to the way we have defined our model it makes sense to require that for each node v_j,

$$\sum_{v_i \in \text{infl}(v_j)} |e_{ij}| = 1.$$

This way, weights are homogenised.

An instance $S = (G, S_0)$ of an opinion graph consists of an opinion graph $G = (V, E)$ and a function $S_0 : V \mapsto \{0, 1\}$. The function S_0 assigns a label (1 or 0) to each vertex. S_0 can be thought of as the initial setup of the labels at the beginning of the problem ($t = 0$). In general, given a certain moment $t \in \mathbb{Z}_{\geq 0}$ in time, each vertex $v_i \in V$ has a label $S_t(v_i) = v_i(t)$ which stores a binary variable. We will use the notation $v_i(t)$ only when it is not ambiguous (a single instance S is being considered).

The (deterministic) transition rates function we use in all opinion graphs is

$$v_j(t+1) = \begin{cases} 0, & \text{if } \sum_{v_i \in \text{infl}(v_j)} v_i(t)|e_{ij}| < \frac{1}{2}, \\ v_j(t), & \text{if } \sum_{v_i \in \text{infl}(v_j)} v_i(t)|e_{ij}| = \frac{1}{2}, \\ 1, & \text{if } \sum_{v_i \in \text{infl}(v_j)} v_i(t)|e_{ij}| > \frac{1}{2}, \end{cases}$$

which inductively defines S_i for all $i \in \mathbb{Z}_{\geq 0}$ given G and S_0. This transition function is very similar to the one defined for linear voter models in [13][5], but it is deterministic; a node's opinion *flips* if its corresponding transition rate in the linear voter model is greater than $1/2$.

2.2 Graphs based on lattices

We now define a specific type of opinion graphs, on which we focus in this work. Their regular structure uses some ideas from [13] and other papers which discuss the voter model, as well as from [6] and [15]. Nodes are arranged in an integer lattice. Nevertheless, our lattices are different from the ones used in all other texts mentioned so far, since other authors assume vertices are arranged in a lattice, while we assume each element of the lattice is a group of nodes which influence and are influenced by contiguous groups. For this purpose, we will consider n-dimensional lattices formed by Cartesian products of the sets $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_s$ for any $s \in \mathbb{N}, s \geq 3$. Given one of these lattices L, we now define the concept of an L-shaped graph. This concept allows us to define blocks which repeat in space and form a (possibly infinite) graph whose structure is the one of L. Figures 1 and 2 are examples of this.

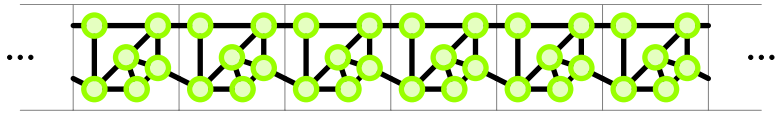

Figure 1: A $\mathbb{Z}$-shaped graph. Reflexive edges and the weights and directions of all edges are not shown.

[5]The equations we consider in (2.1) are very similar to the ones of the linear voter model; equivalent expressions are obtained for the left-hand sides in (2.1) if $p(i, j)$ is replaced by $|e_{ij}|$ and $\eta(i)$ is replaced by $v_i(t)$ in the equations of transition rates.

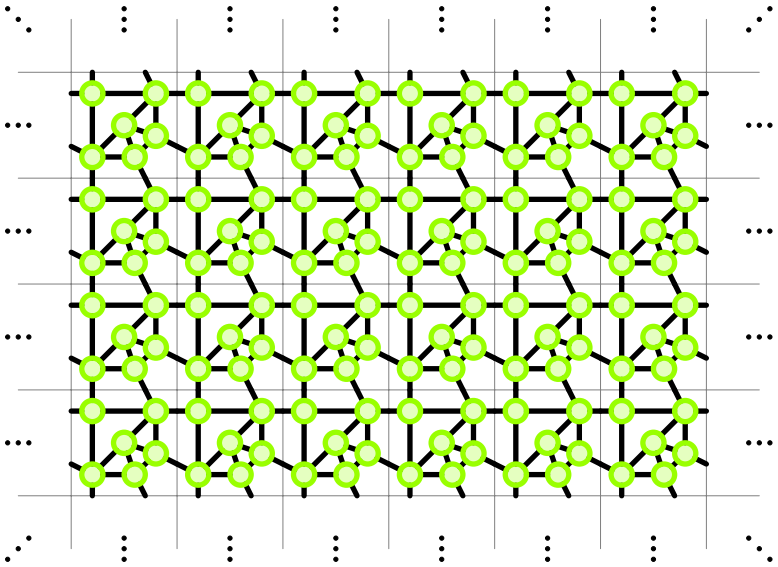

Figure 2: A $\mathbb{Z} \times \mathbb{Z}$-shaped graph. In this depiction, two nodes are connected by an edge if and only if one of them is influenced by the other one. Reflexive edges and the weights and directions of all edges are not shown.

Intuitively, L-shaped graphs consist of repeating blocks in a lattice which have the same structure and are connected to neighbours with which they share a border; these connections have the same structure everywhere in the lattice. However, blocks at the border of the lattice cannot have exactly the same structure, since they do not have as many blocks around them to which they can be connected. In order to solve that problem, we only require that all blocks which have exactly the same blocks around them (and in the same positions) have the same structure. This means two blocks are necessarily equal if and only if their relative position (first block, block in the middle, last block) is the same for each coordinate. Figure 3 depicts a general case, in which not all blocks are equal.

We introduce some notation in order to deal with this issue: the set of relative positions of a block in a given lattice

$$L = E_1 \times E_2 \times \cdots \times E_d$$

is given by

$$\mathrm{Rel}(L) \stackrel{\text{def}}{=} F_1 \times F_2 \times \cdots \times F_d$$

where

$$F_i \stackrel{\text{def}}{=} \begin{cases} \{\texttt{Mid}\}, & \text{if } E_i = \mathbb{Z}, \\ \{\texttt{Fst}, \texttt{Mid}\}, & \text{if } E_i = \mathbb{N}, \\ \{\texttt{Fst}, \texttt{Mid}, \texttt{Lst}\}, & \text{if } E_i = \mathbb{N}_s. \end{cases}$$

Given an element of L, expressed as $(e_1, e_2, \ldots, e_d)$, its relative position is obtained by replacing each coordinate e_i by its corresponding relative position $\mathrm{Rel}(e_i)$ given by

$$\mathrm{Rel}(e_i) \stackrel{\text{def}}{=} \begin{cases} \texttt{Fst}, & \text{if } e_i = 1 \text{ and } E_i \neq \mathbb{Z}, \\ \texttt{Lst}, & \text{if } E_i = \mathbb{N}_{e_i}, \\ \texttt{Mid}, & \text{in all other cases.} \end{cases}$$

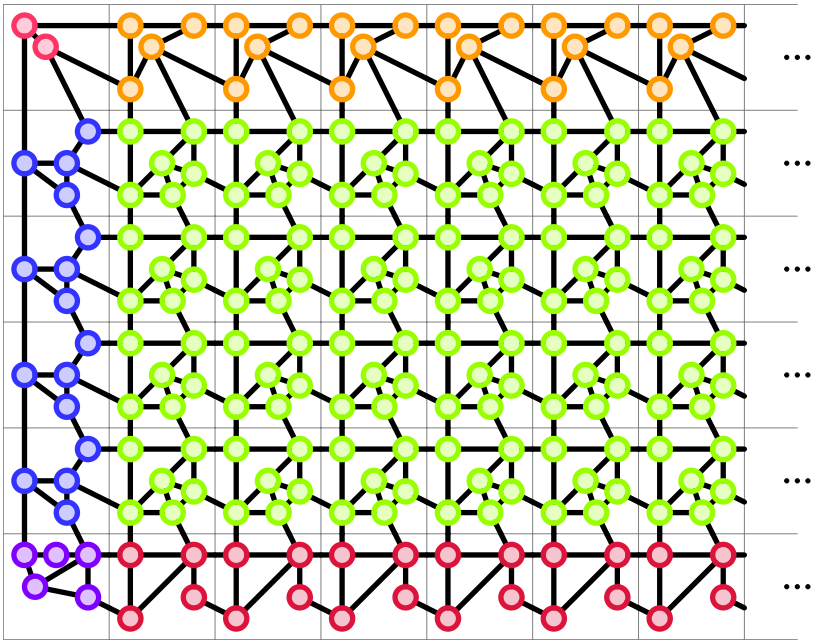

Figure 3: An $\mathbb{N} \times \mathbb{N}_6$-shaped graph. Not all blocks are the same, since $\mathbb{N}$ has a first element and $\mathbb{N}_6$ has both first and last elements. This implies there exist six different relative positions, for each of which a description of a block is required. Reflexive edges and the weights and directions of all edges are not shown.

We now define L-shaped graphs formally. We start by defining what an interface is. Interfaces are used to represent connections to contiguous blocks when describing an individual block of an L-shaped graph. In Figures 1, 2 and 3, these cross-block connections are represented by edges which cross a grid line. Formally, an interface is a function $I : \mathbb{N}_k \to \{\uparrow, \downarrow\}$. The objective of an interface is representing the way a block is connected to its successor or its predecessor in a certain dimension. The number of elements an interface has represents the number of connections. If $I(r) = \uparrow$, the edge corresponding to connection r leaves the predecessor block and enters the successor block. Likewise, if $I(r) = \downarrow$, the corresponding edge leaves the successor block and enters the predecessor block.

Each block in an L-shaped graph is connected to exactly one interface for each contiguous block. Moreover, in order to guarantee the graph is regular enough, two interfaces which belong to the same dimension are always the same for each block and for all blocks of the graph, including not general ones (whose relative position is not $\{\texttt{Mid}, \texttt{Mid}, \ldots, \texttt{Mid}\}$). Figure 4 shows how a block is connected to four interfaces.

When a single block (which is part of an L-shaped graph) is defined, we cannot directly consider nodes which influence or are influenced by it in the L-shaped graph, since they depend on where the block is placed. The definition of a block intuitively consists of nodes, edges which interconnect them and edges which leave or enter nodes via interfaces. In order not to make definitions too complicated, the approach taken in this work consists in adding *phantom nodes* for every edge which leaves or enters the block through an interface, just like in Figure 4. This way, we do not need to define a special type of edge. Instead, we mark these phantom nodes as part of the interface by means of functions which are part of the definition of the blocks of the L-shaped graph. When the definitions of the

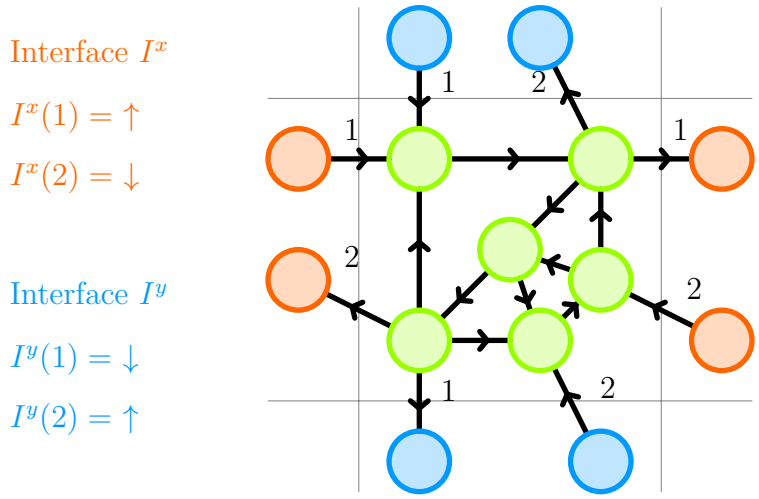

Figure 4: A general block connected to four interfaces (for each of the two dimensions and as both predecessor and successor for each dimension). *Phantom nodes* (orange and blue) are part of the description of an individual block. These nodes are removed when making this opinion graph part of an L-shaped graph and their edges are connected to nearby blocks of the L-shaped graph. Reflexive edges and the weights of all edges are not shown. Interfaces belonging to the same dimension are always equal (in this case, two instances of I^x connect the block to the blocks on the left and on the right, and two instances of I^y connect the block to the blocks above and below). However, interfaces corresponding to different dimensions need not be isomorphic and can be completely different from each other.

different blocks (depending on the relative position in the L-shaped graph) are given, the L-shaped graph is built by creating copies of them and connecting each block to its neighbours. This is done by removing all phantom nodes (which are only useful when defining a general block) and connecting contiguous blocks directly, while preserving the original properties of the edges connected to phantom nodes in the definitions of abstract blocks.

This intuition about interfaces is formalised in the following definitions. We remark that Definitions 3 and 4 only consider individual blocks of an L-shaped graph, which are connected to interfaces so they can become part of the larger L-shaped graph.

Definition 3 (Opinion graph connected to an interface)**.** *An opinion graph $G = (V, E)$ is said to be connected as predecessor to an interface $I : \mathbb{N}_k \to \{\uparrow, \downarrow\}$ via the function $I_p : \mathbb{N}_k \to V$ if:*

(1) I_p is injective.

(2) If $I(r) = \uparrow$, then there exists a unique edge $e \in E$ whose destination is $I_p(r)$. Its weight equals 1, its source does not belong to the image set of I_p and there exist no edges whose source is $I_p(r)$.

(3) If $I(r) = \downarrow$, then there exists a unique non-reflexive edge $e \in E$ whose source is $I_p(r)$. Its destination is not a node in the image set of I_p and there exist no non-reflexive

edges whose destination is $I_p(r)$.

A graph is said to be connected as successor to an interface I if the same conditions for connection as predecessor, interchanging the roles of $\uparrow$ and $\downarrow$, apply for a given injective function $I_s : \mathbb{N}_k \to V$.

The nodes which belong to the image of I_p are called phantom nodes.

When it is not known if the block is connected as a successor or as a predecessor, we use the notation $I_{p,s}$ for the injective function which marks phantom nodes. In Definition 3, the function $I_{p,s}$ uniquely assigns a phantom node to each connection. Conditions (2) and (3) merely state that the edges which connect phantom nodes to other nodes should behave as intended by the interface I. If $I(r) = \uparrow$, the phantom node $I_{p,s}(r)$ behaves as a tube leaving the predecessor and entering the successor. On the other hand, if $I(r) = \downarrow$, the phantom node $I_{p,s}(r)$ behaves as a tube leaving the successor and entering the predecessor.

Definition 4 makes it possible to connect several interfaces to a single block. In order to have a consistent definition which can be used to interconnect blocks in an L-shaped graph, phantom nodes must be different for each connection to an interface and they cannot interfere with each other. The latter condition is quite important in order to guarantee all connections are local and the L-shaped graph is well defined.

Definition 4 (Opinion graph connected to multiple interfaces)**.** *An opinion graph $G = (V, E)$ is said to be connected to multiple interfaces $\{I^j\}_{j\in J}$ via the functions $\{I^j_{p,s}\}_{j\in J}$ if it is connected to each interface I^j via the function $I^j_{p,s}$ and the following conditions are satisfied.*

(1) The images of the injective functions $\{I^j_{p,s}\}_{j\in J}$ are pairwise disjoint.

(2) There is no edge whose source v_s and destination v_d are in the union of the image sets of $\{I^j_{p,s}\}_{j\in J}$.

The core G^c of a graph G which is connected to multiple interfaces $\{I^j\}_{j\in J}$ via the functions $\{I^j_{p,s}\}_{j\in J}$ consists of the graph G, from which all phantom nodes of all interfaces and all edges which leave or enter phantom nodes have been removed.

In general, the core of a given block is not an opinion graph, since the sum of the weights of the influential edges of each node is not necessarily 1 after removing some of them. However, the sum of the weights becomes 1 again when the core is connected to other cores in an L-shaped graph.

We now have enough tools to formally define what an L-shaped graph is, given an arbitrary L. This definition formalises the concepts discussed above.

Definition 5 (L-shaped graph)**.** *Given $L = E_1 \times E_2 \times \cdots \times E_d$, an L-shaped graph is completely determined by the following definitions:*

(1) Interfaces $\{I^j\}_{j\in\mathbb{N}_d}$ for each of the d dimensions of L.

(2) Finite opinion graphs G_R for all $R \in Rel(L)$. It is necessary that these graphs be connected to the following interfaces.

(a) *For all elements $R \in Rel(L)$ and all $j \in \mathbb{N}_d$, if the j-th element of R is not* `Lst`*, then G_R must be connected to I^j as a predecessor.*

(b) *For all elements $R \in Rel(L)$ and all $j \in \mathbb{N}_d$, if the j-th element of R is not* `Fst`*, then G_R must be connected to I^j as a successor.*

The L-shaped graph G corresponding to these definitions consists of the disjoint union of the graphs $G_l \cong G^c_{Rel(l)}$ for each $l \in L$, which are interconnected by adding the following edges.

For all $j \in \mathbb{N}_d, e = (e_1, e_2, \ldots, e_{j-1}, e_j, e_{j+1}, \ldots, e_d) \in L$, given $I^j : \mathbb{N}_k \to \{\uparrow, \downarrow\}$, if $e^{j+} \stackrel{def}{=} (e_1, e_2, \ldots, e_{j-1}, e_j + 1, e_{j+1}, \ldots, e_d) \in L$, then an edge is added to the L-shaped graph as follows for each $r \in \mathbb{N}_k$.

(1) If $I^j(r) = \uparrow$, its source is the node originally connected to $I^j_p(r)$ in G^c_e, its destination is the node originally connected to $I^j_s(r)$ in $G^c_{e^{j+}}$ and its weight is the one of the only non-reflexive edge whose source is $I^j_s(r)$ in $G_{Rel(e^{j+})}$.

(2) If $I^j(r) = \downarrow$, its source is the node originally connected to $I^j_s(r)$ in $G^c_{e^{j+}}$, its destination is the node originally connected to $I^j_p(r)$ in G^c_e and its weight is the one of the only non-reflexive edge whose source is $I^j_p(r)$ in $G_{Rel(e)}$.

Each of the graphs G_l is called a block.

When constructing G by following this definition, it is guaranteed that the sum of the weights of all edges of $inf(v_i)$ equals 1, for all $v \in G$. However, there might be edges with common source and destination if a node in G_R influences or is influenced by several phantom nodes of an interface function $I_{p,s}$. If this happens, all these edges are removed and replaced by a single edge whose source and destination are the ones of the removed edges, and whose weight is the sum of the weights of the removed edges.

Blocks are interconnected in Definition 5 by connecting the two sides of interfaces to which contiguous blocks are connected. Figure 5 shows how defining two interfaces and six graphs which are connected to those interfaces following a $\{\texttt{Fst}, \texttt{Mid}\} \times \{\texttt{Fst}, \texttt{Mid}, \texttt{Lst}\}$ arrangement characterise the $\mathbb{N} \times \mathbb{N}_6$-shaped graph depicted in Figure 3 via Definition 5. The reader can also have a look at Figure 6, which provides intuition in how blocks are connected to each other and how edge weights are assigned.

2.3 The deterministic problem is increasing

Intuitively, the transition function we have defined for the general instance of an opinion graph is increasing when restricted to a specific node, and its properties are similar to those of increasing functions in general. We will now define these concepts more precisely and show how they limit certain expressiveness properties of the deterministic version of the problem. Given an opinion graph $G = (V, E)$,

$$L_G \stackrel{\text{def}}{=} \{S : V \mapsto \{0, 1\}\}$$

denotes the set of all possible labellings of G.

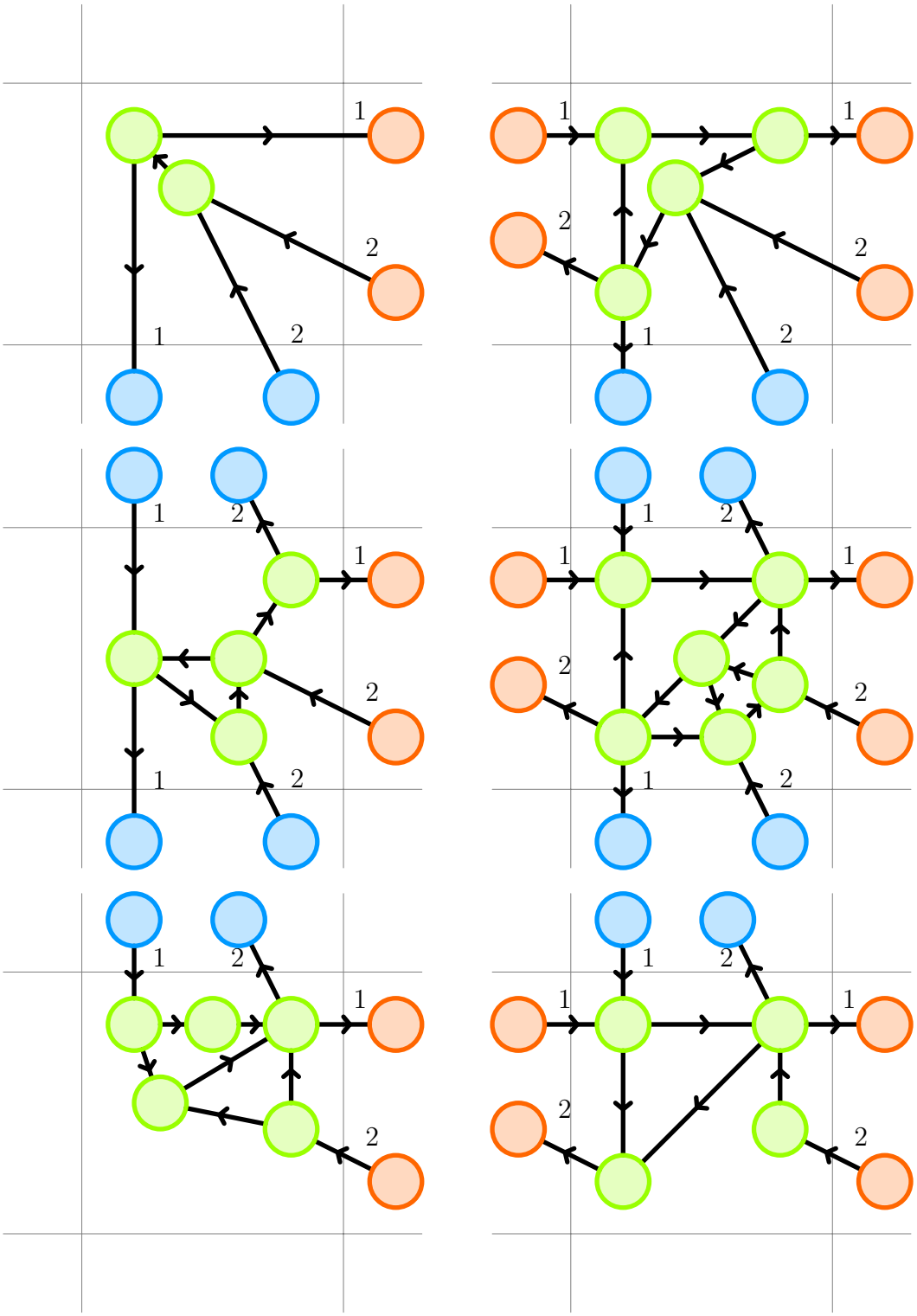

Figure 5: Definitions for the six necessary types of graphs used to define the graph from Figure 3. Reflexive edges and the weights of all edges are not shown.

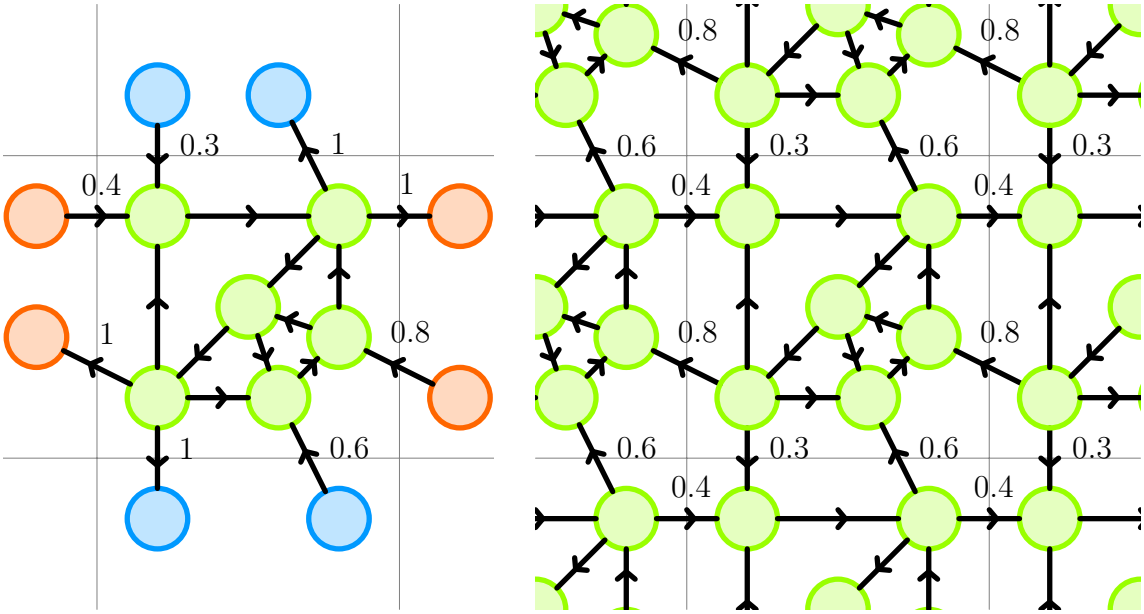

Figure 6: Definition of $G_{(\texttt{Mid},\texttt{Mid})}$ on the left and part of the resulting $\mathbb{Z} \times \mathbb{Z}$-shaped graph on the right. Reflexive edges and some of the weights are not shown. The weights of connections to an interface which leave the block always equal 1 in G_R by definition, and they are replaced by the weights of incoming connections when constructing an L-shaped graph, as it can be seen on the right.

Definition 6 (partial order on L_G)**.** *Given an opinion graph $G = (V, E)$ we define a partial order on the functions L_G by defining*

$$S^\alpha \leq S^\beta \overset{def}{\iff} \forall v \in V, S^\alpha(v) \leq S^\beta(v).$$

This way we partially order the possible states of the graph G; the definition of the partial order implies that $S^\alpha \leq S^\beta$ when and only when all node labels in S^α are lower than or equal to the corresponding labels in S^β.

We now give another definition which naturally extends the concept of increasing functions to the partial order in L_G.

Definition 7 (increasing functions $L_G \mapsto L_G$)**.** *Given an opinion graph $G = (V, E)$, a function $f : L_G \mapsto L_G$ is increasing if and only if*

$$\forall S^\alpha, S^\beta \in L_G, S^\alpha \leq S^\beta \implies f(S^\alpha) \leq f(S^\beta).$$

The following theorem has a lot of consequences and significantly reduces the expressiveness of opinion graphs. Even though we will later prove opinion graphs are essentially Turing-complete, this theorem implies we need to be careful when defining the instances; in practice, we will need to construct pairwise non-comparable instances in order to guarantee that they are different enough.

Theorem 2. *Given an opinion graph $G = (V, E)$, the transition function f which maps a certain state of labels in G (S_t) to the next state (S_{t+1}) is increasing.*

Proof. Let $S_t^\alpha, S_t^\beta \in L_G$ and assume $S_t^\alpha \leq S_t^\beta$. From the definition of $\leq$ we need to prove that given any vertex $v_j \in V$, $S_{t+1}^\alpha(v_j) \leq S_{t+1}^\beta(v_j)$. The inequality

$$\sum_{v_i \in \text{infl}(v_j)} S_t^\alpha(v_i)|e_{ij}| \leq \sum_{v_i \in \text{infl}(v_j)} S_t^\beta(v_i)|e_{ij}|$$

is satisfied because $S_t^\alpha(v_i) \leq S_t^\beta(v_i)\ \forall v_i$, the weights of the edges are non-negative and inequalities are preserved by sums. The function

$$S_{t+1}^\alpha(v_j) = \begin{cases} 0, & \text{if } \sum_{v_i \in \text{infl}(v_j)} S_t^\alpha(v_i)|e_{ij}| < \frac{1}{2}, \\ S_t^\alpha(v_j), & \text{if } \sum_{v_i \in \text{infl}(v_j)} S_t^\alpha(v_i)|e_{ij}| = \frac{1}{2}, \\ 1, & \text{if } \sum_{v_i \in \text{infl}(v_j)} S_t^\alpha(v_i)|e_{ij}| > \frac{1}{2}, \end{cases}$$

seen as a function whose arguments are $\sum_{v_i \in \text{infl}(v_j)} S_t^\alpha(v_i)|e_{ij}|$ and $S_t^\alpha(v_j)$, is increasing[6]. That implies each individual output of the function S_{t+1}^α in function of S_t^α is increasing, which in turn implies the global transition function is increasing.

Two corollaries can be easily deduced from the previous theorem. Their proofs are trivial and not particularly relevant for the rest of the text, so we list them for completeness without proving them.

Corollary 1 (inequalities relating instances preserved)**.** *Given a graph G and two instances $S = (G, S_0), T = (G, T_0)$, if $S_0 \leq T_0$ then*

$$\forall t \in \mathbb{Z}_{\geq 0}, S_t \leq T_t.$$

Corollary 2 (increasing sequences of labels converge)**.** *Given an instance of the problem $S = (G, S_0)$, if $S_1 \geq S_0$ then $S_{n_1} \geq S_{n_2}$ whenever $n_1 \geq n_2$. This implies that for each node v_i, the sequence of its labels $\{v_i(t)\}_t$ is eventually constant, although the convergence is not necessarily uniform[7] for all nodes.*

These corollaries have numerous important consequences; for instance, Corollary 1 implies `NOT`-gates cannot be simulated by choosing a specific graph, choosing a certain node v (*input node*) whose label at time 0 equals the input of the `NOT`-gate, simulating the graph for a fixed number of iterations and asserting the value of a certain node w (*output node*) equals the output of the `NOT`-gate; if the final w is 0 when the initial v is 1, changing the instance by decreasing the initial value of v cannot increase the value of w. Besides, Corollary 2 reduces the complexity of simulating opinion graphs for which $S_1^\alpha \geq S_0^\alpha$, since only vertices whose label is 0 need to be reevaluated, and only when the value of a neighbour increases to 1.

[6]When both arguments increase, the value of the function increases, although maybe not strictly. Note that this implies $S_{t+1}^\alpha(v_j)$ increases or stays the same if the labels of S_t^α are all increased or left unmodified.

[7]This means the sequence of values of any given node always converges, since these sequences are constant or flip once from 0 to 1. However, the values do not converge at the same time, and in infinite graphs there might not be a time at which all values have converged.

3 Complexity properties of opinion graphs

In this section we prove opinion graphs based on lattices are Turing-complete and discuss some of the properties of the complexity of simulating such graphs. The results we obtain are mostly based on one key fact: it is possible to simulate Turing machines with opinion graphs by using a $\mathbb{Z}$-shaped graph in which each block represents a cell and encodes the sets Q and Γ and the transition function δ which describe the Turing machine. We first describe how opinion graphs are functionally complete, then explain how a Turing machine can be simulated. Last but not least, we deduce several important results about the problem's complexity.

3.1 Opinion graphs are functionally complete

The results discussed in previous sections might seem discouraging, since they show `NOT`-gates cannot be simulated as such in opinion graphs. This makes it necessary to take a different approach when tackling this problem. In order to solve the issue, we are going to prove functional completeness by constructing graphs which duplicate information by keeping the opposite label of each node in another node. This duplication of information does not substantially increase the complexity of the graphs and solves the problem of the increasing transition function; if we consider two states $S^{\alpha}_{t_0}$ and $S^{\beta}_{t_0}$ of an opinion graph $G = (V, E)$ which satisfy

$$\begin{aligned} S^{\alpha}_{t_0}(v_i) &= 0, & S^{\beta}_{t_0}(v_i) &= 1, \\ S^{\alpha}_{t_0}(v_j) &= 1, & S^{\beta}_{t_0}(v_j) &= 0, \end{aligned}$$

they are not comparable with respect to the partial order introduced in Definition 6, so none of the problems we have in Subsection 2.3 apply in this case. We now prove that opinion graphs are functionally complete, just like described in Theorem 3.

Theorem 3 (Functional completeness of opinion graphs)**.** *Given variables $x_1, \ldots, x_t$ and a propositional logic formula $P(x_1, \ldots, x_t)$, there exists a non-negative integer $\Delta \in \mathbb{Z}_{\geq 0}$, known as the propagation time, and a finite opinion graph $G = (V, E)$ in which certain pairwise disjoint sets of nodes $\left\{V^+_{x_i}\right\}^t_{i=1}, \left\{V^-_{x_i}\right\}^t_{i=1}, C$ and nodes v^+_r, v^-_r (possibly in one of the former sets) satisfy the following properties:*

(1) Nodes in the sets $\left\{V^+_{x_i}\right\}^t_{i=1}, \left\{V^-_{x_i}\right\}^t_{i=1}, C$ do not have any non-reflexive incoming edges, which implies that for any choice of initial values S^{α}_t,

$$S^{\alpha}_{t+1}(v) = S^{\alpha}_t(v) \text{ if } v \in \left\{V^+_{x_i}\right\}^t_{i=1} \cup \left\{V^-_{x_i}\right\}^t_{i=1} \cup C.$$

(2) There exists a labelling S^C for nodes in C, such that if

$$\begin{aligned} &\forall i \in \mathbb{N}_t, v \in V^+_{x_i}, S^{\alpha}_{t_0}(v) = x_i, \\ &\forall i \in \mathbb{N}_t, v \in V^-_{x_i}, S^{\alpha}_{t_0}(v) = \neg x_i, \\ &\forall v \in C, S^{\alpha}_{t_0}(v) = S^C(v), \end{aligned}$$

then it is true that

$$S^{\alpha}_{t_0+\Delta}(v_r^+) = P(x_1,\ldots,x_t) \text{ and } S^{\alpha}_{t_0+\Delta}(v_r^-) = \neg P(x_1,\ldots,x_t), \tag{3}$$

even if G is integrated inside a larger graph, and its transitions are modified by influencing nodes in $\left\{V_{x_i}^+\right\}_{i=1}^t \cup \left\{V_{x_i}^-\right\}_{i=1}^t$ arbitrarily, by means of external nodes.

Being able to arbitrarily add edges which influence nodes in $\left\{V_{x_i}^+\right\}_{i=1}^t \cup \left\{V_{x_i}^-\right\}_{i=1}^t$, while still satisfying (3), lets us use Theorem 3 to include graphs which calculate arbitrary propositional logic formulas inside larger graphs. This will be very useful when proving general complexity properties of opinion graphs.

We use structural induction for proving opinion graphs are functionally complete.

The possible base cases are the formulas $\top$, $\bot$ and x_i for any i. These base cases are easily seen to be satisfied. For $\top$, a possible G consists of two nodes $v_1 = v_r^+$ and $v_2 = v_r^-$ and no non-reflexive edges. This graph satisfies the property of functional completeness explained above for $\Delta = 0$, provided that $v_1, v_2 \in C$ and

$$S^C(v_1) = 1, S^C(v_2) = 0.$$

For $\bot$, the same graph, output nodes and $\Delta = 0$ work, although the labelling S^C must then satisfy

$$S^C(v_1) = 0, S^C(v_2) = 1.$$

The third base case ($P(x_1,\ldots,x_t) = x_i$) can be obtained by considering the same graph, the same $\Delta = 0$, the same v_r^+, v_r^- and the sets

$$V_{x_i}^+ = \{v_1\}, V_{x_i}^- = \{v_2\}.$$

In this base case, all other node sets considered in Theorem 3 (C and all other sets $V_{x_j}^+, V_{x_j}^- \mid j \in \mathbb{N}_t, j \neq i$) are empty.

We now discuss inductive steps. It suffices to prove that given two formulas P_1 and P_2 for which it is possible to construct opinion graphs as in Theorem 3, analogous graphs can be constructed for each of the formulae

$$\neg P_1, P_1 \vee P_2, P_1 \wedge P_2,$$

since it is well known that the set $\{\neg, \vee, \wedge\}$ is functionally complete in logic.

The construction for $\neg P$ given a construction for the formula P is almost immediate. Given G^P, Δ^P, the sets $\left\{V_{x_i}^{+,P}\right\}_{i=1}^t, \left\{V_{x_i}^{-,P}\right\}_{i=1}^t, C^P$, the nodes $v_r^{+,P}, v_r^{-,P}$ and $S^{C,P}$, all of which satisfy the conditions in Theorem 3 for P, a construction for $\neg P$ is obtained by interchanging the roles of $v_r^{+,P}$ (which becomes $v_r^{-,\neg P}$) and $v_r^{-,P}$ (which becomes $v_r^{+,\neg P}$), and not altering any other element of the construction.

We now give a proof for $P_1 \vee P_2$ given constructions for P_1 and P_2. Given both constructions, we first consider the disjoint union of both of them, which is defined as follows.

$$\begin{aligned} G &\stackrel{\text{def}}{=} (V^{P_1} \sqcup V^{P_2}, E^{P_1} \sqcup E^{P_2}), \\ V_{x_i}^+ &\stackrel{\text{def}}{=} V_{x_i}^{+,P_1} \sqcup V_{x_i}^{+,P_2}, \\ V_{x_i}^- &\stackrel{\text{def}}{=} V_{x_i}^{-,P_1} \sqcup V_{x_i}^{-,P_2}, \\ C &\stackrel{\text{def}}{=} C^{P_1} \sqcup C^{P_2}. \end{aligned}$$

By using the symbol $\sqcup$, we mean the disjoint union of the two sets involved; by using it here, we essentially create a graph which consists of two separate copies of the constructions for P_1 and P_2. These two copies have no common vertices or edges, even if they have the same original definition (which might happen if P_1 and P_2 have a common subformula). S^C is defined in C and extends both S^{C,P_1} and S^{C,P_2} (it agrees with both S^{C,P_1} on C^{P_1} and S^{C,P_2} on C^{P_2}).

We now describe how to complete this disjoint union in order to obtain a construction for $P_1 \vee P_2$. Intuitively, we need to add an `OR`-gate made of new nodes, whose input nodes are the original output nodes of G, which are

$$v_r^{+,P_1}, v_r^{-,P_1}, v_r^{+,P_2}, v_r^{-,P_2}.$$

The problem with such an approach is that we need to take time constraints into account. Let us suppose that the values of nodes in the sets $\left\{V_{x_i}^+\right\}_{i=1}^t, \left\{V_{x_i}^-\right\}_{i=1}^t$ correspond to a certain assignment of the variables $x_1, \ldots, x_t$ at time t_0. Then, the values of v_r^{+,P_1} and v_r^{-,P_1} encode $P_1(x_1, \ldots, x_t)$ at time $t_0 + \Delta^{P_1}$, and those of v_r^{+,P_2} and v_r^{-,P_2} encode $P_2(x_1, \ldots, x_t)$ at time $t_0 + \Delta^{P_2}$. Since Δ^{P_1} and Δ^{P_2} are not necessarily equal, the inputs given to the `OR`-gate do not necessarily enter the gate at the same time, which implies a correct result can never be guaranteed.

This issue can be solved by adding chains of nodes which generate a delay in the fastest output, so both outputs arrive at the same time. If $\Delta^{P_2} < \Delta^{P_1}$, we add the nodes

$$v_{\Delta^{P_2}+1}^{ret+}, v_{\Delta^{P_2}+2}^{ret+}, \ldots, v_{\Delta^{P_1}-1}^{ret+}, v_{\Delta^{P_1}}^{ret+}, \qquad v_{\Delta^{P_2}+1}^{ret-}, v_{\Delta^{P_2}+2}^{ret-}, \ldots, v_{\Delta^{P_1}-1}^{ret-}, v_{\Delta^{P_1}}^{ret-},$$

and the following edges, whose weight is 1.

(1) An edge whose source is v_r^{+,P_2} and whose destination is $v_{\Delta^{P_2}+1}^{ret+}$. The presence of this edge implies the value of node $v_{\Delta^{P_2}+1}^{ret+}$ at time $t_0 + \Delta^{P_2} + 1$ is the value of node v_r^{+,P_2} at time $t_0 + \Delta^{P_2}$, which is $P_2(x_1, \ldots, x_t)$.

(2) For all $i \in \left\{\Delta^{P_2} + 2, \Delta^{P_2} + 3, \ldots, \Delta^{P_1} - 1, \Delta^{P_1}\right\}$, an edge whose source is v_{i-1}^{ret+} and whose destination is v_i^{ret+}. Analogously, the presence of these edges implies the value of node v_i^{ret+} at time $t_0 + i$ is $P_2(x_1, \ldots, x_t)$.

(3) Analogous edges (like those in points (1) and (2)) which connect the nodes

$$v_r^{-,P_2} \to v_{\Delta^{P_2}+1}^{ret-} \to v_{\Delta^{P_2}+2}^{ret-} \to \cdots \to v_{\Delta^{P_1}-1}^{ret-} \to v_{\Delta^{P_1}}^{ret-}.$$

By adding these edges, the value of the node $v_{\Delta^{P_1}}^{ret+}$ at time $t_0 + \Delta^{P_1}$ is $P_2(x_1, \ldots, x_t)$, and that of the node $v_{\Delta^{P_1}}^{ret-}$ at time $t_0 + \Delta^{P_1}$ is $\neg P_2(x_1, \ldots, x_t)$. These two nodes have a similar role to the original nodes v_r^{+,P_2} and v_r^{-,P_2}, but their delay is Δ^{P_1} instead of Δ^{P_2}, so they are coordinated with the nodes v_r^{+,P_1} and v_r^{-,P_1}. The new edges only influence new nodes, so all properties assumed in the induction hypotheses are preserved after modifying the graph.

An analogue of this modification can be carried out if $\Delta^{P_1} < \Delta^{P_2}$, so no matter how long delays are, we can always create an artificial delay in the fastest graph so all output nodes are coordinated.

We now describe how to add an OR-gate to the (possibly modified) disjoint union if $\Delta^{P_1} = \Delta^{P_2} \stackrel{\text{def}}{=} \Delta$, which we can now assume. We add four more nodes to the graph, which are

$$v_r^+, v_r^-, v_{\texttt{OR}}^+, v_{\texttt{OR}}^-.$$

The nodes v_r^+ and v_r^- are the output nodes of $\stackrel{\text{def}}{=} P_1 \vee P_2$, which will be calculated in $\Delta+1$ iterations. These nodes process the information provided by the output nodes of P_1 and P_2 at time $t_0 + \Delta$, so their values at time $t_0 + \Delta + 1$ represent $P_1 \vee P_2$. The nodes $v_{\texttt{OR}}^+$ and $v_{\texttt{OR}}^-$ are used to influence v_r^+ and v_r^- so the logical OR is calculated correctly. They need to have specific values, so we add them to C and we extend the definition of S^C by defining

$$S^C(v_{\texttt{OR}}^+) = 1, S^C(v_{\texttt{OR}}^-) = 0.$$

We add the following edges, which only affect the newly added nodes.

(1) Reflexive edges whose weight is 1 for the nodes $v_{\texttt{OR}}^+$, $v_{\texttt{OR}}^-$.

(2) Three edges whose common weight is 1/3, whose common destination is v_r^+ and whose sources are $v_{\texttt{OR}}^+, v_r^{+,P_1}, v_r^{+,P_2}$. It is trivial that if the constraints given by S^C are satisfied, then the value of v_r^+ at time $t+1$ is the logical OR of the values of v_r^{+,P_1}, v_r^{+,P_2} at time t.

(3) Three edges whose common weight is 1/3, whose common destination is v_r^- and whose sources are $v_{\texttt{OR}}^-, v_r^{-,P_1}, v_r^{-,P_2}$. Just like in point (2), it is trivial that if the constraints given by S^C are satisfied, then the value of v_r^- at time $t+1$ is the logical AND[8] of the values of v_r^{-,P_1}, v_r^{-,P_2} at time t.

After modifying the disjoint union by following the steps described above, we are left with a construction for $P_1 \vee P_2$ which satisfies the properties of Theorem 3. Figure 7 represents the construction graphically. A construction for $P_1 \wedge P_2$ can be obtained similarly by defining

$$S^C(v_{\texttt{AND}}^+) = 0, S^C(v_{\texttt{AND}}^-) = 1,$$

and using them instead of $v_{\texttt{OR}}^+$ and $v_{\texttt{OR}}^-$, or by using De Morgan's laws. The proof for functional completeness of opinion graphs is thus complete.

3.2 Building a Turing machine with opinion graphs

Functional completeness of opinion graphs and the concept of L-shaped graphs let us define an $\mathbb{N}$-shaped graph which can simulate a semi-infinite tape Turing machine. This construction will be extremely useful to prove many complexity properties of the problem we are considering. We basically use each block of the $\mathbb{N}$-shaped graph to encode a cell of the Turing machine we are considering, and construct the graph such that simulating its evolution is equivalent to simulating the evolution of the Turing machine.

Before we begin to discuss the central topic of this section, we make an important remark related to terminology. The word *state* in the context of Turing machines can be ambiguous if not used properly, since it can have two different meanings. On the one hand, the

[8]This makes sense for the role of v_r^-, since De Morgan's laws state that, in particular, $\neg P_1 \wedge \neg P_2 = \neg(P_1 \vee P_2)$.

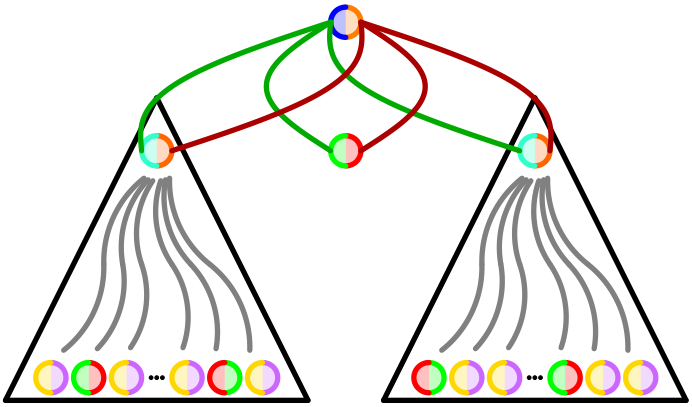

Figure 7: Graphical representation of the inductive step carried out when adding an OR-gate. The direction and weight of newly added edges are not shown. Pairs of nodes with opposite values are shown as two semicircles. Input nodes and nodes in C (bottom of the triangles) generate intermediate outputs (top of the triangles), whose OR-value (node above) is computed by means of two nodes (one circle), whose values always equal 1 and 0, and the six edges mentioned in the text.

machine's state can refer to the content of the state register, which always is one of the finitely many possible states. On the other hand, the machine's state can mean the global state, which consists of the m-configuration, the position of the head and the contents of the tape. During the rest of this work, we use the terms m-configuration, internal state and state for the former concept, and we use the terms global state, state of progress and state of the system for the latter.

Given a semi-infinite tape Turing machine whose alphabet is Γ, whose set of states is Q and whose transition function is $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}$ (if the machine is in state q, σ is read and $\delta(q, \sigma) = (q', \sigma', D)$, then the machine will replace σ by σ' on the tape, change its internal state to q' and move left, stay or move right, if D equals L, S or R respectively, except for the first cell, for which the effect of L is staying in the same cell), we assume without loss of generality that the values of the sets Γ and Q can be encoded by using a finite number of binary variables. In general, we use the notation $s_1, s_2, \ldots, s_\mu$ for binary variables which encode the states, and $a_1, a_2, \ldots, a_\nu$ for binary variables which encode the alphabet. The two encodings need not be related to each other.

The definition of a semi-infinite tape Turing machine implies that for every cell e of the machine, given the following information,

- the definition of the Turing machine,
- whether or not the head is currently pointing at e or a cell next to it, and if it is the case, the current m-configuration,
- the symbols written on cell e and the cell(s) next to it,

it is possible to calculate the following information (which can be calculated by a Boolean circuit if the states and the alphabet have been encoded by means of binary variables),

- whether or not the head will be pointing at e at the next step, and if it is the case, the m-configuration at the next step,
- the symbol written on cell e at the next step.

This allows us to emulate a semi-infinite tape Turing machine by using an $\mathbb{N}$-shaped opinion graph which *locally* calculates all variables related to the state of the system.

In Theorem 4, we merely state that given a Turing machine M and an initial global state, it is possible to simulate how M works as time passes by using an $\mathbb{N}$-shaped graph G, although time passes slower; in order for G to simulate one step of computation carried out by M, we need to simulate G for Δ_M steps (where $\Delta_M \in \mathbb{N}$ only depends on M) and look at the final result.

G consists of one copy of $G^c_{\texttt{Fst}}$ and infinitely many copies of $G^c_{\texttt{Mid}}$ which are interconnected by interfaces. Inside each block G_e, there exist certain special nodes which encode the following data.

- Whether or not the current location of the head of the Turing machine is the e-th cell $(v_e^{p,+}, v_e^{p,-})$.
- The value of the tape at the e-th cell $(\left\{v_{e,i}^{a,+}\right\}_{i=1}^{\nu}, \left\{v_{e,i}^{a,-}\right\}_{i=1}^{\nu})$.
- The current internal state $(\left\{v_{e,i}^{s,+}\right\}_{i=1}^{\mu}, \left\{v_{e,i}^{s,-}\right\}_{i=1}^{\mu})$. As we will see below, these nodes are only guaranteed to store the actual internal state if the head of the machine is pointing to cell e.

There also exists a set C_e whose elements are vertices of G_e and must satisfy certain constraints which guarantee that subgraphs of G_e which compute propositional logic formulas work the way the are supposed to. Just like we saw when proving opinion graphs are functionally complete, two different nodes (whose values are opposite) are used for each encoded bit. These sets of nodes are the same for blocks of the same type; in $G^c_{\texttt{Mid}}$, there exist disjoint sets of nodes

$$\left\{v_{\texttt{Mid}}^{p,+}, v_{\texttt{Mid}}^{p,-}\right\}, \left\{v_{\texttt{Mid},i}^{s,+}\right\}_{i=1}^{\mu}, \left\{v_{\texttt{Mid},i}^{s,-}\right\}_{i=1}^{\mu}, \left\{v_{\texttt{Mid},i}^{a,+}\right\}_{i=1}^{\nu}, \left\{v_{\texttt{Mid},i}^{a,-}\right\}_{i=1}^{\nu}, C_{\texttt{Mid}}, \tag{4}$$

whose copies inside every block $G_e \cong G^c_{\texttt{Mid}}$ are

$$\left\{v_e^{p,+}, v_e^{p,-}\right\}, \left\{v_{e,i}^{s,+}\right\}_{i=1}^{\mu}, \left\{v_{e,i}^{s,-}\right\}_{i=1}^{\mu}, \left\{v_{e,i}^{a,+}\right\}_{i=1}^{\nu}, \left\{v_{e,i}^{a,-}\right\}_{i=1}^{\nu}, C_e, \tag{5}$$

respectively. This also applies to $G_1 \cong G^c_{\texttt{Fst}}$. In the rest of this document, and in Definition 8 and Theorem 4 in particular, the notation mentioned in (5) is used for denoting the copies of the sets and nodes mentioned in (4) (or the analogous sets of $G^c_{\texttt{Fst}}$) inside each block G_e. This abuse of notation is also used for labellings $S^C_{\texttt{Fst}}, S^C_{\texttt{Mid}}$ of $C_{\texttt{Fst}}, C_{\texttt{Mid}}$, respectively, which are defined analogously inside each block G_e and are referred to as S^C_e[9].

[9]These labellings assign certain fixed labels to nodes in C_e and play the same role as the labelling S^C in Theorem 3; they guarantee that logic gates used to model the internal logic of a Turing machine M behave as expected.

We first define what it means for a labelling of an $\mathbb{N}$-shaped graph (where the node sets discussed above are defined) to encode the state of progress of a Turing machine M. This definition is similar to others given in similar contexts; the labelling represents a certain global state if and only if the labels of the sets defined and discussed above fully encode the state of the system.

Definition 8 (Representation of a state of progress by a labelling)**.** *Given a semi-infinite tape Turing machine M, an $\mathbb{N}$-shaped opinion graph G, disjoint subsets of nodes of G^c_{Fst}*

$$\left\{v^{p,+}_{Fst}, v^{p,-}_{Fst}\right\}, \left\{v^{s,+}_{Fst,i}\right\}^{\mu}_{i=1}, \left\{v^{s,-}_{Fst,i}\right\}^{\mu}_{i=1}, \left\{v^{a,+}_{Fst,i}\right\}^{\nu}_{i=1}, \left\{v^{a,-}_{Fst,i}\right\}^{\nu}_{i=1}, C_{Fst},$$

analogous disjoint subsets of nodes of G^c_{Mid}

$$\left\{v^{p,+}_{Mid}, v^{p,-}_{Mid}\right\}, \left\{v^{s,+}_{Mid,i}\right\}^{\mu}_{i=1}, \left\{v^{s,-}_{Mid,i}\right\}^{\mu}_{i=1}, \left\{v^{a,+}_{Mid,i}\right\}^{\nu}_{i=1}, \left\{v^{a,-}_{Mid,i}\right\}^{\nu}_{i=1}, C_{Mid},$$

labellings S^C_{Fst}, S^C_{Mid} of C_{Fst}, C_{Mid}, respectively, and a state of progress[10] $\alpha = \left(q, \{d_n\}_{n\in\mathbb{N}}, x\right)$ of M, a labelling S^G of the nodes of G is said to represent α if and only if the following conditions are satisfied inside each block G_e.

(1) The values $\left\{S(v^{a,+}_{e,i})\right\}^{\nu}_{i=1}, \left\{S(v^{a,-}_{e,i})\right\}^{\nu}_{i=1}$ encode the symbol d_e written on the e-th cell of M.

(2) $v^{p,+}_e = 1, v^{p,-}_e = 0$ if $x = e$[11], and $v^{p,+}_e = 0, v^{p,-}_e = 1$ otherwise.

(3) If $x = e$, the values $\left\{S(v^{s,+}_{e,i})\right\}^{\mu}_{i=1}, \left\{S(v^{s,-}_{e,i})\right\}^{\mu}_{i=1}$ encode the state q.

(4) For all $v \in C_e$, $S(v) = S^C(v)$.

Condition (4) of Definition 8 is not directly related to encoding α, but guarantees the next state of progress is calculated correctly, so we include it for convenience. Having discussed how a semi-infinite tape Turing machine can be encoded in an $\mathbb{N}$-shaped graph and what the roles of sets in (5) are, we now prove opinion graphs are Turing-complete.

Theorem 4 (Semi-infinite tape Turing machines can be simulated by $\mathbb{N}$-shaped opinion graphs)**.** *Given a semi-infinite tape Turing machine M, there exist a $\Delta_M \in \mathbb{N}$, an $\mathbb{N}$-shaped opinion graph G, disjoint subsets of nodes of G^c_{Fst}*

$$\left\{v^{p,+}_{Fst}, v^{p,-}_{Fst}\right\}, \left\{v^{s,+}_{Fst,i}\right\}^{\mu}_{i=1}, \left\{v^{s,-}_{Fst,i}\right\}^{\mu}_{i=1}, \left\{v^{a,+}_{Fst,i}\right\}^{\nu}_{i=1}, \left\{v^{a,-}_{Fst,i}\right\}^{\nu}_{i=1}, C_{Fst}, \tag{6}$$

analogous disjoint subsets of nodes of G^c_{Mid}

$$\left\{v^{p,+}_{Mid}, v^{p,-}_{Mid}\right\}, \left\{v^{s,+}_{Mid,i}\right\}^{\mu}_{i=1}, \left\{v^{s,-}_{Mid,i}\right\}^{\mu}_{i=1}, \left\{v^{a,+}_{Mid,i}\right\}^{\nu}_{i=1}, \left\{v^{a,-}_{Mid,i}\right\}^{\nu}_{i=1}, C_{Mid} \tag{7}$$

and labellings S^C_{Fst}, S^C_{Mid} of C_{Fst}, C_{Mid} respectively, such that given an initial state of progress $\alpha = \left(q_0, \{d_n\}_{n\in\mathbb{N}}, x_0\right)$ and a labelling S_0 of G, if S_0 represents α, then $S_{\Delta_M \cdot t}$ represents the state of progress obtained starting from α after simulating M for t steps[12].

[10] The notation we use for the state of progress α is as follows: q_0 is the m-configuration, $\{d_n\}_{n\in\mathbb{N}}$ are the symbols written on the tape and x_0 is the position of the head.

[11] Here, x is the cell at which the head of M can be found in the state of progress α. Therefore, $x = e$ if and only if the head of M is at cell e in α.

[12] Just like we described in Section 2, $S_{\Delta_M \cdot t}$ is the labelling of G if the initial one was S_0 and $\Delta_M \cdot t$ time steps have passed.

A proof of this theorem can be obtained by applying Theorem 3 and defining blocks of G appropriately, as we will now see.

Given a semi-infinite tape Turing machine M, a moment of time t and a cell e, it is possible, as we explained above, to know if the head will be on e at time $t+1$ (and what the internal state of M will be if so), as well as the symbol written on cell e at time $t+1$, if that same information is known at time t for cell e and its neighbouring cell(s). To explore this dependence, we define δ_q, δ_σ and δ_D to be the projections of the transition function of M. This means

$$\delta(q,\sigma) = (q',\sigma',D) \iff \delta_q(q,\sigma) = q', \delta_\sigma(q,\sigma) = \sigma', \delta_D(q,\sigma_r) = D.$$

If p_t^e is a binary variable which is true if and only if the position of the head of M is e at time t, σ_t^e is the symbol written on cell e at time t and q_t^e stores the internal state of M at time t whenever $p_t^e = \top$, then the following equations characterise[13] the global state of M at time $t+1$ given the global state at time t.

$$\begin{aligned} p_{t+1}^e = &\left(p_t^e \wedge \left(\delta_D\left(q_t^e, \sigma_t^e\right) = \mathtt{S}\right)\right) \\ &\vee \left(p_t^{e+1} \wedge \left(\delta_D\left(q_t^{e+1}, \sigma_t^{e+1}\right) = \mathtt{L}\right)\right) \\ &\vee \left(p_t^{e-1} \wedge \left(\delta_D\left(q_t^{e-1}, \sigma_t^{e-1}\right) = \mathtt{R}\right)\right), \end{aligned} \tag{8}$$

$$\sigma_{t+1}^e = \begin{cases} \delta_\sigma\left(q_t^e, \sigma_t^e\right), & \text{if } p_t^e = \top, \\ \sigma_{t+1}^e, & \text{otherwise,} \end{cases} \tag{9}$$

$$q_{t+1}^e = \begin{cases} \delta_q\left(q_t^e, \sigma_t^e\right), & \text{if } p_t^e \wedge \left(\delta_D\left(q_t^e, \sigma_t^e\right) = \mathtt{S}\right), \\ \delta_q\left(q_t^{e+1}, \sigma_t^{e+1}\right), & \text{if } p_t^{e+1} \wedge \left(\delta_D\left(q_t^{e+1}, \sigma_t^{e+1}\right) = \mathtt{L}\right), \\ \delta_q\left(q_t^{e-1}, \sigma_t^{e-1}\right), & \text{if } p_t^{e-1} \wedge \left(\delta_D\left(q_t^{e-1}, \sigma_t^{e-1}\right) = \mathtt{R}\right), \\ \text{undefined}, & \text{otherwise.} \end{cases} \tag{10}$$

If the internal states and the alphabet symbols are binarily encoded, the equations (8), (9) and (10) make it possible to obtain propositional logic formulas whose variables are the bits of

$$\{p_t^x, \sigma_t^x, q_t^x \mid x \in \{e-1, e, e+1\}\} \tag{11}$$

and whose evaluation yields the bits of p_{t+1}^e, σ_{t+1}^e and q_{t+1}^e. These formulas are different for the first block, but have the same structure for the rest of the blocks. We define some notation for these formulas. Given the variables in (11), where a tape symbol σ_t^x is encoded by the binary variables $a_{1,t}^x, a_{2,t}^x, \ldots, a_{\nu,t}^x$ and an m-configuration q_t^x is encoded by the binary variables $s_{1,t}^x, s_{2,t}^x, \ldots, s_{\mu,t}^x$, we use the notation $\mathcal{F}_{\mathtt{Mid}}^p$ to denote the general formula for p_{t+1}^x whose variables are

$$\left\{p_t^x, a_{1,t}^x, a_{2,t}^x, \ldots, a_{\nu,t}^x, s_{1,t}^x, s_{2,t}^x, \ldots, s_{\mu,t}^x \mid x \in \left\{e-1, e, e+1\right\}\right\}. \tag{12}$$

Similarly, we use the notation $\mathcal{F}_{\mathtt{Mid},i}^a$ for the general formula which calculates $a_{i,t+1}^x$ given the binary variables in (12) and $\mathcal{F}_{\mathtt{Mid},j}^s$ for the general formula which calculates $s_{j,t+1}^x$

[13] If the cell we consider is $e = 1$, the equations change, since there is no cell on the left. We avoid excessive repetition by not discussing this special case explicitly. It is analogous to the general one (it is simply obtained by replacing p_t^{e-1} by $\bot$ and $(\delta_D\left(q_t^e, \sigma_t^e\right) = \mathtt{S})$ by $(\delta_D\left(q_t^e, \sigma_t^e\right) \in \{\mathtt{L}, \mathtt{S}\})$). This particular case is covered by the proof of Theorem 4 without the need for more complex reasoning.

given the binary variables in (12). We also define analogous notation for the formulas corresponding to the first block by replacing `Mid` by `Fst`.

For each of these formulas $\mathcal{F}^p_R, \mathcal{F}^a_{R,i}, \mathcal{F}^s_{R,j}$, where $R \in \{\texttt{Fst}, \texttt{Mid}\}$, there exist opinion graphs which compute them and are constructed by using Theorem 3, for which we use the notation $\mathcal{G}^p_R, \mathcal{G}^a_{R,i}, \mathcal{G}^s_{R,j}$. We use the notation $\mathbb{F}_R$ for the set of all such formulas used to calculate variables at the relative position R, which is given by

$$\left\{\mathcal{F}^p_R\right\} \cup \left\{\mathcal{F}^a_{R,i} \mid i \in \mathbb{N}_\nu\right\} \cup \left\{\mathcal{F}^s_{R,j} \mid j \in \mathbb{N}_\mu\right\}.$$

We also define $\mathbb{F} \stackrel{\text{def}}{=} \mathbb{F}_{\texttt{Fst}} \cup \mathbb{F}_{\texttt{Mid}}$. We analogously define $\mathbb{G}_{\texttt{Fst}}, \mathbb{G}_{\texttt{Mid}}, \mathbb{G}$ for the corresponding sets of graphs generated by Theorem 3 using formulas from the sets $\mathbb{F}_{\texttt{Fst}}, \mathbb{F}_{\texttt{Mid}}, \mathbb{F}$ respectively. From now on, we assume the propagation times of all graphs in $\mathbb{G}$ are all the same and equal to $\Delta \in \mathbb{Z}_{\geq 0}$. This can be assumed, since the set $\mathbb{G}$ is finite (in fact, its number of elements equals $2(1 + \mu + \nu)$), so the propagation time of all graphs in $\mathbb{G}$ is bounded by a certain $\Delta \in \mathbb{Z}_{\geq 0}$. If some graphs have a lower propagation time, it can be increased to equal Δ by adding a *double chain* of nodes and connecting it to the output, as in the proof of Theorem 3.

We now describe an $\mathbb{N}$-shaped opinion graph which satisfies the properties stated in Theorem 4 by describing the structures inside each block and how they are interconnected to each other. Before defining them, we make two remarks. Firstly, the corresponding Δ_M (related to Theorem 4 and its properties) of this graph equals $\Delta + 3$, where Δ is the propagation delay of all graphs in $\mathbb{G}$. Secondly, we have not defined how to handle the undefined case of (10) when constructing formulas $\mathcal{F}^s_{R,j} \mid j \in \mathbb{N}_\mu$ and their corresponding opinion graphs. This case can be treated arbitrarily (for example, the output of these formulas must always be 0 for the undefined case) or left unspecified, since these bits will not be relevant for any other computation (in the undefined case, $p^c_{t+1} = \perp$, and the bits of the variable q^c_{t+1} do not determine the outputs for $t + 2$ or any other future time if $p^c_{t+1} = \perp$). Any formulas $\mathcal{F}^s_{R,j} \mid j \in \mathbb{N}_\mu$ which give the right outputs when they are defined will suffice for our purpose.

We also remind the reader that the set $\mathbb{F}$ depends on M. We could have used notation like $\mathbb{F}^M$ in order to remind ourselves that this construction corresponds to a particular and arbitrary Turing machine. However, we have decided not to in order not to make the notation unnecessarily tedious.

We now describe each block of the opinion graph G corresponding to M as the disjoint union of several structures, one of which is the set of special nodes defined in Theorem 4, which encode the presence of the head, the internal state of M if the head is present and the symbol written on each cell. From now on, we will refer to these nodes as v-nodes. For each block G_e, these nodes are

$$v^{p,+}_e, v^{p,-}_e, \left[v^{s,+}_{e,j}, v^{s,-}_{e,j}\right]^\mu_{j=1}, \left[v^{a,+}_{e,i}, v^{a,-}_{e,i}\right]^\nu_{i=1}. \tag{13}$$

The second structure present inside each block G_e, where $\text{Rel}(e) = R$, is the disjoint union of all graphs of $\mathbb{G}_R$. In terms of the graphs G_R from Definition 5, disjoint copies of all graphs in $\mathbb{G}_{\texttt{Fst}}, \mathbb{G}_{\texttt{Mid}}$ are added to $G_{\texttt{Fst}}, G_{\texttt{Mid}}$, respectively. Since the same graphs are present in blocks of the same type, these structures are compatible with Definition 5.

In order to make sure logic gates work properly in G, we need to define C_B and the labelling S^C_B from Theorem 4, so they agree with the definitions of the sets C and the corresponding

labellings S^C, from Theorem 3, on all graphs in $\mathbb{G}_B$. This way, all constraints of all $\mathcal{G} \in \mathbb{G}$ inside G are satisfied, so it is possible to use Theorem 3 for each copy of these graphs and thus guarantee transition functions are calculated correctly.

A third structure inside each block G_e consists of *pseudo-input nodes* given by

$$w_x^{p,+}, w_x^{p,-}, w_{x,j}^{s,+}, w_{x,j}^{s,-}, w_{x,i}^{a,+}, w_{x,i}^{a,-}, \tag{14}$$

where $i \in \mathbb{N}_\nu, j \in \mathbb{N}_\mu$ and $x \in \{e-1, e, e+1\}$[14]. From now on, we will refer to these nodes as w-nodes. In order not to make notation too complicated, we do not add an additional index to each w-node which indicates to which block it belongs. It is important to note that there exist w-nodes in different blocks whose indices are the same, although no two w-nodes of G_e can have the same indices.

We now describe how to interconnect these structures. Before giving technical details, we discuss how information is transmitted informally, so the reader understands it intuitively and is able to understand why we choose to interconnect nodes using certain patterns before reading the end of the proof. Inside each block, there are v-nodes, w-nodes and copies of elements of $\mathbb{G}$. A cycle which simulates one simulation step of a Turing machine consists of $\Delta_M = \Delta + 3$ simulation steps of the graph. Before the first simulation step, information about the current global state is stored in v-nodes. This information is transmitted to w-nodes during the first simulation step of the graph (each w-node is fully influenced by its corresponding v-node, by using interfaces if necessary). During the second step, information stored in w-nodes is transmitted to the corresponding input nodes of all $\mathcal{G} \in \mathbb{G}$ by adding non-reflexive edges which copy information from w-nodes to input nodes every simulation step. Therefore, the second condition of Theorem 3 is still satisfied. Due to the way we have defined the sets $\mathbb{F}$ and $\mathbb{G}$, this implies the following: if at time t the input nodes of all graphs in $\mathbb{G}$ have coherent values which encode a certain state of progress of M, and constraints inside each graph are satisfied, then the output nodes of all graphs $\mathcal{G} \in \mathbb{G}$ encode the values of the next state of progress of M at time $t+\Delta$, where Δ is the propagation time of all elements of $\mathbb{G}$. So after $\Delta+2$ simulation steps of G, the initial state of progress of M stored in v-nodes yields the next state of progress, which is encoded in the output nodes of the elements of $\mathbb{G}$. Finally, this information is transmitted to v-nodes during a $(\Delta + 3)$-th simulation step. v-nodes are fully influenced by their corresponding output nodes. In conclusion, one transition between two states of progress of M is simulated in $\Delta + 3 = \Delta_M$ steps; the next state of progress is stored in v-nodes and overwrites the previous one.

We now describe the edges and the interfaces which interconnect nodes formally. The first interconnection we describe is the one which connects v-nodes to w-nodes. Each w-node is totally influenced[15] by the only v node with the same superscripts and subscripts. This implies that when simulating the graph, w-nodes have the value their corresponding v-node had one simulation step before. However, there is an issue which needs to be addressed; given a w-node, its corresponding v-node might not belong to the same block. This problem is solved by means of interfaces. Inside each block G_e, there are w-nodes for storing all information related to all variables of the e-th block and the block(s) next to it. In order to transfer this information from v-nodes to w-nodes, the interface which

[14] In G_1, x cannot equal $e - 1 = 0$.

[15] The weight of the edge which connects a v-node to a w-node equals 1.

connects two consecutive blocks G_e, G_{e+1} consists of $2(\mu+\nu+1)$ connections which connect v-nodes inside G_e to w-nodes inside G_{e+1}, as well as $2(\mu+\nu+1)$ analogous connections from v-nodes in G_{e+1} to w-nodes in G_e. v-nodes are thus connected to w-nodes by using edges whose weight is 1 as follows:

$$v_e^{p,*} \to w_e^{p,*}, v_{e,j}^{s,*} \to w_{e,j}^{s,*}, v_{e,i}^{a,*} \to w_{e,i}^{a,*}, \tag{15}$$

where $* \in \{+,-\}$ and there exist two or three edges which leave each v-node, since its two or three corresponding w-nodes can be found in G_{e-1} (if $e \neq 1$), G_e and G_{e+1}.

At this point, it might seem unnatural to introduce w-nodes, which seem not to be necessary since information could also flow from v-nodes to input nodes directly. We have decided to use them in order to simplify the construction of G, since trying to send the encoded state of the system to input nodes directly from v-nodes requires a much more detailed study of the interfaces. Using w-nodes as an intermediate step makes it possible to use exactly two interfaces per transmitted bit. The number of input nodes corresponding to each bit is not known in advance and might be different for the first block, since $\mathbb{G}_{\texttt{Fst}} \neq \mathbb{G}_{\texttt{Mid}}$, so it would be much more tedious to define a graph in which information is transmitted directly.

We now describe how to connect w-nodes to the input nodes of graphs in $\mathbb{G}$. Inside every block G_e, whose relative position is $\mathrm{Rel}(e) = R$, every two w-nodes (w_b^+, w_b^-) which represent a variable b are in turn connected to all their corresponding input nodes $v^+ \in V_b^+, v^- \in V_b^-$[16] inside each graph of $\mathbb{G}_R$. More formally, each graph $\mathcal{G} \in \mathbb{G}_R$ calculates one variable of the next state of progress of M by using the values of some of the variables in (12), obtained from its input nodes $v^+ \in V_b^+, v^- \in V_b^-$ for each variable b. All reflexive edges of these nodes present in the original construction of $\mathcal{G}$ (using the proof of Theorem 3) are removed and replaced by other edges, whose sources are w-nodes inside G_e, and whose weights equal 1. For each graph $\mathcal{G} \in \mathbb{G}$, this is done by introducing the three following types of edges.

$$w_x^{p,*} \to v, \ \forall v \in V_{p_t^x}^*, \tag{16}$$

$$w_{x,i}^{a,*} \to v, \ \forall v \in V_{a_{i,t}^x}^*, \tag{17}$$

$$w_{x,j}^{s,*} \to v, \ \forall v \in V_{s_{j,t}^x}^*. \tag{18}$$

Just like above, the character $*$ is used for both $+$ (for nodes whose values represent their corresponding variables) and $-$ (for nodes whose values represent the logical negation of their corresponding variables), and indices have the ranges $i \in \mathbb{N}_\nu, j \in \mathbb{N}_\mu, x \in \{e-1, e, e+1\} \setminus \{0\}$. These connections follow the same patterns in all blocks of the same type, so the obtained structure satisfies the regularity properties of $\mathbb{N}$-shaped graphs. As we stated above, these edges guarantee information flows from w-nodes to their corresponding input nodes inside the elements of $\mathbb{G}_R$.

We now describe the last connections we add to G to make it complete and simulate M as intended. These connections transmit the information about the next state of progress computed by graphs of $\mathbb{G}$ back to v-nodes.

[16] We remind the reader that notation V_b^+, V_b^- refers to the same notation from Theorem 3 applied to $\mathcal{G}$. w-nodes are connected to these sets for all copies of every $\mathcal{G} \in \mathbb{G}$ present in G.

Let us define some notation first in order to describe these edges. Every graph $\mathcal{G} \in \mathbb{G}$ is constructed from Theorem 3, so there exist output nodes (v_r^+, v_r^-) which store the value of a certain variable calculated by $\mathcal{G}$ after Δ simulation steps. Inside the block G_e whose relative position is $R \in \{\texttt{Fst}, \texttt{Mid}\}$, we define $\mathcal{S}^{p,+}$ and $\mathcal{S}^{p,-}$ to be the output nodes v_r^+ and v_r^-, respectively, of the copy of $\mathcal{G}_R^p$ in G_e. We use similar notation for the rest of the graphs: $\mathcal{S}_i^{a,+}$ and $\mathcal{S}_i^{a,-}$ are the output nodes of the copy of $\mathcal{G}_{R,i}^a$ inside G_e, and $\mathcal{S}_j^{s,+}$ and $\mathcal{S}_j^{s,-}$ are the output nodes of the copy of $\mathcal{G}_{R,j}^s$.

Inside G_e, we connect output nodes to v-nodes by using edges whose weight is 1 (so information in output nodes is fully transmitted to v-nodes). These connections are made naturally, by connecting the output nodes of every variable to the v-nodes of the same variable, as follows.

$$\mathcal{S}^{p,*} \to v_e^{p,*}, \mathcal{S}_j^{s,*} \to v_{e,j}^{s,*}, \mathcal{S}_i^{a,*} \to v_{e,i}^{a,*}. \tag{19}$$

We have now completely described G and all the edges which connect v-nodes, w-nodes and copies of $\mathbb{G}_R$ to each other. We have already discussed why G satisfies the conditions of Theorem 4 informally. After describing G formally, we now explain why G simulates an arbitrary semi-infinite tape Turing Machine M.

Taking $\Delta_M = \Delta + 3$ in Theorem 4, where Δ is the propagation time of all graphs in $\mathbb{G}$, and given an initial state S_0 which represents a certain state of progress $\alpha = \left(q_0, \{d_n\}_{n\in\mathbb{N}}, x_0\right)$, we know that the v-nodes represent α, just like in the first three conditions of Definition 8. Also, the labels of all vertices $v \in C$ are the labels assigned by S^C. Due to how we have defined C and S^C above in G, this basically means the constraints of all graphs in $\mathbb{G}$ are satisfied. Therefore, these graphs compute propositional logic formulas whose variables are read from input nodes and whose result is produced in output nodes after Δ steps. Inside each copy of a graph from $\mathbb{G}$, we have only changed the way input nodes are influenced when defining G. This means all constraints of all graphs will always be satisfied for all successive states of S_0, so the values of all propositional logic formulas $\mathcal{F} \in \mathbb{F}$ will be computed every simulation step, with a delay of Δ steps.

Since v-nodes are connected to w-nodes as we described in (15), w-nodes have the value their corresponding v-node had in the previous simulation step of G. Therefore, after one simulation step $S_0 \to S_1$, w-nodes encode the state α. Also, since w-nodes are connected to the corresponding input nodes of all graphs, like we described in (16), (17) and (18), all input nodes of all graphs in G have the values of all variables which encode α after two simulation steps (S_2). As we have explained before, the way we have defined $\mathbb{F}$ and $\mathbb{G}$, the construction of G and the assumptions about S_0 imply that, after Δ simulation steps ($S_2 \to S_{\Delta+2}$), all information about the next state of progress of M has been computed and is stored in the output nodes of the graphs of $\mathbb{G}$. Finally, since we have connected these output nodes to v-nodes as in (19), that is, by connecting nodes which are related to the same variable of the next state of progress of M, this information is transferred back to v-nodes after one simulation step ($S_{\Delta+2} \to S_{\Delta+3}$). Let us summarise: after $\Delta + 3 = \Delta_M$ simulation steps, Definition 8 is satisfied, but instead of representing the global state α, $S_{\Delta+3}$ represents the next global state. By using induction, we conclude that after simulating the evolution of G for $\Delta_M \cdot t$ steps starting from the labelling S_0, the information in v-nodes encodes the global state of M obtained after t iterations of the machine, if the initial global state was α.

We end this section with some remarks which we are not going to discuss and prove in

detail. First and foremost, it is important to note that the construction of G explained above can be modified in order to simulate an infinite tape in both directions. This is done by using a $\mathbb{Z}$-shaped graph in which only a general block exists ($G_{\texttt{Mid}}$). It can also be made finite, by generating a finite number of blocks whose relative position is `Mid` and adding a block at the end, G_{r+1}, whose outgoing interfaces are always sending the information $p_t^{r+1} = \perp$ back to G_r. This graph simulates a Turing machine which can only use a finite amount of space, and whose head *disappears* if it tries to move beyond the r-th cell. If we consider the last two blocks as one single block whose relative position is `Lst`, we can simulate an instance of M which uses no more than r cells by using an $\mathbb{N}_r$-shaped graph. This fact is quite important, so we state it here as a theorem we will refer to later.

Theorem 5 (Semi-infinite tape Turing machines which use r cells can be simulated by $\mathbb{N}_r$-shaped opinion graphs)**.** *Given a semi-infinite tape Turing machine M and $r \in \mathbb{N}$, there exist a $\Delta_M \in \mathbb{N}$ and an $\mathbb{N}_r$-shaped opinion graph G with disjoint subsets of nodes of G_B^c*

$$\left\{v_{\texttt{Fst}}^{p,+}, v_{\texttt{Fst}}^{p,-}\right\}, \left\{v_{\texttt{Fst},i}^{s,+}\right\}_{i=1}^{\mu}, \left\{v_{\texttt{Fst},i}^{s,-}\right\}_{i=1}^{\mu}, \left\{v_{\texttt{Fst},i}^{a,+}\right\}_{i=1}^{\nu}, \left\{v_{\texttt{Fst},i}^{a,-}\right\}_{i=1}^{\nu}, C_{\texttt{Fst}}, \tag{20}$$

and labellings S_B^C of C_B for $B \in \{\texttt{Fst}, \texttt{Mid}, \texttt{Lst}\}$, such that given an initial state of progress $\alpha = \left(q_0, \{d_n\}_{n\in\mathbb{N}}, x_0\right)$, starting from which the head of M never moves beyond the r-th cell, and a labelling S_0 of G, if S_0 represents α, then $S_{\Delta_M \cdot t}$ represents the state of progress obtained starting from α after simulating M for t steps.

Definition 8 only applies to $\mathbb{N}$-shaped opinion graphs. However, it can be restricted for $\mathbb{N}_r$-shaped graphs, so the notion of representing a global state is well defined in Theorem 5.

The second remark related to Theorem 4 is the fact that the construction of G can be modified so we only need basic local initial conditions; a *reset* signal could be sent from left to right by using appropriate additional nodes, which would initialise the contents of the tape and set $p_t^e = \perp$ for blocks to which the reset signal arrives. This signal or a finite number of different signals could be used to initialise nodes in C, to make sure logic gates work as intended after receiving the reset signal.

Furthermore, simulating G using limited resources every step can be done by assuming all nodes have constant values which are the same across different blocks and never change beyond the rightmost block which has ever been visited by the head. These labels should be chosen to represent the head not being present and a *blank symbol* on the tape.

Last but not least, there is an important remark which is indispensable for the proofs of some results we state and prove below. These proofs rely on the construction we explain for proving Theorem 4. In this construction, v-nodes, which *display* the state of progress of M every Δ_M simulation steps, transmit all their information to w-nodes every simulation step, and this transmission of information is unconditional and only depends on the values of v-nodes. In the next simulation step, information is transmitted from w-nodes to input nodes of elements of $\mathbb{G}$. This transmission of information is unconditional as well, and is not corrupted if the values of v-nodes are manually modified after the first simulation step, since all information has already been transmitted to w-nodes.

Theorem 3 implies information about the next global state of M will be transmitted to the output nodes of elements of $\mathbb{G}$ during the next $\Delta = \Delta_M - 3$ simulation steps, even if the values of v-nodes are manually modified during these Δ cycles; Theorem 3 guarantees

the values of variables which have already reached input nodes will not be altered, since v-nodes do not belong to any element of $\mathbb{G}$.

This independence makes it possible to simulate Δ_M different instances of M by using a single simulation of G, as follows: in order to initialise G with the states of progress $\{\alpha_1, \alpha_2, \ldots, \alpha_{\Delta_M}\}$, we start from a labelling S for which $\forall\ v \in C_e$, $S(v) = S^C(v)$ (so logic gates function properly) and apply the following algorithm to G.

Algorithm 1 Initialising G to simulate Δ_M instances simultaneously starting from the states of progress $\{\alpha_1, \alpha_2, \ldots, \alpha_{\Delta_M}\}$

1: **for** $i = 1, 2, \ldots, \Delta_M$ **do**
2: Replace the labels of v-nodes so the labelling of G now represents α_i
3: Advance the simulation of G one step
4: **end for**

Theorem 4 and the independence properties we have just discussed imply all instances are simulated at the same time and are represented in v-nodes cyclically. When replacing the labels of v-nodes in Algorithm 1, instances which have already been inserted are not corrupted, since the simulation has advanced, but not more than $\Delta_M - 1$ steps. At the end of Algorithm 1, Δ_M simulation steps have passed since α_1 was introduced in G, so the next state of progress is represented in v-nodes. During the next iteration consisting of Δ_M simulation steps, the next states of progress corresponding to all initial global states $\{\alpha_1, \alpha_2, \ldots, \alpha_{\Delta_M}\}$ are represented in G in order of insertion. This process repeats every Δ_M simulation steps and simulates every instance one step further.

In order to use this parallel simulation and conceptually describe which instances we intend to simulate in specific cases, we use the term *level of information* when referring to one of the Δ_M available slots there are for running an instance of Δ_M. The first one consists of information stored in v-nodes, the second one consists of information stored in w-nodes, and so on. Conceptually, information moves to the next level every simulation step, and transforms to represent the next state of progress when moving from the last level to the first one. Of course, it can be decided to simulate a single instance by initialising v-nodes correctly in S_0 and initialising all other nodes which are not in C to random values.

3.3 Decidability and complexity properties of opinion graphs

The results we have obtained in previous sections are very useful for analysing the complexity properties of opinion graphs in general, and those of L-shaped graphs in particular. The two key results we have obtained so far are Theorem 4 and Theorem 5, from which all relevant complexity results about opinion graphs can be deduced. We begin this subsection by discussing general properties infinite opinion graphs have, and end it determining the complexity of simulating finite L-shaped opinion graphs.

Theorem 4 implies L-shaped opinion graphs can be as expressive as Turing machines. We have also mentioned it is not necessary to initialise an infinite number of nodes in order to obtain this expressiveness; initialising a finite number of nodes at the leftmost block of an $\mathbb{N}$-shaped graph suffices. This leads us to the following theorem.

Theorem 6 (Limit properties in L-shaped graphs are undecidable)**.** *Given an L-shaped graph G and a finite or infinite initialisation condition*[17] *for G, the following problems are undecidable in general.*

(a) Given a node v of G, does there exist a certain labelling S_0 of G, which respects the initialisation condition, and a certain $t \in \mathbb{N}$, for which $S_t(v) = 1$?

(b) Does there exist a certain labelling S_0 of G, which respects the initialisation condition and eventually stabilises ($\exists t \in \mathbb{N} \mid S_{t+1} = S_t$)?

(c) Given a finite set Z of nodes of G, does there exist a certain labelling S_0 of G, which respects the initialisation condition and for which it is possible to estimate the asymptotic proportion of nodes which agree (or disagree) in Z?

Proof. (a) Consider the universal semi-infinite tape Turing machine $\mathcal{U}$ defined in [4]. This machine accepts the input $\langle x, \alpha \rangle$ if the machine which is represented by α accepts x. We define a Turing machine $\mathcal{U}_a$ whose behaviour is slightly different from that of $\mathcal{U}$. For our purpose, the input of $\mathcal{U}_a$ always begins with the special tape symbol $\sigma_\perp$ (which is not used elsewhere), and is then followed by $\langle x, \alpha \rangle$. $\mathcal{U}_a$ simulates the instance $\langle x, \alpha \rangle$, just like $\mathcal{U}$, but without modifying the first cell of the machine. If $\mathcal{U}$ accepts $\langle x, \alpha \rangle$, then $\mathcal{U}_a$ also does, but moves its head to the first cell and overwrites it with the symbol $\sigma_\top$ (also never used elsewhere) before stopping completely. For problem (a), we define G to be the graph which is constructed using the proof of Theorem 4 with $M = \mathcal{U}_a$. We assume without loss of generality that the following two properties are true.

- If $\sigma_t^x = \sigma_\top$, then $a_{1,t}^x = 1$, and if $\sigma_t^x \neq \sigma_\top$, then $a_{1,t}^x = 0$. In other words, the first bit used to encode $\sigma_\top$ is 1, while it is 0 for all other alphabet elements.
- G can be initialised locally by appropriately setting the values of the nodes in the first m cells, and it will simulate $\mathcal{U}_a$ correctly, since it can correctly interpret a *reset* signal sent from G_{m+1}.

Given a general input $\langle x, \alpha \rangle$, Theorem 4 and related results provide a finite[18] or infinite[19] initialisation condition of G which makes it possible to simulate how $\mathcal{U}_a$ processes $\langle x, \alpha \rangle$. The initialisation condition makes sure logic gates work as expected, and saves the initial state, place of the head and contents of the tape in all levels of information of G, or in the one which corresponds to v-nodes.

Note that due to the encoding of tape symbols we have assumed (first assumed property) and how $\mathcal{U}_a$ works, $\langle x, \alpha \rangle$ being accepted by $\mathcal{U}$ is equivalent to the first

[17] This means the initial labelling of the L-shaped graph is a certain S_0, and a possibly infinite list of conditions of the form $S_0(v_i) = l_i$, $S_0(v_i) = S_0(v_j)$ or $S_0(v_i) = \neg S_0(v_j)$ is given. We formulate this theorem in a very general way, allowing incomplete descriptions of S_0, in order to emphasise that the undecidability of the problems we consider is **not** related to infinite initialisation conditions which cannot be fully processed in a finite amount of time.

[18] This initialisation can be specified by using a finite initialisation condition, due to the second assumed property.

[19] The infinite case is also included here, since arbitrarily defining more nodes starting from a finite initialisation condition merely gives more details about the initial state of progress of M, without changing the answer to the problem.

symbol of the tape of $\mathcal{U}_a$ eventually becoming $\sigma_\top$, which is in turn equivalent to the label of $v_{1,1}^{a,+}$ (the v-node which codifies the first bit of the symbol on the first cell) eventually taking the value 1 forever *during the simulation steps in which v-nodes represent the global state of* $\mathcal{U}_a$. Therefore, the behaviour of $\mathcal{U}$ on any input of the form $\langle x, \alpha \rangle$ can be reduced to problem (a) *for a certain subset of simulation steps*, making it undecidable as well.

We are not done with the proof of (a) yet, as we know that $v_{1,1}^{a,+}$ only represents one of the variables of the state of the system every $\Delta_{\mathcal{U}_a}$ steps, so it might be possible to prove the label of $v_{1,1}^{a,+}$ equals 1 in a simulation state in which v-nodes do not represent the state of progress of $\mathcal{U}_a$. However, as we discussed before, the way we have constructed $\mathcal{U}_a$ in Theorem 4 makes it possible to simulate $\Delta_{\mathcal{U}_a}$ instances simultaneously, and exactly one of them is represented in v-nodes every cycle. If $\Delta_{\mathcal{U}_a}$ copies of the same instance are simulated in all $\Delta_{\mathcal{U}_a}$ levels of information, the label of $v_{1,1}^{a,+}$ becomes 1 if and only if $\sigma_\perp$ is replaced by $\sigma_\top$ on the first cell of $\mathcal{U}_a$. This makes it possible to guarantee that the initial condition of G chosen for $\langle x, \alpha \rangle$ extends the equivalence discussed before to all simulation cycles of G. That implies the behaviour of $\mathcal{U}$ on arbitrary inputs can be reduced to problem (a) the usual way, so problem (a) is undecidable.

(b) For proving (b), we use some ideas from (a), but change the construction of the semi-infinite tape Turing machine. In this case, we choose $\mathcal{U}_b$ to be a Turing machine whose global state stabilises[20] when its input is $\langle x, \alpha \rangle$ if and only if $\mathcal{U}$ accepts $\langle x, \alpha \rangle$. Constructing $\mathcal{U}_b$ can be done in many different ways; inserting cyclic loops in the transition function for stationary global states in which $\langle x, \alpha \rangle$ is not accepted, is one possible approach. We now define G to be the graph which is constructed taking $\mathcal{U}_b$, not $\mathcal{U}_a$. Using the same argument as in (a), it is possible to simulate $\Delta_{\mathcal{U}_b}$ *equal* instances of $\mathcal{U}_b$ in G.

If all simulated instances are equal, G eventually stabilises if and only if the state of progress of $\mathcal{U}_b$ stabilises; it is obvious that G does not stabilise if one of the instances does not stabilise, and if they are all equal and eventually stabilise, G stabilises, since each level of information of G eventually contains the information related to the stable state of progress, which does not change when instances flow through the levels of information of G, as all instances attain the stable state of progress. This leads us to a reduction which maps the behaviour of $\mathcal{U}$ for an arbitrary given input $\langle x, \alpha \rangle$ to problem (b). Therefore, (b) is undecidable.

(c) In general, this problem is undecidable, even if Z is arbitrarily large. We choose $\mathcal{U}_c$ such that, given input $\langle x, \alpha \rangle$, the machine eventually writes the symbol $\sigma_\perp$ on all cells from left to right if $\mathcal{U}$ does not accept $\langle x, \alpha \rangle$, while $\sigma_\top$ is written if $\mathcal{U}$ does accept. In order to implement this, the symbol $\sigma_\perp$ can be written progressively from left to right while $\mathcal{U}$ is simulated on the right, and if it is seen that $\mathcal{U}$ accepts the given input, then the head goes to the beginning of the tape and overwrites everything using $\sigma_\top$. Analogously, we now use G to denote the graph constructed using $\mathcal{U}_c$. If, as in the previous problems, we assume the same input is provided in

[20] By definition, M stabilises if and only if the state of progress of M stops changing, i.e. the head of M stays in the same place, the m-configuration does not change and the symbol on the tape is not modified.

all levels of information of G and choose $Z_k = \left\{v_{e,1}^{a,+}\right\}_{e=1}^{k}$ [21], then the values of all nodes of Z_k will eventually be 1 if $\langle x, \alpha \rangle$ is accepted by $\mathcal{U}$, since all tape symbols will eventually become $\sigma_\top$, or 0 if $\langle x, \alpha \rangle$ is not accepted. This construction reduces the same undecidable problem considered in (a) and (b) to problem (c), which is thus undecidable.

□

Even though Theorem 6 might seem natural given the results we obtained in Section 3.2, it is not a direct consequence of any previously proved theorem, and looks very discouraging at first glance: it implies the problem of simulating opinion graphs and deducing their behaviour, which can be directly applied to model opinion dynamics in modern societies, is undecidable, and we are not even able to give global quantitative estimations of the graph's evolution for arbitrarily large amounts of time.

We now state and prove the finite version of this result. Just like Theorem 4 was used for proving Theorem 6, the proof of the following theorem heavily relies on the corresponding finite version of Theorem 4, which is Theorem 5. We first give a definition which provides a way to generate L-shaped graphs with the same invariant local structure. This definition is closely related to Definition 5, and is very relevant for a special case of the theorem we discuss below.

Definition 9 (L-shaped graphs of variable size)**.** *Given $d \in \mathbb{N}$, interfaces $\{I^j\}_{j \in \mathbb{N}_d}$ and finite opinion graphs G_R for every $R \in \{\texttt{Fst}, \texttt{Mid}, \texttt{Lst}\}^d$ which are multiply connected to the following interfaces,*

(a) If the j-th element of R is not `Lst`*, then G_R must be connected to I^j as a predecessor.*

(b) If the j-th element of R is not `Fst`*, then G_R must be connected to I^j as a successor.*

we define, for each $L = E_1 \times E_2 \times \cdots \times E_d$ (where each E_i equals $\mathbb{N}_{s_i}$ for certain $s_i \geq 3$), the graph $G(L)$ as the finite L-shaped graph obtained by using the construction from Definition 5.

We define $|L|$, the block cardinality of $G(L)$, to be the cardinality of $L = \prod_{i=1}^{d} E_i = \prod_{i=1}^{d} \mathbb{N}_{s_i}$, which equals $\prod_{i=1}^{d} s_i$.

We state almost all of the properties related to finite graphs in one single theorem, since they are closely related to each other and we obtain the same complexity class. We remind the reader some assumptions we make. These assumptions are natural and used in [4].

(1) The way Turing machines are encoded guarantees that given a machine's description by listing the tables of the transition functions, there exists a polynomial-time algorithm which encodes it. Similarly, the algorithm for decoding is also polynomial (the number of alphabet symbols and states is bounded by a polynomial function of the length of the string which represents the semi-infinite tape Turing machine).

[21] Z_k is the set of v-nodes which encode the first bit of the symbol written on the tape, for the first k cells. Its cardinality is k.

(2) When encoding opinion graphs in a language, we list the nodes and edges of the graph, so this representation is a polynomial function of the number of nodes. We also list all the rational weights of all edges. The representation used for rational numbers guarantees that operating with them is possible by using polynomial-time algorithms.

Problems (c) and (d) of Theorem 7 are not decision problems. However, they can trivially be transformed to a number of decision problems which is linear in the input of problems (c) and (d). Besides, problems (c) and (d) are very practical and have many applications. Therefore, we state them as optimisation problems. What we prove is that obtaining all bits of their solutions can be done using a polynomial amount of space, and they are also **PSPACE**-hard. This approach is equivalent to decision problems, as defined in [4].

Theorem 7 (**PSPACE**-completeness of global properties of L-shaped graphs)**.** *The following problems, whose inputs are a finite L-shaped graph G and an initialisation condition for G, are **PSPACE**-complete.*

(a) Given a node $v \in G$, does there exist a labelling S_0 of G, which satisfies the initialisation condition, and a $t \in \mathbb{N}$, for which $S_t(v) = 1$?

(b) Does there exist a labelling S_0 of G which satisfies the initialisation condition and eventually stabilises?

(c) Of all possible initial labellings S_0^α which satisfy the initialisation condition, which one maximises the proportion of nodes whose value is 1 at time t[22]? Which maximum proportion and corresponding initial and final graphs are obtained? What about all possible times?

(d) Suppose there exist polynomial-time functions which calculate the cost of initial labellings, and the benefit of labellings at time t. Which initial labelling, with cost not higher than c and which satisfies the initialisation condition, maximises the benefit at time t? Which maximum benefit and corresponding initial and final graphs are obtained? What about all possible times?

Proof. First of all, we check all of these problems belong to **PSPACE**. Simulating opinion graphs for an arbitrary number of steps can be done in polynomial space, due to our assumptions. Problem (a) can be solved using polynomial space by considering all possible initial labellings one at a time and simulating all of them. Binary counters can be used to detect when to stop simulating, since the number of simulation steps eventually exceeds the number of possible steps of the graph. It is well-known and discussed in [4] that these counters use a linear amount of space. Saving the labelling S_0 also uses linear space. Problem (b) can be solved similarly; an initial labelling is discarded if it has not converged when the number of steps attains the number of possible labellings of G.

Problem (c) is a special case of problem (d), which can be solved by considering all possible initial labellings, one at a time, simulating them if they satisfy all requirements and keeping the best initial and final labellings depending on the outcome at time t. If

[22]In (c) and (d) of Theorem 7, t is given as a binary number.

the best result is needed for all times $t \in \mathbb{Z}_{\geq 0}$, then the simulations can be run until the number of simulation steps exceeds the number of possible configurations of the graph. This can be done using a polynomial amount of space, as explained in [4].

We now show these problems are **PSPACE**-complete by showing that $L_{accsp} \leq_p L$ for all of the considered L associated with the problems described in (a-d), where L_{accsp} is the language defined in Equation (1), in Part 1.1.1 of the introduction. Given an instance $(M, w, 1^n)$ of this problem, it is possible to transform it to $(M', w, 1^{p(n)})$ using a polynomial-time algorithm, so that if M accepts w in space n, then M' accepts w and the first cell, with an initial value of $\sigma_\perp$, changes once to $\sigma_\top$, and the global state of M' stabilises, while M' does not accept, the first cell keeps its initial value of $\sigma_\perp$ and the global state does not stabilise if not. We know by Theorem 5 that this problem can be transformed to an opinion graph G which simulates M', and due to the first transformation from M to M', the simulation of G leads to the answer to the problem L_{accsp} for $(M, w, 1^n)$. After the transformation from $(M, w, 1^n)$ to G, all problems (a-d) can be used to determine if $(M, w, 1^n) \in L_{accsp}$; the problem described in (a) can be used by taking $v = v_{1,1}^{a,+}$ and using the same construction as in Theorem 6. The problem described in (b) can also be used, since G converges if and only if the global state of M' converges[23]. Problems (c) and (d) can be used to analyse G by looking at the final opinion graph after simulating, which these problems obtain. For all problems (a-d), the initialisation condition we use consists of the initial assignments when constructing G using Theorem 5 to make sure logic gates work, as well as the construction from the proof of Theorem 6, which assigns fixed values to all levels of information. This initialisation condition assigns a value to each node, which implies no freedom to modify the nodes' labels is given. For problems (c) and (d), this means a completely determined graph is simulated, and whether or not M' accepts is deduced by looking at the final label of $v_{1,1}^{a,+}$, which is calculated by problems (c) and (d) as part of the final graph. Regarding problems (a) and (b), the completely determined graph is simulated as well and the definition of M' given input $(M, w, 1^n)$ implies that M' accepts if and only if problems (a) and (b) are true.

All we need to show in order to finish this proof is that transforming an instance $(M, w, 1^n)$ to its corresponding G can be done in polynomial time. This follows from previous theorems we have already discussed. First of all, the construction used in Theorem 3 transforms propositional logic formulas to opinion graphs in polynomial time, since in most of the construction a constant number of nodes and edges is added for each symbol in a formula. There is an exception, which is when chains of nodes are added to achieve equal delays for different opinion graphs. However, the length of these chains is a linear function of the delay of an already constructed graph. This makes the construction quadratic in the length of the formulas. Also, it is shown in [4] that there exist polynomial-time algorithms to transform a semi-infinite tape Turing machine M to the set of logic formulas $\mathbb{F}_M$ which determine its transition functions. All these formulas are then transformed to the corresponding set of graphs $\mathbb{G}_M$ from Theorems 4 and 5 using the polynomial-time algorithm discussed in 3. These graphs are then copied not more than n times to construct G, so the time complexity remains bounded by a polynomial. Initialising

[23] Of course, in order to make sure G converges if and only if the global state of M' converges, it is necessary to use the same type of initialisation as in Theorem 6; all levels of information have to simulate the same instance.

nodes as described in the proofs of Theorems 4 and 6 by using Algorithm 1 can also be done in polynomial time. Therefore, transforming $(M, w, 1^n)$ to G is possible using a polynomial-time algorithm, which implies all problems discussed in this theorem are **PSPACE**-complete. □

This theorem is very discouraging as well; important questions about arbitrary graphs cannot be easily answered. This might make it extremely difficult to understand opinion dynamics in real life. Nevertheless, this task might turn out to be much easier in real-life conditions. After all, the graphs we are considering in the proof of Theorem 7, which are first defined in the proof of Theorem 3, are clearly very different from real-life social graphs; all nodes influence *exactly* one node, apart from themselves, and many of them just receive an opinion from one node during a cycle and pass it on to another node.

It is important to note that the proof given above can be applied to finite opinion graphs in general; although the arguments used to prove polynomial reduction from L_{accsp} to the problems described in Theorem 7 rely heavily on repetitive patterns in $\mathbb{N}$-shaped graphs and the constructions discussed in previous sections, proving that all problems from Theorem 7 belong to **PSPACE** can be done without assuming that G is an L-shaped graph. Therefore, the space which is needed to simulate an arbitrary opinion graph to obtain conclusions about its behaviour is a polynomial function of its number of vertices, and nothing can be said about the time complexity, apart from what is implied by Theorem 1, since we now know the general problem is **PSPACE**-complete. Of course, there are several special cases for which the complexity can be determined; one of them is Corollary 2. If the initial condition of G completely determines the value of all nodes and satisfies Corollary 2, then it is possible to simulate the evolution of G in polynomial time, since the number of simulation steps before the labels of nodes in G converge does not exceed the number of vertices[24].

We now state and prove a specific case of Theorem 7 which can be used when the structure of the L-shaped graph is fixed. Although this Corollary is predictable, it provides more precise space bounds for the fixed structure case, so we state it for completeness.

Corollary 3 (Space bounds for L-shaped graphs of variable size)**.** *Given $d \in \mathbb{N}$, interfaces $\{I^j\}_{j\in\mathbb{N}_d}$ and finite opinion graphs G_R for every $R \in \{\texttt{Fst}, \texttt{Mid}, \texttt{Lst}\}^d$ as in Definition 9, there exist Turing machines M_a, M_b, M_c and a constant K such that problems (a), (b) and (c) from Theorem 7 can be solved by M_a, M_b, M_c respectively, using no more than $K \cdot |L|$ cells if the input graph of the problem is $G(L)$, for all $L = E_1 \times E_2 \times \cdots \times E_d$.*

Proof. Iterating through all states, saving states, counting the number of nodes which agree for a certain state and counting the number of simulation steps can all be done in linear space. All we need to prove is that the graph can be saved in linear space and be simulated in linear space, as a function of $|L|$. Since the local structure of the graph is fixed, it is possible to make the alphabet of the Turing machines rich enough to save two copies of the following information as a tape symbol.

[24]Suppose G has n_V nodes, and it takes $n_C > n_V$ simulation steps for their labels to converge. Then, since every node $v \in G$ *flips* its opinion at most once due to Corollary 2, not more than n_V opinion changes take place in strictly more than n_V simulation steps, during which the graph has not stabilised yet. That is a contradiction, since the pigeonhole principle implies there have been no opinion changes during a certain cycle, which is impossible, since that implies G has actually converged before n_C steps.

(a) The state of the nodes belonging to a block G_e.

(b) Information about the relative position of G_e, which is given by an element of $\{\texttt{Fst}, \texttt{Mid}, \texttt{Lst}\}^d$.

(c) A Boolean $\text{it}(G_e)$, which is usually $\bot$.

Blocks of the graph can be encoded using linear space by using this method and saving them in lexicographical order. In order to simulate the graph, the machine can iterate through all blocks of the graph and compute their next state, saving a copy of it in each block without destroying the copy of the current state, which needs to be saved to compute the next state of other blocks.

In order to calculate the next state of a block G_e, information about nodes which are connected to incoming interfaces of G_e is needed. If this information is obtained, the transition functions of the nodes of G_e can be encoded in M_a, M_b, M_c, since the structure of the blocks is fixed. In order to obtain this information for a certain dimension d, the Turing machines can set $\text{it}(G_e) = \top$ to remember its position and move their heads to the left and to the right in order to find the positions at which the position of the block in dimension d changes (this can be done by looking for blocks whose relative positions are all `Fst` for dimensions which have a lower priority than d in the lexicographic order), and set their *it* Boolean to $\top$ as well. In order to find the blocks which are above and below G_e in dimension d, the Turing machines need to find the blocks which are at the same distance of the following increase (or decrease) in the value of dimension d as G_e. This calculation can be done without using counters, by marking blocks (modifying their *it* value, or even using a constant number of additional Boolean variables inside every block). This makes it possible to find all the needed information to compute the next state of every block without using more than $|L|$ cells, so simulating the graph, and thus problems (a), (b) and (c) from Theorem 7 can be solved using a linear amount of space. □

Of course, similar space bounds for problem (d) from Theorem 7 can be calculated, as long as bounds are given for the space used by the cost and benefit functions.

The last theorem of this part of the work provides the complexity class of simpler versions of the problems discussed above. We remind the reader that the language

$$L_{tmsat} \stackrel{\text{def}}{=} \left\{ \langle \alpha, x, 1^n, 1^t \rangle \mid \exists u \in \{0,1\}^n \text{ s. t. } M_\alpha \text{ accepts 1 on input } \langle x, u \rangle \text{ within } t \text{ steps} \right\}$$

is **NP**-complete. For a proof of this fact, see [4]. We also remind the reader that we assume certain standard properties of the representation of opinion graphs and Turing machines, just like we explained before Theorem 7.

Theorem 8 (**NP**-completeness of local properties of L-shaped graphs)**.** *The following problems, whose inputs are a finite L-shaped graph G, an initialisation condition for G and a positive integer $t \in \mathbb{N}$ which is given in unary*[25]*, are **NP**-complete.*

(a) Given a node $v \in G$, does there exist a certain labelling S_0 of G which satisfies the initialisation condition and for which $S_t(v) = 1$?

[25] In the input, t is given as 1^t. This fact is extremely important, since the problems are not **NP**-complete if t is given in binary.

(b) Does there exist a labelling S_0 of G which satisfies the initialisation condition and is stable at time t?

(c) Given a rational number $r \in [0, 1]$, does there exist a labelling S_0 of G which satisfies the initialisation condition and such that the proportion of nodes whose value is 1 in S_t is greater than or equal to r?

(d) Suppose there exist polynomial-time functions which calculate the cost of initial labellings, and the benefit of labellings at time t. Given $c, b \in \mathbb{Q}$, does there exist a labelling S_0 of G which satisfies the initialisation condition and such that

$$cost(S_0) \leq c \text{ and } benefit(S_t) \geq b?$$

Proof. All of these problems are in **NP**. If the labelling S_0 is given for problem (a), simulating one cycle of G can be done in polynomial time, since the simulation rules have polynomial complexity. Simulating t cycles is still feasible in polynomial time, since t is given in unary as part of the input. Problem (b) can also be checked given S_0 in polynomial time, since simulating for t steps and checking stability once can be done in polynomial time.

Problems (c) and (d) also consist of simulating for a unarily given period of time and computing polynomial-time functions, if S_0 is given to check it is a valid initialisation condition.

We now prove all of these problems are **NP**-hard. We proceed as in Theorem 7. We already know from previous results (Theorem 7 in particular) that transforming the description of a Turing machine and a given input to its corresponding opinion graph can be done in polynomial time. This applies to this theorem, since machines which run in polynomial time also run in polynomial space.

Given an instance $\langle \alpha, x, 1^n, 1^t \rangle$ of L_{tmsat}, it can be transformed to a modified instance $\langle \alpha', \sigma_\perp x, 1^n, 1^{p(t)} \rangle$ which accepts if and only if the original instance accepts, where p is a polynomial and such that if this new instance accepts, it has stabilised at time $p(t)$ and has changed the value $\sigma_\perp$ to $\sigma_\top$ in the first cell, while it does not stabilise at time $p(t)$ and the symbol $\sigma_\perp$ does not change in the first cell if not. This instance can be transformed to a graph G in polynomial time. For this theorem, we choose the initial condition such that the values of all nodes are determined, except for those nodes which correspond to the symbols of u. We also introduce initialisation conditions of the type $S_0(v_i) = \neg S_0(v_j)$ on these nodes so that the introduced variables are coherent (two nodes which belong to the same variable have opposite labels). After this transformation is done, problems (a) and (b) can be used to determine if $\langle \alpha', \sigma_\perp x, 1^n, 1^{p(t)} \rangle$ belongs to L_{tmsat} or not using the same arguments and assumptions as in Theorem 7, because of the equivalences and how the initial conditions have been defined; these only make it possible to vary the nodes which correspond to the symbols of u.

For problems (c) and (d), a similar construction can be used. For these problems we need to add a special node in the first block which is influenced by the nodes whose variables are the internal state of M', such that if M' accepts, its value becomes 1, but it does not if M' does not accept. Since all other nodes of the opinion graph are associated with another node whose label is the opposite, due to the constructions from Theorem 3 and Theorem 4, the proportion of nodes which agree is strictly greater than $1/2$ if the

value of this node is 1, while it is not if the value of this special node is 0. Therefore, problems (c) and (d) can simulate the opinion graph obtained from M' using the same initialisation condition as for problems (a) and (b), and if M' accepts for a certain u, then the proportion of nodes which agree will be greater than $1/2$. This can be expressed in terms of cost and benefit by not using the cost and taking the benefit as the proportion of favourable nodes. The reductions for all problems (a-d) are thus complete. □

As in Theorem 7, we note that this theorem also applies to general opinion graphs; the four problems are **NP**-complete in general for arbitrary opinion graphs which are represented as a set of vertices and a set of weighted edges.

4 Heuristic analysis of real-world graphs

After analysing the theoretical complexity of opinion graphs, we generate instances which are similar to real-world cases by taking into consideration the properties we discussed in Subsection 1.1.3. We try to use several non-exhaustive algorithms to solve a particular case of a problem we have discussed in the previous section. Our objective is clear; we want to be able to obtain satisfactory results by using heuristic approaches, since the known correct methods for solving the problems we have discussed are not feasible if the input size is large.

We explain the problem and the algorithms we have used, and provide the numerical results we have obtained as well. Several relevant conclusions can be obtained from the behaviour of the algorithms we analyse. The main conclusion is clear; we are indeed able to obtain good results when using heuristic approaches.

4.1 The problem we consider

We want to construct graphs whose properties are similar to the three properties explained in Subsection 1.1.3. These are a low average shortest path, a high clustering coefficient and the distribution of node degrees. For this purpose, we create a type of graph which is similar to L-shaped graphs. On the one hand, it consists of blocks which are interconnected to each other in a two-dimensional lattice-like structure. In order to make these connections as regular as possible, blocks at the edges of the lattice are connected to blocks at the opposite edge, creating a structure which is topologically similar to a torus. Apart from this structure, additional edges, which interconnect nodes globally, are added. These edges ensure that the average shortest path length is low compared to that of the regular lattice. They are added following a version of the preferential attachment rule discussed in [3]. That way, the graph is much more similar to a real-world graph, since this creates a lot of inequality in the frequency distribution of the nodes' degrees, which follows a power law, as described in [3].

Edges are directed and added to the graph using natural positive weights. After all edges have been added, the graph is *homogenised*; all the weights of incoming edges of every node are divided by the sum of the incoming weights, so the sum of the weights becomes 1. Then, the rule described in Section 2 is applied to make the graph evolve in time.

The practical analysis of this section focuses on graphs with specific characteristics. These characteristics are modified for certain parts of the analysis, but the default ones are used unless stated otherwise. The general version of the graph consists of 2500 blocks (50 blocks wide, 50 blocks high). Every block contains 10 nodes, and each node has 15 internal incoming edges. The sources of these edges are chosen randomly inside the block, and the weights before homogenising are chosen randomly and uniformly from the set $W_e \stackrel{\text{def}}{=} \{3, 4, 5, 6, 7\}$. When several edges happen to have the same source and destination, it is as if they were a single edge whose weight is the sum of the individual weights.

Each interface between blocks consists of 10 connections to each side of the interface (20 connections in total). The sources and destinations are also chosen randomly, as well as the weights of the interface edges, which are chosen from W_e as well. By default, 50000 global edges are added to the graph. Their weights are also chosen randomly from W_e.

In order to add the global edges in a way which creates a power law in the distribution of their sources, the following algorithm is used.

Algorithm 2 Choosing sources and destinations of global edges to obtain a degree distribution which follows a power law without altering any other regularity properties

```
1: weightsStorage ← weightsStorage(v) = 0, ∀v ∈ G
2: destinations ← randomly generated multiset of nodes, whose cardinal equals the
   number of desired global edges
3: for B = B_1, B_2, ..., B_f, where B_1, B_2, ..., B_f are a random partition of all nodes in
   G and |B_1| = |B_2| = ··· = |B_f| do
4:     weightsStorage(v) ← weightsStorage(v) + 1, ∀v ∈ B
5:     for d ∈ destinations ∩ B do
6:         s ← randomly chosen node, probability of choosing s is proportional to weight
   of s in weightsStorage
7:         Add the edge s → d to G
8:         weightsStorage(v) ← weightsStorage(v) + 1, ∀v ∈ {s, d}
9:     end for
10: end for
```

Algorithm 2 follows the approach described in [3] to obtain graphs whose long-distance nodes are distributed following a power law, by considering the graph to be dynamic and using preferential attachment; nodes which are new tend to be influenced by nodes whose influence is already high. This makes certain nodes very attractive to be bribed when trying to change other nodes' opinions as efficiently as possible. In graphs whose long-distance edges are generated randomly, this does not take place. It is very important to choose the partition $B_1, B_2, \ldots, B_f$ randomly, since not choosing it randomly can create asymmetries in the graph and make small clusters of nodes too powerful. In our implementation we take $|B_i| = 100$.

Algorithm 2 needs an efficient implementation of `weightsStorage`. Implementing this data structure efficiently is not trivial, since range updates and range queries are needed to calculate and update cumulative probabilities in logarithmic time. This can be done using two Fenwick trees, using the strategy and code from [1]. More information about how this code was used in this work and its license can be found in Appendix I.

A relatively heavy reflexive edge is added to all nodes in order to ensure the model behaves like the real world (one's opinion is the most important factor which determines one's future opinion). The weight of this edge is chosen randomly and uniformly, from $\mathbb{N}_{25}$.

When the graph is initialised, the probability of a specific node agreeing is exactly $1/2$. Also, all nodes are assigned an integer cost (or weight) which is chosen randomly from $\{30, 31, \ldots, 50\}$. This cost has never appeared in previous sections and is related to the problem we want to solve. Also, when each node is generated, the probability that it becomes *locked* is $1/2$. This is also related to the problem we try to solve in this section and is related to being able to *bribe* a node.

The problem we need to solve, given the graph, is finding an optimum selection of nodes which are not locked and whose sum of weights does not exceed a maximum value (which is 50000 by default), such that if their opinion is *flipped* to 1, then the proportion of

nodes whose opinion is 1 is maximised after 20 simulation cycles. We essentially need to allocate resources; we have a maximum budget we can use to change people's opinion, and we strategically need to choose people whose opinion is important enough to change the opinion of connected nodes as much as possible to obtain a population which is as favourable as possible at a given moment in time. This problem can have practical applications, such as distributing electoral budget as efficiently as possible in order to obtain a good result in an election, and it is obvious that it is a particular case of problem (c) from Theorem 8.

The default constants we have chosen for determining the structure and the probabilities of this problem have been chosen carefully in order to ensure the properties of the graph are as similar as possible to those of real-world graphs. However, it is extremely difficult to choose them rigorously, as we could not find any research which is similar enough to this problem and has calculated these constants using scientifically obtained data from society. Therefore, it might be necessary to change some of the constants in order to obtain more realistic results if they are found to be inaccurate.

4.2 Basic strategies to solve the problems and obtained results

We have considered 10 different basic algorithms to solve this problem. Many of them are similar to each other, since the heuristics they consider in order to choose nodes to flip are very similar. We refer to these algorithms using the notation `Alg0` to `Alg9`. We now explain how the first five algorithms, `Alg0` to `Alg4`, work. The other five algorithms, `Alg5` to `Alg9`, are obtained by modifying how the first five algorithms apply their heuristics.

(a) `Alg0` ranks the nodes based on their cost. When this algorithm is applied, nodes whose value is 0 and which are not locked are chosen by increasing cost and their opinion is flipped to 1 until there is no remaining available cost left.

(b) `Alg1` considers a metric which intuitively represents how much a node influences other nodes. For each node, we define its first influence to be the sum of the homogenised weights of all outgoing edges of that node. By considering homogenised weights from the perspective of destination nodes, we accurately calculate what the real impact of a flip can be after one simulation step. `Alg1` merely chooses nodes whose first influence is higher, until there is no available cost left.

(c) `Alg2` considers the cost per unit of first influence of a node and tries to minimise it, choosing nodes such that their first influence divided by their cost is as high as possible.

(d) `Alg3` tries to maximise the second influence of chosen nodes. We define the second influence of a node v to be the total sum of the products of homogenised edge weights, for the edges of every directed path of the graph which starts at v and whose length is two. More formally, it is defined as

$$\text{secinf}(v_i) = \sum_{v_j \mid v_i \in \text{infl}(v_j)} \sum_{v_k \mid v_j \in \text{infl}(v_k)} |e_{ij}| \cdot |e_{jk}|,$$

where we assume that duplicate edges at generation time have been merged into a single edge. At first sight, it might seem like `Alg3` is better than `Alg1`, since it considers second-degree influence, which is not taken into account by `Alg1`. Nevertheless, nothing can be said about which heuristic is better at first sight, and it is also worth noting that computing the second influence of a node is computationally expensive, compared to computing its first influence, which is given by $\sum_{v_j \mid v_i \in \text{infl}(v_j)} |e_{ij}|$.

(e) `Alg4` tries to minimise the cost per unit of second influence; it is related to `Alg3` in the same way as `Alg2` is related to `Alg1`.

Algorithms `Alg5` to `Alg9` use exactly the same heuristics as algorithms `Alg0` to `Alg4`, respectively. However, the standard algorithms `Alg0` to `Alg4` are applied five times by spending a cost which is not higher than one fifth of the total available cost, plus previous unused cost from a previous iteration, if left. Before each iteration, a heuristic correction is applied to the graph; since we need to maximise the proportion of nodes which agree when $t = 20$, the graph obtained until the current iteration is simulated, and nodes which agree when $t = 20$ are *temporarily locked*. These nodes are not considered by the heuristic in the next iteration. During the next iterations, the labels of more nodes will be flipped to 1, thus uniformly increasing the values of labels, as defined in Definition 6. Corollary 1 implies the labels of temporarily locked nodes will still be 1 after the next iterations. Hence, assuming it is not necessary to flip temporarily locked nodes at the very beginning of the simulation in order to influence certain areas in advance, it seems more efficient not to flip nodes when $t = 0$ if we know they will agree when $t = 20$.

4.2.1 General comparison

All ten algorithms have been compared by running them 50 times. Apart from running them for graphs generated using the graph generation algorithm described above, two modifications have been introduced to better understand how they behave. The first modification we have considered is `NonReGE`, which generates graphs whose global edges are generated by using random sources and destinations. As we have explained before, this modification makes the graphs less realistic, since the distribution of global edge sources in real-world graphs follows a power law instead of a Poisson distribution, which is obtained if edges are generated randomly. The second modification is `Stretched`, which transforms the graph by making it 2500 blocks wide instead of 50 blocks wide and 50 blocks tall. This makes the graph much less compact, which makes it more difficult to influence many nodes from a single node without using global edges. In Table 1, we show how well all basic heuristics work on graphs generated by different types of algorithm by showing the final average proportion of favourable nodes. It can be seen that `Alg7` outperforms every other algorithm. This claim has been checked with a confidence of 99% using the hypothesis testing method designed by Sture Holm, described in [10] and implemented in [2]; additional instances had to be generated for comparing the performance of `Alg7` and `Alg9` and obtaining unequivocal conclusions. The ranking we obtain for all types of graphs is always the same:

`Alg7`, `Alg9`, `Alg6`, `Alg8`, `Alg5`, `Alg2`, `Alg4`, `Alg1`, `Alg3`, `Alg0`.

Modification	Alg0	Alg1	Alg2	Alg3	Alg4
Standard	0.7480152	0.7828424	0.7940712	0.7780384	0.7901952
NonReGE	0.7522288	0.7767368	0.7881176	0.7721000	0.7855128
Stretched	0.7042544	0.7374504	0.7479160	0.7356360	0.7447552
Both	0.7032392	0.7233448	0.7352504	0.7184056	0.7308472
Modification	Alg5	Alg6	Alg7	Alg8	Alg9
Standard	0.8768376	0.8994000	0.9118392	0.8956056	0.9078544
NonReGE	0.8793336	0.8964144	0.9079536	0.8923088	0.9042080
Stretched	0.8235320	0.8445024	0.8573536	0.8411672	0.8538840
Both	0.8225656	0.8327856	0.8479816	0.8266624	0.8420864

Table 1: Average proportion of favourable nodes obtained when applying all basic algorithms to different modifications of the standard problem. 50 different instances for each cell were used.

By remembering what each algorithm is based on, we conclude that first influence is a better criterion than second influence, although it intuitively seems better to use second influence, since it considers more information. It can also be seen that considering the cost per unit of influence is a better approach than considering the number of units of influence of a node.

Only considering the weight of nodes when selecting them is the worst possible strategy. However, what affects results the most is dividing heuristics in five steps and temporarily locking nodes (Alg5-Alg9) or not (Alg0-Alg4).

When the graph is stretched or global nodes are not realistic, the graph is less compact, so obtaining a good solution becomes more difficult, and the attained proportions of favourable nodes in Table 1 are lower. There are two exceptions, which are Alg0 and Alg5. These two algorithms only consider cost when choosing nodes, and their success rate increases when transitioning from standard graphs to NonReGE-graphs. This is because all other algorithms always select all or almost all of the nodes whose first influence is very high, while Alg0 and Alg5 do not. When global edges are distributed randomly, the number of highly influential nodes decreases significantly, making it less problematic when a certain heuristic does not take influence into account.

4.2.2 Benchmarks of the best basic strategy

In order to better understand how changing parameters affects the problem we are trying to solve, we analyse how the best basic strategy (Alg7) behaves when certain generation parameters of the opinion graphs change. All obtained results have been tested by using at least 50 different instances, and hypothesis testing methods from [10] (implemented in [2]) have been used to verify all statistically significant claims we make.

Changing the maximum weight of reflexive edges

When we defined the standard model we use for constructing graphs, we specified that the weights of global edges are taken randomly and uniformly from $\mathbb{N}_{25}$. More generally, they could be taken from $\mathbb{N}_x$, and this choice greatly affects the average stubbornness of nodes, making it much more difficult to change the value of other nodes by influencing

them from outside when x is large.

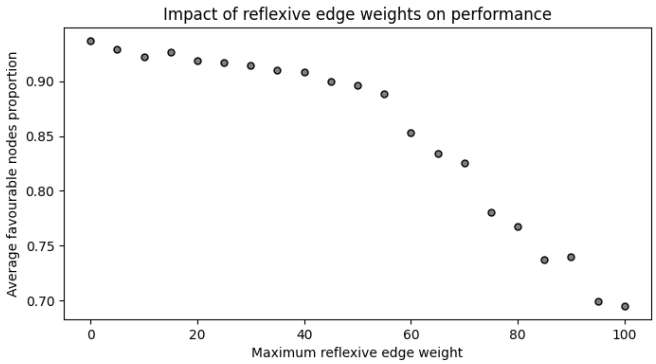

Figure 8: Changing the maximum possible weight of reflexive nodes affects the results obtained by the best basic strategy. 50 different graphs were generated and solved for each data point in order to obtain average results. Apart from the change in the reflexive nodes' maximum weight, no parameters are different from the standard ones. 21 different values ($\{0, 5, 10, \ldots, 100\}$) have been considered.

Figure 8 shows what the consequences of changing this parameter are; our intuition is confirmed. Although the general trend is clear, it can also be seen that changing this parameter affects the average favourable proportion of nodes more when it becomes large, while the effects are less noticeable when $x \leq 40$.

Changing the number of global edges

We can also modify the standard model by modifying the number of global edges without altering any of the other parameters. This should theoretically make the graph more compact and interconnected, reducing the average shortest path length and making it easier to spread opinion trends.

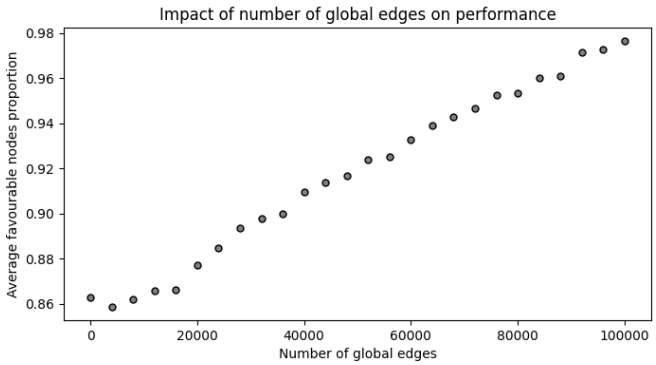

Figure 9: Changing the number of global edges affects `Alg7`'s performance. 50 different graphs were generated and solved for each data point in order to obtain average results. It can be seen that the trend is almost linear. The correlation between both variables equals 0.996. Nevertheless, this trend is not global, since the maximum attainable value in the y-axis is 1. Only the number of global edges is changed, all other parameters are the standard ones. 26 different numbers of global edges ($\{0, 4000, 8000, \ldots, 100000\}$) have been considered.

The trend in Figure 9 is confirmed and is essentially linear, but simulations beyond 100000 global edges show that it is not (the slope decreases beyond 100000 global edges), and the proportion of favourable nodes stabilises at around 0.996 when the number of global edges is greater than 140000.

Changing the available cost

The last parameter we have studied in order to understand how problem conditions alter results is the available budget. It is obvious that increasing the available budget necessarily increases the average proportion of favourable nodes, since previous assignments of labels can be reused and completed to solve graphs which have the same structure. However, nothing can be said about the trend type before simulating instances and obtaining conclusions.

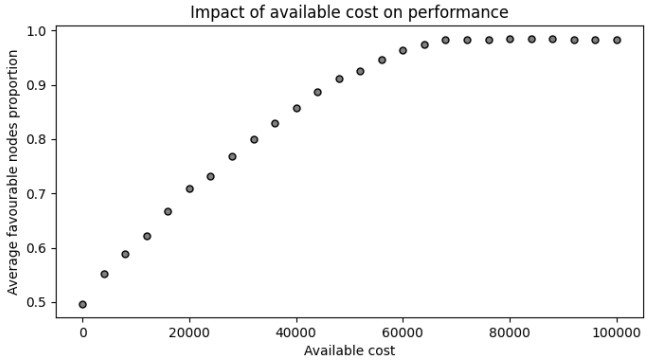

Figure 10: Proportion of favourable nodes compared to total available cost. 50 different graphs were generated and solved for each data point, and were not reused for other data points, in order to obtain average results. The trend seems to be exponential if the proportion of opposing nodes is considered instead (the coefficient of determination, R^2, equals 0.943). All used instances of graphs are generated using the standard methods described in 4.1. 26 different available budgets ($\{0, 4000, 8000, \ldots, 100000\}$) have been considered.

Even though there seems to be an exponential trend, it might not be a correct assumption. It is interesting to see that the success rate stabilises at a total available cost of around 70000. It seems that nothing else can be done beyond that point to convince more nodes, since the remaining nodes which disagree are almost always locked and are influenced by other nodes which disagree and are locked as well.

4.3 Using a genetic algorithm

A genetic algorithm has been designed and used to improve the results of `Alg7`. This algorithm is initialised by using the results from Algorithms `Alg5` to `Alg9`. Crossover and mutation are defined as follows.

For crossover, each sample is randomly ranked before crossover takes place; nodes which appear before in the sample after ranking are more important than nodes which appear at the end. Crossover consists of selecting the most important nodes from each sample (iteratively selecting the most important node left for both samples at the same time, and

of course not selecting a node again if there are repetitions), and stop selecting when it is no longer possible because no more budget is available.

For mutation, the graph is simulated for 20 cycles and a *bribed* node is unmarked if it is detected that the sum of the weights of incoming edges (from nodes whose label value is 1) is high (> 0.6), since it is assumed that the label of that node would become 1 anyway, due to influence from neighbours. Then, new nodes are added randomly by using the weight which has been given back by unmarked nodes. No more than 10 nodes are unmarked per iteration in order not to substantially alter a solution when mutating.

Every iteration starts by considering five instances. Crossovers for each possible combination are done, and each instance is also randomly mutated six times. The best five instances of all of these modifications, including the initial ones, are used in the next iteration.

In order to test this genetic algorithm, we have chosen the available budgets $c_1 = 20000$ and $c_2 = 40000$, and run this genetic algorithm using 50 graphs for each of these budgets. The obtained proportions of favourable nodes can be found in Table 2.

Cost used	`Alg7`	10 iterations	20 iterations	30 iterations	40 iterations
c_1	0.7045312	0.7063823	0.7138368	0.7194051	0.7231830
c_2	0.8592466	0.8647835	0.8674822	0.8714732	0.8751670

Table 2: Average proportion of favourable nodes obtained when applying the genetic algorithm described in this section to the standard problem, which has been modified by changing the available cost to c_1 and c_2. 50 different instances for each cell were used.

Although there seems to be some improvement and it has been checked using [2] that the improvement is statistically significant, this algorithm takes a very long time to run, and the improvement is not very significant. It might be possible to obtain similar or better results by considering additional heuristics or a more efficient genetic algorithm.

However, it is true that the genetic algorithm we have considered seems to be better than the best basic strategy (`Alg7`), and it might also be the case that the improvement is not very significant because the proportion of favourable nodes cannot be increased much more for these instances using any method.

5 Conclusion

This work has allowed us to obtain important complexity properties about opinion graphs; they are essentially as expressive as Turing machines. Therefore, fully solving optimisation problems in order to determine a good strategy for influencing a population's opinion is, in most cases, not feasible. All relevant problems for carrying out this task are at least **NP**-hard, and the most complex ones are even **PSPACE**-hard.

In spite of this hurdle, we have been able to propose relatively simple strategies to solve one of the **NP**-complete problems successfully using opinion graphs whose properties make them similar to real-world examples. As we have already mentioned, the graphs we have considered when proving complexity properties are very different from real-world examples. It appears that real-world graphs are much more regular, and very simple heuristics yield good results. Of course, the performance of greedy strategies can be improved by temporarily locking promising nodes in intermediate steps and using genetic algorithms, which improve the proportion of favourable nodes even more.

Even though the results we have obtained are satisfying and provide a lot of insight, there remains much to be done. On the one hand, more types of graphs could have been analysed. Besides, the types of graphs we have considered could have been analysed in greater depth. On the other hand, not many different heuristics have been used to analyse the problem we wanted to optimise, and more complex genetic algorithms could have been used to obtain better results.

The practical problem we have analysed is relatively general, but some variations of the problem could be considered in further research. For instance, the weight distribution could be modified. In the real world, people who are influential are usually more expensive to bribe than people whose influence is lower. Taking this correlation into account would make the graph even more realistic.

The theoretical results we have obtained are very complete and have made us understand the type of problems which arise when studying opinion graphs. Nevertheless, there are aspects which have not been covered, some of which are average case complexity and approximability classes. More research could be done in order to provide a more accurate classification of the theoretical problems we have discussed in this work.

There is one more aspect which could be studied to generalise the obtained results; the transition function used to update node labels in opinion graphs in general could be made probabilistic and possibly non-linear to compare how nodes behave depending on the considered function. A lot of recent papers have focused on non-linear voter models, and applying these models to our problem would make it possible to obtain very general conclusions which could even be applied outside the scope of this work.

References

[1] Binary Indexed Tree : Range Update and Range Queries. `https://www.geeksforgeeks.org/binary-indexed-tree-range-update-range-queries/`. [Accessed 13-March-2023].

[2] Non-parametric multiple groups one vs all. `https://tec.citius.usc.es/stac/ranking.html`. [Accessed 15-April-2023].

[3] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.

[4] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[5] Claudio Castellano, Miguel A. Muñoz, and Romualdo Pastor-Satorras. Nonlinear q-voter model. *Physical Review E*, 80(4), October 2009.

[6] Raymond Chiong and Michael Kirley. Effects of iterated interactions in multi-player spatial evolutionary games. *IEEE Transactions on Evolutionary Computation*, 16(4):537–555, August 2012.

[7] Kari Eloranta. *Voter dynamics in deterministic cellular automata*, volume 8, pages 51–58. De Gruyter, 1996.

[8] Juan Fernández-Gracia, Krzysztof Suchecki, José J. Ramasco, Maxi San Miguel, and Víctor M. Eguíluz. Is the voter model a model for voters? *Physical Review Letters*, 112(15), April 2014.

[9] Roger Guimerà, Leon Danon, Albert Díaz-Guilera, Francesc Giralt, and Alex Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6), December 2003.

[10] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[11] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Pearson, Upper Saddle River, NJ, 3 edition, June 2006.

[12] Hui-Jia Li, Lin Wang, Yan Zhang, and Matjaž Perc. Optimization of identifiability for efficient community detection. *New Journal of Physics*, 22(6):063035, June 2020.

[13] Thomas M. Liggett. *Stochastic Interacting Systems: Contact, Voter and Exclusion Processes*. Springer Berlin Heidelberg, 1999.

[14] Naoki Masuda, Nathanael Gibert, and Sidney Redner. Heterogeneous voter models. *Physical Review E*, 82(1), July 2010.

[15] Matjaž Perc, Jesús Gómez-Gardeñes, Attila Szolnoki, Luis M. Floría, and Yamir Moreno. Evolutionary dynamics of group interactions on structured populations: a review. *Journal of The Royal Society Interface*, 10(80):20120997, March 2013.

[16] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[17] Vishal Sood, Tibor Antal, and Sidney Redner. Voter models on heterogeneous networks. *Physical Review E*, 77(4), April 2008.

[18] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

Appendix I: More information about used software

In order to simulate the problem and the performance of the algorithms we have considered, as well as for writing this document, two computer programs have been developed and used.

One of them generates random graphs with given parameters and simulates their performance, just like it has been described in this document. There has not been enough time to make this program user-friendly, so it has been used by designing tests as part of the program and running them separately.

The other program has been designed to help the author to create the figures of this document and the final presentation. It is impossible to find any other tools which can generate figures like the ones of this document and successfully integrate them in a LaTeX document, so it seemed necessary to develop such a program. The program iteratively visits folders of the location from which it is run, and looks for TikZ code and scripts whose syntax has been created by the author. This program then transforms TikZ code as indicated by the scripts to create new TikZ code which integrates transformed copies of the original TikZ code. Using these script files might not be very intuitive, but it was found to be very practical, since one of the original TikZ files can be modified and the transformed TikZ code can be instantly regenerated.

Two classes of the code which generates random opinion graphs are refactored code obtained from `https://www.geeksforgeeks.org/binary-indexed-tree-range-update-range-querie` to implement Algorithm 2 efficiently. The license of this code is `CC-BY-SA`, whose full text can be read at `https://creativecommons.org/licenses/by-sa/2.0/`. This license has been reused for that part of the code and it is indicated in code files.

All of the code which has been written and used for this work has been submitted along with this document and will be made available at `https://github.com/Martin-ga` when possible.

Appendix II: Raw data examples

Large samples of raw data have been submitted along with this document, and even larger samples of raw data will later be published at `https://github.com/Martin-ga` when deemed appropriate. Here we give a small example, which corresponds to proportions of favourable nodes obtained for generating the fifth data row of Table 1. Each entry is calculated using the proportions obtained from 50 instances.

Alg5	Alg6	Alg7	Alg8	Alg9
0.87236	0.87672	0.90224	0.89008	0.90672
0.85836	0.86488	0.88288	0.85880	0.87560
0.87752	0.90848	0.91012	0.90732	0.91392
0.87684	0.91644	0.92016	0.91260	0.91960
0.87560	0.89868	0.90756	0.89604	0.90524
0.86604	0.86564	0.89108	0.87944	0.89136
0.89832	0.92832	0.93644	0.93716	0.94596
0.89148	0.89808	0.91628	0.89432	0.90344
0.89500	0.90968	0.92036	0.90508	0.91544
0.86404	0.90040	0.91092	0.88528	0.89972
0.90084	0.90472	0.92080	0.90104	0.91496
0.87920	0.88972	0.91000	0.89124	0.90856
0.87080	0.89908	0.90672	0.88912	0.89796
0.87104	0.90492	0.91140	0.89812	0.90992
0.87608	0.89860	0.91252	0.89028	0.90496
0.87884	0.88804	0.90436	0.88648	0.90284
0.88812	0.89192	0.90560	0.89312	0.90492
0.85032	0.86084	0.88200	0.85272	0.88176
0.87400	0.89500	0.91100	0.88876	0.89672
0.89408	0.93196	0.93572	0.92984	0.93256
0.87404	0.91648	0.91988	0.90088	0.91588
0.85096	0.86080	0.88040	0.85340	0.87020
0.86152	0.88620	0.90412	0.89280	0.89920
0.87464	0.91716	0.92304	0.91232	0.91764
0.90408	0.92512	0.92696	0.90288	0.92612
0.87676	0.90376	0.91744	0.88552	0.90788
0.85656	0.88808	0.89992	0.88192	0.88868
0.86632	0.87516	0.89652	0.85888	0.87940
0.87708	0.88844	0.90636	0.89140	0.89508
0.85544	0.89924	0.90964	0.89432	0.90912
0.89256	0.90904	0.93112	0.92272	0.92960
0.89480	0.91944	0.92896	0.92184	0.92820
0.90072	0.93552	0.93480	0.92920	0.92972
0.88712	0.90660	0.92520	0.89572	0.91592
0.86960	0.90424	0.90976	0.90164	0.91360
0.87304	0.88144	0.89064	0.87468	0.89016
0.86236	0.87400	0.89396	0.87000	0.88816

0.88404	0.91300	0.92468	0.91464	0.92060
0.89676	0.91836	0.93636	0.92120	0.93420
0.86796	0.90228	0.91576	0.89788	0.90452
0.88596	0.94116	0.93664	0.93068	0.93936
0.83724	0.85844	0.87712	0.85460	0.87584
0.89016	0.92668	0.93528	0.92684	0.93084
0.87508	0.89780	0.90896	0.90412	0.91328
0.90988	0.92224	0.93380	0.90308	0.91824
0.88132	0.91360	0.92188	0.92392	0.92752
0.88300	0.89252	0.91012	0.87832	0.90564
0.86080	0.88308	0.89240	0.87196	0.88372
0.87432	0.90016	0.91456	0.89748	0.91196
0.85888	0.87784	0.88752	0.87860	0.89028