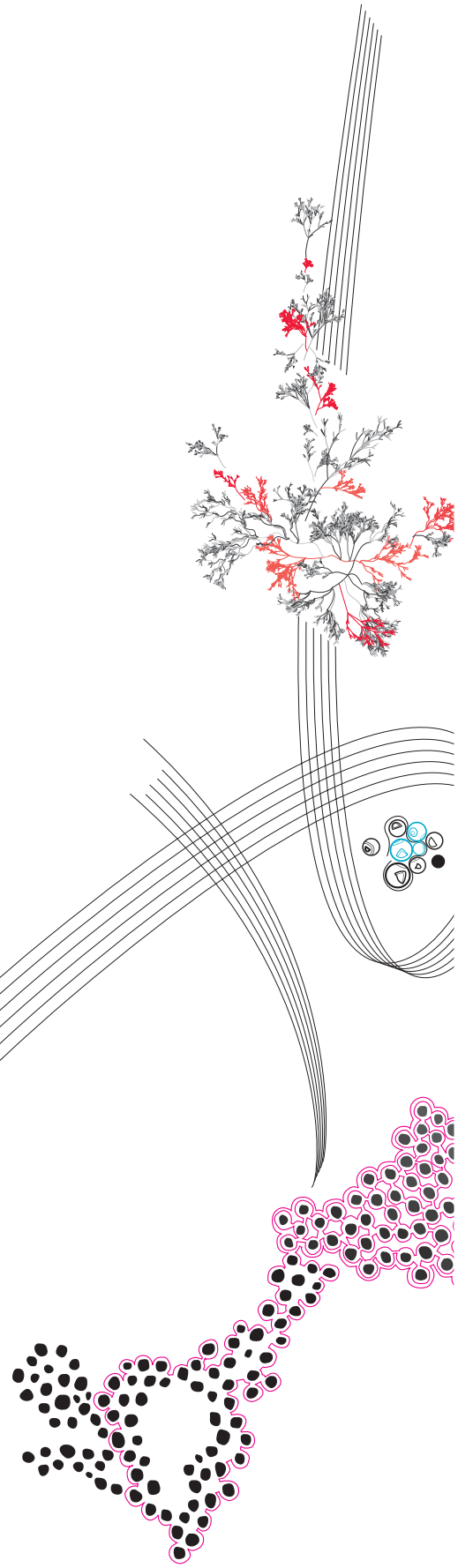

BSc Thesis Applied Mathematics

Controllability of Laplacian Leader Follower Dynamics: Algorithmic Approach

Mikuláš Vanoušek

Supervisor: M. A. Lopuhaä-Zwakenberg, F. L. Schwenninger, A. A. Wierzba

March, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

First, I would like to thank my supervisors Milan A. Lopuhaä-Zwakenberg, Felix L. Schwenninger, and Alexander A. Wierzba. They all fulfilled their roles beyond my expectations, and I am grateful for the time they devoted. Their flexibility allowed for the last-minute change in study plan, which had me write my thesis two quartiles earlier than originally planned. Especially Alexander, for whom supervising was a first, put himself behind this thesis as if it was his most important project these days.

I would like to thank all the teachers at University of Twente who helped prepare me for this project. My gratitude extends to the programme management of both Applied Mathematics and Technical Computer Science, as well as other non-pedagogical staff – notably Lilian Spijker and Judith Timmer, who played a crucial role in my hasty change in study plan.

I would like to thank my classmates, most notably my double degree cohort. Their companionship helped me push through the miscellaneous challenges of the study. The boards of Abacus and Inter-Actief organized social events that brought me closer to my peers. My amazing friends in Team of Builders helped me through all personal struggles and advised me on how to bring this big project to a successful end during our weekly meetings.

Last, but not least – in fact the most – I would like to thank my parents. It was their generous support, both emotional and financial, which allowed me to pursue the double degree. I wouldn't be in a position to write this thesis today if I had to keep a job next to my studies.

Writing this thesis was a valuable life lesson, teaching me not just research and academic skills, but also giving me confidence to tackle large projects, which do not have a rigid and detailed specification handed to me by a superior.

Controllability of Laplacian Leader Follower Dynamics: Algorithmic Approach

Mikuláš Vanoušek *

March, 2024

Abstract

Laplacian leader follower dynamics is a consensus algorithm used for distributed coordination of networked systems, the topology of which is given in the form of a graph. We provide an algorithm to classify graphs into their controllability classes (essentially controllable, conditionally controllable, and completely uncontrollable) which is asymptotically faster than one designed for an arbitrary state space linear system. We use this algorithm to answer questions about resiliency of this dynamic to random changes in topology, and to investigate the probability a topology is essentially controllable.

Keywords: Laplacian leader follower dynamics, controllability, resilience

*Email: m.vanousek@student.utwente.nl

Contents

1 Introduction

1.1 Motivation

Suppose a network of autonomous agents — for example a swarm of quadcopter drones, where each is able to communicate with the adjacent drones – which are all subject to some internal time-continuous dynamics. Suppose further we desire to control this network using an input signal. In practice, if the network is large, it is unfeasible to communicate with each of these agents separately. It would be highly practical if one could control them, while only communicating with a small group of them, the so-called leaders.

One potential form of dynamics for such a system is the **Laplacian leader-follower dynamic (LLFD)**. LLFD is a consensus algorithm, that is, it causes all the agents to converge to a common opinion. The controllability of the resulting dynamic system will in general depend on both the graph structure and our choice of leaders.

In practice, it is possible for the graph structure of the network to unexpectedly change. Resilience of the system to these random changes is essential as we do not want to lose control, just because two agents lose connection, or because one agent malfunctions. It is thus natural to ask how the controllability properties change under random changes to the graph.

1.2 Research Questions

Suppose we are designing a networked system with a given number of agents, that will behave according to some specified behaviour – the Laplacian leader-follower dynamic. The system may be controllable, depending on our selection of leaders. If the system is controllable for every (reasonable) choice of leaders, we call it essentially controllable, if for some choices of leaders, we call it conditionally controllable and if for no choice of leaders, we call it completely uncontrollable.

In practice, it is possible for the topology of the network (represented by a graph) to change unpredictably, due to hardware failure, wireless communication interference, etc. **In what topology shall we connect the agents, so that the system is resilient to these random changes?** In particular, we would like the probability that the system is essentially controllable after a random change to be high. We will consider two types of random change:

- A random edge is lost (two agents losing connection).
- A random vertex and all edges adjacent to it are lost (one agent malfunctions).

One may suspect the complete graph to be an answer to the posed question. But by the nature of Laplacian leader-follower dynamics, we need certain asymmetry or irregularity in the topology to achieve essential controllability. The complete graph is in fact completely uncontrollable.

In order to answer the research question about resilience to random changes, we will first focus on the following preliminary research questions. In [2], graphs are classified into essentially controllable, conditionally controllable, and completely uncontrollable. Suppose a networked system with a known graph structure follows the Laplacian leader-follower dynamic. We will attempt to devise and implement an algorithm to decide whether the system is essentially controllable, conditionally controllable, or completely uncontrollable, and we will analyse the complexity of this

an algorithm. We will then use this algorithm to investigate the main research question on small graphs.

Essential controllability and asymmetry of the underlying graph are closely connected. We will thus examine whether, as we increase the number of vertices, the ratio of the number of essentially controllable graphs to the number of asymmetric graphs approaches 1.

1.3 Literature Review

[15] is first to consider the controllability of Laplacian leader-follower dynamics, its theoretical findings are advanced in [2], which also introduces the definition of the controllability classes. It provides two propositions we used in the construction of the algorithm. The authors also provide a table evaluating the number of graphs in each of the controllability class.

[5] discusses the application of LLFD in construction of aerial swarms. [13] highlights the usefulness of LLFD for the distributed coordination of networked agents in general. This contributes to the motivation of our research.

[16] asks questions about resilience of graphs under random changes, but for a leader-follower dynamic that is different from LLFD. While their approach can not be easily adapted to help in our research, others investigating this question, be it in a different context, indicates the question is relevant and its investigation worthwhile.

The experimental part of this thesis is powered by the *Julia* programming language [3] and the *Graphs.jl* library [7]. The list of all unique connected graphs with a given number of vertices was generated using the program *Nauty*, see [10].

We used insights from [9] to determine the number of random graphs we need to investigate for a given precision and confidence.

2 Graph Theory & State Space Systems

2.1 Graph Theory

This subsection is adapted from *A. Graph Theory* in *Preliminaries* of [2]. We omit citation marks to improve readability, as the source text consists of common mathematical facts.

By a **labelled graph**, we mean a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of a finite **vertex set** $\mathcal{V}$ and an **edge set** $\mathcal{E} \subseteq \{\{v, w\} \mid v, w \in \mathcal{V}, v \neq w\}$.[1] The **order** of the graph $\mathcal{G}$ is the cardinality of its vertex set $\mathcal{V}$ and we will denote it with n. The **neighbours** of $v \in \mathcal{V}$ is the vertex set, $\mathcal{N}_v := \{w \in \mathcal{V} \mid \{v, w\} \in \mathcal{E}\}$ and the **degree** of v is $d_v := |\mathcal{N}_v|$.

A **path** in $\mathcal{G}$ of length k is a subgraph of $\mathcal{G}$ consisting of vertices $\{v_0, v_1, \ldots, v_k\} \subseteq \mathcal{V}$ and edges $\{\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}\} \subseteq \mathcal{E}$, where all the v_i are distinct. For such a path, v_0 and v_k are called the **terminal vertices**. We call two vertices u, v **connected** iff there exists a path with u, v as terminal vertices, and we say a graph $\mathcal{G}$ is **connected** if there is a path between any pair of vertices. Suppose $R := \{(u, v) \mid u, v \in \mathcal{V},\ u, v \text{ are connected}\}$. Then R is an equivalence

[1] In the context of LLFD, we consider only simple graphs – undirected, with no loops or multiple edges. Self loops would cancel algebraically by the definition of LLFD. Multiple edges could be entertained in future work.

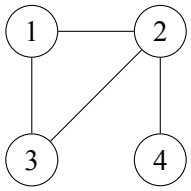

Figure 1: An example of a graph.

relation on $\mathcal{V}$ and the partitions of $\mathcal{V}$ induced by it are called **connected components** of $\mathcal{G}$. We call an edge $\{v, w\}$ a **bridge** of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ iff $\mathcal{G}' := (\mathcal{V}, \mathcal{E} \setminus \{v, w\})$ has more connected components than $\mathcal{G}$. For example, the graph in Figure 1 is connected and in that graph, $\{2, 4\}$ is a bridge.

Henceforth, when not explicitly stated otherwise, we fix an ordering on the vertex set $\mathcal{V}$ and thus, without loss of generality, we take $\mathcal{V} = \{1, \ldots, n\}$, where n is the order of $\mathcal{G}$. The **adjacency matrix** of $\mathcal{G}$ is the $n \times n$ matrix A defined as $A_{ij} = 1$ if $\{i, j\} \in \mathcal{E}$ and $A_{ij} = 0$ otherwise, where A_{ij} denotes the entry of A in the ith row and jth column. The adjacency matrix of the example graph in Figure 1 is

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

We denote by D the **degree matrix** of $\mathcal{G}$, i.e., the diagonal matrix whose ith diagonal entry is d_i. The **Laplacian matrix** of $\mathcal{G}$ is given by

$$L = D - A$$

The Laplacian matrix of the example graph is

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

The Laplacian matrix L is symmetric and positive semidefinite, and thus the eigenvalues of L can be ordered $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. The ones vector $\mathbf{1}_n := \begin{bmatrix} 1 & 1 & \ldots & 1 \end{bmatrix}^\intercal$ is an eigenvector of L with eigenvalue $\lambda_1 = 0$. The algebraic multiplicity of the 0 eigenvalue is the number of connected components of $\mathcal{G}$, in particular, $\mathcal{G}$ is connected iff $\lambda_1 = 0$ is a simple eigenvalue of L. For our purposes, by the eigenvalues (eigenvectors) of a graph $\mathcal{G}$, we mean the eigenvalues (eigenvectors) of its Laplacian matrix L.

A mapping $\varphi : \mathcal{V} \to \mathcal{V}$ is an **automorphism** of $\mathcal{G}$ iff it is a bijection and $\{i, j\} \in \mathcal{E}$ implies that $\{\varphi(i), \varphi(j)\} \in \mathcal{E}$. The identity mapping is an automorphism of every graph, and we call it the **trivial automorphism**. For example, the graph in Figure 1 has exactly one non-trivial automorphism: $\varphi = \{1 \to 3,\ 2 \to 2,\ 3 \to 1,\ 4 \to 4\}$. We call a graph **asymmetric** iff the identity automorphism is its only automorphism. Let $\sim$ be a relation, where $\mathcal{G}_1 \sim \mathcal{G}_2$ iff there exists an automorphism that maps $\mathcal{G}_1$ to $\mathcal{G}_2$. Then $\sim$ is an equivalence relation, and by an **unlabelled graph** we understand an equivalence class induced by $\sim$.

2.2 Systems & Control

The following facts come from the University of Twente bachelor course reader Linear Systems Theory by Gjerrit Meinsma, however can be also found in [12].

Definition 1. *By a **linear state space system**, we understand the pair of matrices* $(A, B) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n_u}$ *and the resulting system of differential equations*

$$\begin{aligned} \dot{x} &= Ax + Bu \\ x(0) &= x_0 \end{aligned} \tag{1}$$

where $u : [0, \infty) \to \mathbb{R}^{n_u}$ *is the input,* $x : [0, \infty) \to \mathbb{R}^n$ *is the state, and* $x_0 \in \mathbb{R}^n$ *is the initial state.*

Usually, the definition of a linear state space system includes the output $y : [0, \infty) \to \mathbb{R}^{n_y}$, but we decided to omit it as it is not necessary for our work. Many real world phenomena can be modelled with a state space linear system, for example mass-spring-damper system, RC circuits or population dynamics.

Later, we investigate the autonomous behaviour of a linear state space system – that is, we let $u = 0$.

Proposition 1. *If* $u = 0$*, the solution to a linear state space system* (A, B) *is given by* $x(t) = e^{At}x_0$.

Often, we are interested in whether we can steer the system to a particular state x_1 by applying an appropriate input. This gives rise to the notion of controllability.

Definition 2. *A system* (A, B) *is **controllable** iff for every pair of states* $x_0, x_1 \in \mathbb{R}^n$*, and* $x(0) = x_0$*, there is a* $t_1 \geq 0$ *and an input* $u : [0, t_1] \to \mathbb{R}^{n_u}$ *such that* $x(t_1) = x_1$.

It can be shown that controllability of a system (A, B) can be characterized in terms of a matrix rank condition.

Proposition 2. *A linear state space system* (A, B) *is controllable iff the controllability matrix*

$$\mathcal{C} := \begin{bmatrix} B \\ AB \\ \vdots \\ A^{n-1}B \end{bmatrix}$$

has full column rank.

2.3 Laplacian Leader Follower Dynamics

Suppose a network of agents – each agent has a communication link to some others. Such network can be represented with a labelled graph. An example of such a network can be a swarm of quadcopter drones. Suppose further that each agent in the network has an opinion. Generally, by an opinion of an agent j, denoted x_j, we understand a vector in $\mathbb{R}^k$. In the swarm of drones example, it may be the direction and the speed in which a swarm of quadcopters is moving, or its location. We can, however, analyse every component of the opinion independently, hence for the sake of simplicity we will in the rest of this work assume $x_j \in \mathbb{R}$. One can see an example of such a network of agents in Figure 3.

In the absence of control, it is (in many contexts) desirable for the agents to arrive at a compromise – have their opinions converge to the same value. For that, we can use a so-called consensus mechanism. One way of achieving this (in a connected graph) is Laplacian Leader Follower Dynamics (LLFD).

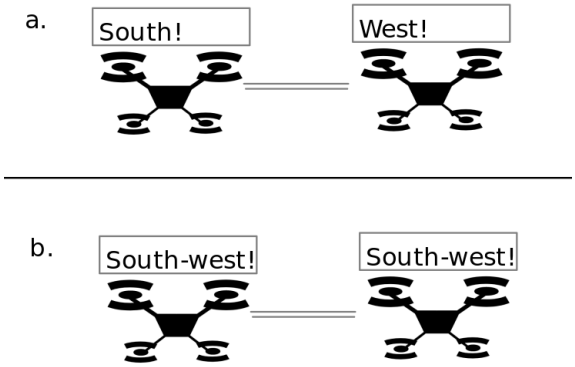

Figure 2: A network of two drones reaching a compromise on their direction of movement.

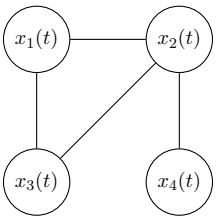

Figure 3: An example of a network of agents with opinion $x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_4(t) \end{bmatrix}$.

Each agent j modifies their opinion x_j to closer align it with the opinion of its neighbours in the following manner:

$$\dot{x}_j = \sum_{i \in \mathcal{N}_j} (x_i - x_j) \tag{2}$$

The component-wise Equation 2 is equivalent to the vector form equation

$$\dot{x} = [A(\mathcal{G}) - D(\mathcal{G})]x = -L(\mathcal{G})x, \tag{3}$$

where $x \in \mathbb{R}^n$ is the opinion of the network and $A(\mathcal{G})$, $D(\mathcal{G})$, $L(\mathcal{G})$ are the adjacency, degree, and Laplacian matrix respectively, as defined in Subsection 2.1.

While convergence to a compromise is a good behaviour in the absence of external interference, it is (in many contexts) practical to be able to have some influence over the opinions in the network. The optimal scenario is if we arrive at a controllable system, where we have the freedom to impose any particular vector of opinions x. We are attempting to control the network of agents with a one dimensional input u. We choose a set of leaders and influencing all of them in the same way. Our leader selection can be represented with the binary control vector $b \in \{0,1\}^n$[2]. Then our system can be represented with

$$\dot{x} = -L(\mathcal{G})x + bu.$$

Definition 3. *Let $n \in \mathbb{N}$ and $\mathcal{G}$ be a graph with n vertices. Then by $\mathcal{G}$**'s LLFD** we understand the state space system described by*

$$\dot{x} = -L(\mathcal{G})x + bu \tag{4}$$

with $b \in \{0,1\}^n$ representing our choice of leaders.

What makes LLFD an interesting consensus algorithm is its simplicity. The edges have no weight and the leaders are all treated as one by the controller.

2.3.1 Autonomous behaviour of Laplacian Leader Follower Dynamics

We first examine LLFD without external interference – we assume $u = 0$.

Lemma 1. *If $u = 0$, then the sum of opinions of all the agents in the network is constant.*

Proof. We have

$$\frac{d}{dt}\sum_{j=1}^{n} x_j = \sum_{j=1}^{n} \dot{x}_j = \sum_{j=1}^{n}\sum_{i\in\mathcal{N}_j} (x_i - x_j)$$

But if we sum over every vertex and then all its neighbours, we consider each edge $\{k,l\}$ in the graph exactly twice – when $j = k, i = l$ and when $j = l, i = k$. These two visits to a particular edge yield $x_l - x_k$ and $x_k - x_l$ respectively, which of course sum to zero. Because this is true for every edge, we get

$$\frac{d}{dt}\sum_{j=1}^{n} x_j = 0. \quad \square$$

Proposition 3. *If $\mathcal{G}$ is connected and $u = 0$, then the individual opinions in the network converge to the average of initial state x_0.*

Proof. In other words, we are looking to prove that if $x : [0,\infty) \to \mathbb{R}$ is a solution of $\dot{x} = -L(\mathcal{G})x$ and $x(0) = x_0$, then

$$\lim_{t\to\infty} x(t) = \frac{\sum_j (x_0)_j}{n}\begin{bmatrix}1\\1\\\vdots\\1\end{bmatrix}$$

Let L be the Laplacian matrix of $\mathcal{G}$. Because L is symmetric and positive semidefinite, we know that its eigenvalues are real and non-negative, and that L is diagonalizable. Because L is the Laplacian matrix, it has $\begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix}^\intercal$ as an eigenvector with $\lambda_1 = 0$ and because $\mathcal{G}$ is connected, $\lambda_1 = 0$ is a simple eigenvalue. So let the eigenvalues of L be $\lambda_1 = 0 < \lambda_2 \leq ... \leq \lambda_n$.

Let T be the matrix whose ith column is the eigenvector of L that corresponds to eigenvalue λ_i. Then

$$-L = -\left(T \begin{bmatrix}\lambda_1 & 0 & \dots & 0\\ 0 & \lambda_2 & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & \lambda_n\end{bmatrix} T^{-1}\right) = T \begin{bmatrix}-\lambda_1 & 0 & \dots & 0\\ 0 & -\lambda_2 & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & -\lambda_n\end{bmatrix} T^{-1}.$$

And by Proposition 1

$$x(t) = e^{-Lt}x_0 = T \begin{bmatrix}e^{-\lambda_1 t} & 0 & \dots & 0\\ 0 & e^{-\lambda_2 t} & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & e^{-\lambda_n t}\end{bmatrix} T^{-1}x_0.$$

We are looking for $\lim_{t\to\infty} x(t)$. Because $\lambda_1 = 0 < \lambda_2 \leq ... \leq \lambda_n$ we get

$$\lim_{t\to\infty} x(t) = T\left(\lim_{t\to\infty} \begin{bmatrix}e^{-\lambda_1 t} & 0 & \dots & 0\\ 0 & e^{-\lambda_2 t} & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & e^{-\lambda_n t}\end{bmatrix}\right) T^{-1}x_0 = T \begin{bmatrix}1 & 0 & \dots & 0\\ 0 & 0 & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & 0\end{bmatrix} T^{-1}x_0.$$

And because the first eigenvector of L is $\begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix}^\intercal$,

$$T \begin{bmatrix}1 & 0 & \dots & 0\\ 0 & 0 & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & 0\end{bmatrix} T^{-1}x_0 = \begin{bmatrix}1 & 0 & \dots & 0\\ 1 & 0 & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 0 & \dots & 0\end{bmatrix} T^{-1}x_0 = z_1 \begin{bmatrix}1\\ 1\\ \vdots\\ 1\end{bmatrix},$$

where z_1 is the first component of $T^{-1}x_0$ (a scalar).

But Lemma 1 tells us that $\sum_j x(t)_j$ is constant. Hence, also

$$\sum_j (x_0)_j = \lim_{t\to\infty} \sum_j x(t)_j = \sum_j \left(\lim_{t\to\infty} x(t)\right)_j = \sum_j \left(z_1 \begin{bmatrix}1\\ 1\\ \vdots\\ 1\end{bmatrix}\right)_j = nz_1.$$

Hence $z_1 = \sum_j (x_0)_j / n$ and

$$\lim_{t\to\infty} x(t) = \frac{\sum_j (x_0)_j}{n} \begin{bmatrix}1\\ 1\\ \vdots\\ 1\end{bmatrix}. \quad \square$$

2.3.2 Controllability of Laplacian Leader Follower Dynamics

In this section, we partition graphs into three classes based on the controllability properties of the associated LLFD. While in Definition 2 we define controllability for a system (A, B), or (L, b) in case of LLFD, here controllability is examined as a property across all possible binary control vectors b and hence is a property of the Laplacian matrix – in other words it is a property of the graph.

Definition 4. *(Definition 3.1 in [2]) Let $\mathcal{G}$ be a connected graph with Laplacian matrix L and let $\mathcal{B} = \{0, 1\}^n$. Then $\mathcal{G}$ is called:*

1. ***essentially controllable*** *on $\mathcal{B}$ iff (L, b) is controllable for every*[2] *$b \in \mathcal{B} \setminus \ker(L)$;*
2. ***completely uncontrollable*** *on $\mathcal{B}$ iff (L, b) is uncontrollable for every $\mathbf{b} \in \mathcal{B}$;*
3. ***conditionally controllable*** *on $\mathcal{B}$, iff it is neither essentially controllable nor completely uncontrollable on $\mathcal{B}$.*

One might wonder why we chose those three classes – why is this the right way to divide the graphs. The essential controllability and complete uncontrollability are extremes of sorts. For example, for $n = 9$ there are $2^9 = 512$ control vectors, and we require that all but two lead to a controllable system to classify it as essentially controllable, or that none of them lead to a controllable system, to classify it as completely uncontrollable. Yet, as one can see in Table 1, those two scenarios combined are almost as likely as the system being conditionally controllable.

Note that definition 4 is restricted to connected graphs. This is because if a graph is disconnected, then (L, b) is uncontrollable for every $b \in \mathcal{B}$.

3 Graph Classification Algorithm

In this section, we devise and analyse two algorithms to determine which controllability class a given graph belongs to.

3.1 State System Algorithm

For any particular system with n agents and a selection of leaders (represented with the pair (L, b)), we can determine if it is controllable (as defined in Definition 2) by constructing the controllability matrix and checking its rank (see Proposition 2).

We shall analyse the worst-case asymptotic time complexity of this approach. The complexity of matrix multiplication is $\Theta(n^\omega)$. The value of ω depends on the implementation of the matrix multiplication algorithm, and the current best techniques put it around 2.5 [6].[3] The precise value is not crucial to our analysis. In order to construct the controllability matrix, we need to perform n matrix multiplications, which puts the complexity of this step at $\Theta(n^{\omega+1})$. The complexity of checking a rank is lower than that,[4] hence, the total time complexity of checking controllability of (L, b) is dominated by the construction of the controllability matrix and stands at $\Theta(n^{\omega+1})$.

[2] Of course because $\mathcal{G}$ is connected, $\ker(L) = span(\begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix})^\intercal$

[3] Interestingly, in [6] it is shown that the asymptotic complexity of any matrix multiplication can be improved.

[4] A (non-optimal) way to check rank is Gaussian elimination, which has complexity is $\Theta(n^3)$[4]

However, to determine in which controllability class a graph is located, we need to know if the system is controllable for any binary control vectors, of which there are 2^n. The following proposition allows us to cut the number of control vectors for which we check controllability in half, but unfortunately this does not change the asymptotic complexity.

Proposition 4. *(Proposition 4.1 in [2]) Controllability and Binary Complements: Let $n \geq 2$ and consider the controlled Laplacian dynamics (1) with $b \in \{0,1\}^n$. Then the pair (L, b) is controllable if and only if the pair $(L, \bar{b})$ is controllable.*

This means the complexity of classifying a graph with n vertices is $\Theta(n^{\omega+1}2^n)$ – worse than exponential.

3.2 Leveraging LLFD

As explained in the previous subsection, without theoretical knowledge on LLFD, one would use an algorithm with time complexity $\Theta(n^{\omega+1}2^n)$. In this subsection, we leverage theory developed in [2] to derive an algorithm that manages to answer the question in $\Theta(n^{\omega}2^n)$.

The following propositions inspire the algorithm with better asymptotic time complexity.

Proposition 5. *(After remark 4.1 in [2]) Every graph whose Laplacian matrix has a repeated eigenvalue is completely uncontrollable.*

An interesting fact first mentioned in [2], which was rediscovered here, is that completely uncontrollable graphs that do not have a repeated eigenvalue are rare. There are zero such graphs for $n \in \{1, 2, ..., 7\}$, only 10 such graphs with $n = 8$ (out of 2764) and 12 such graphs with $n = 9$ (out of 29750).

Proposition 6. *(Adapted from Proposition 1.ii in [2]) Let $A \in \mathbb{R}^{n\times n}$ be diagonalizable and have distinct eigenvalues. Let $v := U^{-1}b$ where U is a matrix whose columns are linearly independent eigenvectors of A. Then (A, b) is controllable if and only if no component of v is zero.*

Recall that the Laplacian matrix of a simple graph is always diagonalizable, because it is symmetric. Hence, the former two propositions apply, and we can classify a graph with the following algorithm:

1. Calculate the eigenvalues of L and U – the matrix whose columns are the eigenvectors of L.
2. If L has a repeated eigenvalue, $\mathcal{G}$ is completely uncontrollable.
3. Calculate U^{-1}.
4. For every control vector b, check controllability by checking whether $U^{-1}b$ has a zero element.
5. Determine controllability class of $\mathcal{G}$ based on whether all, some, or none of the control vectors lead to controllability.

The computational complexity of step 1 and 2 is polynomial,[5] and we only need to do them once (not for every control vector), so they will be dominated by the exponential complexity. Then for each control vector b, we only need to multiply $U^{-1}b$ and check if it has a zero element, which has time complexity $\Theta(n^\omega)$.

Hence, this classification algorithm has an improved complexity of $\Theta(n^2 2^n)$. After implementing both algorithms, we indeed found the time required to classify the vectors decreased by a factor of n. It is my understanding this is the algorithm the authors of [2] used to construct their Table I, but they do not discuss their algorithmic choice in their paper.

3.3 Shortcomings Stemming from Rounding

An important thing to note is that the algorithm developed in Subsection 3.2 has a limitation. Eigenvalues may be irrational and even if the actual eigenvalue is rational, finding roots of large (characteristic) polynomials can only be done numerically with a certain precision. When we check for equality of two eigenvalues, we thus have to specify a threshold (10^{-9} in our case) and consider two eigenvalues equal if they are closer than that. An analogous issue arises when checking if components of the eigenvectors are non-zero. Therefore, two eigenvalues may in fact (in theory) be different, but closer than our specified threshold. Our algorithm would mistakenly consider these eigenvalues identical, which could lead to a wrong classification.

One would expect this should not be an issue with the simple algorithm, as the controllability matrix is integral and hence its rank can be computed reliably. In practice, the rank of an integral matrix is checked by calculating the singular values of a matrix and checking if they are greater than zero. Here, the same issue with rounding occurs, and a tolerance has to be used. For example, $\begin{bmatrix} 1 & 0 \\ 10^8 & 1 \end{bmatrix}$ will be incorrectly assigned rank 1 by both Matlab and Julia. While there are exact methods for determining the rank of a matrix, we observed they easily overflow integers of fixed size and their performance is much worse than that of the forementioned method.[6]

In practice, the two algorithms agree for all graphs where $n \leq 8$, however, there are 741 graphs of cardinality 9 that are classified differently by the two methods. This corresponds to an error rate of 0.2838%. If the exact rank algorithm is used in the first algorithm instead, it takes three hours instead of several minutes to evaluate, but the results agree with the second method. While the second method is not guaranteed to classify a given graph correctly, it does so for all graphs where $n \leq 9$. As there are over $270\ 000$ such graphs, we are hopeful the second algorithm is likely to work for a particular larger graph.

3.4 Feasibility of a Better Algorithm

It is natural to ask if an algorithm with better asymptotic time complexity exists. The main limitation stems from the necessity to check half of the control vector individually, as the number of these vectors is exponential. So far, no theoretical result is known to us, that would allow for a faster classification algorithm.

[5] The popular QR Algorithm to compute the eigenvalues and eigenvectors has complexity $\Theta(n^3)$[11]. Matrix inversion can be done almost as fast as matrix multiplication.[4]

[6] There also exists an algorithm that does not overflow integers, but its complexity is also impractical, see [1].

3.5 Evaluation of Controllability classes of Small Graphs

For small graphs ($2 \leq n \leq 9$) we generated all unlabelled connected graphs using a program called $Nauty$[10] and then used the classification algorithm devised in Subsection 3.2 to analyse them. We stopped at $n = 9$ as the computational difficulty of constructing such table grows very fast. For example, it would take more than 88 times longer to evaluate the table for $n = 10$ than it did for $n = 9$.[7] The results are summarized in Table 1. A similar table was originally produced by the authors of [2]. We shall discuss the results in Table 1. The only graph with

n	Number of connected graphs	Essentially controllable		Conditionally controllable		Completely uncontrollable	
1	1	1	100.0%	0	0.00%	0	0.00%
2	1	1	100.0%	0	0.00%	0	0.00%
3	2	0	0.00%	1	50.00%	1	50.00%
4	6	0	0.00%	2	33.33%	4	66.67%
5	21	0	0.00%	10	47.62%	11	52.38%
6	112	4	3.57%	49	43.75%	59	52.68%
7	853	84	9.85%	505	59.20%	264	30.95%
8	11117	1992	17.92%	6361	57.22%	2764	24.86%
9	261080	94084	36.04%	137246	52.57%	29750	11.39%

Table 1: Enumeration of controllability classes for small unlabelled connected graphs.

$n = 1$ is a fascinating edge case of Definition 4. It technically satisfies the definition of essential controllability, as there are no binary control vectors outside the kernel of its Laplacian matrix – the system being controllable for all of them is true vacuously. Nevertheless, because the system is one dimensional, it is still controllable for $b = [1]$ – the only control vector for which it makes sense to call it essentially controllable.

There are no essentially controllable graphs with $3 \leq n \leq 5$. At $n = 6$ we see the essentially controllable graphs re-emerge.[8] For $n \geq 6$, with rising n the essentially controllable graphs are more common, while both conditionally controllable and completely uncontrollable become less common.

The last observation is that the number of unlabelled connected graphs rises fast with n. There are about five times as many of them with $n = 6$ than with $n = 5$, almost eight times as many with $n = 7$ than with $n = 6$, about 13 times as many with $n = 8$ than with $n = 7$, and more than 23 times more with $n = 9$ than with $n = 8$.

4 Resilient Graphs

As mentioned in Subsection 1.2, we want to investigate the resilience of LLFD to random changes to the graph. By a random change of the graph, we understand removing a random edge or removing a random vertex and all the edges adjacent to it.

Definition 5. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected graph with $n \geq 2$.*

[7] There are 261080 connected graphs for $n = 9$ and there are 11716571 connected graphs for $n = 10$, which is aproximately 44 times higher. For each graph, we will need more than twice as long, due to the complexity of the algorithm.

[8] Along with the smallest asymmetric graphs.

*We call $\mathcal{G}$ **perfectly edge resilient** iff removing any one edge results in a graph that is connected and essentially controllable.*

*We call $\mathcal{G}$ **perfectly vertex resilient** iff removing any one vertex results in a graph that is connected and essentially controllable.*

We exclude the graph with one vertex from these definitions for two reasons. If the single vertex is removed, we obtain the empty graph, for which LLFD is not defined. Secondly, it is the only connected graph with no edge, so the condition for perfect edge resiliency is true vacuously. The following two conclusions follow from the definitions.

Proposition 7. *If a graph is perfectly edge resilient, it contains no bridges.*

Proof. We prove the contrapositive statement, which is equivalent: If a graph contains a bridge, it is not perfectly edge resilient.

Suppose that $\{v, w\}$ is a bridge of $\mathcal{G}$. By removing this bridge, we obtain a graph that is, by definition of a bridge, disconnected. Therefore, $\mathcal{G}$ is not perfectly edge resilient. □

Proposition 8. *The connected graph with two vertices is the only perfectly vertex resilient graph that contains a bridge.*

Proof. If we remove any of the two vertices of the connected graph with two vertices, we get the graph with one vertex (and no edges), which is essentially controllable. So the connected graph with two vertices is perfectly vertex resilient. The only edge in this graph is of course a bridge.

For the second part of the statement, we once again use proof by contrapositive. The connected graph with two vertices is the only graph with a bridge (or an edge of any kind) with less than three vertices. So let $n \geq 3$ and suppose $\{v, w\}$ is the bridge of a connected graph $\mathcal{G}$. As $\mathcal{G}$ is connected and $n \geq 3$, there is another vertex q which is connected to one of the vertices of the bridge, w.l.o.g. w. Let $\mathcal{G}' = (\mathcal{V} \setminus w, \mathcal{E})$. Then there is no path between v and q, by definition of a bridge. Hence, $\mathcal{G}'$ is disconnected, which means $\mathcal{G}$ is not perfectly vertex resilient. □

We went through all the connected unlabelled graphs with $2 \leq n \leq 9$. For each, we determined whether it is perfectly edge resilient by removing edges one at a time and using the algorithm devised in Subsection 3.2 to classify the resulting graph. Similarly, we removed vertices from the original graph one at a time to look for perfectly vertex resilient graphs. Recall that we did not define resiliency for $n = 1$ or for non-connected graphs. For graphs with $n > 9$, the computational difficulty is too high to consider them all. It took over an hour to consider all connected and unlabelled graphs with $n = 9$ and we estimate it would take more than a 100 times longer to do the same for $n = 10$.

The only connected graph with $n = 2$ is perfectly vertex resilient. At $n = 3$, there are no perfectly edge resilient graphs, but the cycle is completely uncontrollable, yet perfectly vertex resilient. There are no perfectly edge or vertex resilient graphs for $n \in \{4, 5, 6, 7\}$. For $n = 8$, there are 2 perfectly edge resilient graphs, both of which are essentially controllable, and 2 perfectly vertex resilient graphs, both of which are conditionally controllable.

	Conditionally controllable	Essentially controllable
Perfectly edge resilient	9	211
Perfectly vertex resilient	8	34

Table 2: Perfectly resilient graphs with 9 vertices.

For $n = 9$ the situation is summarized in Table 2. There are four graphs with $n = 9$, which are both perfectly vertex and edge resilient. These are the adjacency matrices of those four graphs, all of which are essentially controllable.

$$\begin{bmatrix} 0&0&0&0&1&1&0&1&1\\ 0&0&0&0&0&1&1&1&1\\ 0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&1&1&0\\ 1&0&0&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1&1\\ 0&1&1&1&1&1&0&0&1\\ 1&1&0&1&1&1&0&0&1\\ 1&1&0&0&1&1&1&1&0 \end{bmatrix} \quad \begin{bmatrix} 0&0&0&0&0&1&1&1&1\\ 0&0&0&0&0&1&0&0&1\\ 0&0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&0&1\\ 0&0&0&0&0&0&0&1&1\\ 1&1&0&0&0&0&0&1&1\\ 1&0&1&1&0&0&0&0&1\\ 1&0&1&0&1&1&0&0&0\\ 1&1&0&1&1&1&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} 0&0&0&0&0&1&1&1&1\\ 0&0&0&0&0&1&0&0&1\\ 0&0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&0&1\\ 0&0&0&0&0&0&0&1&1\\ 1&1&0&0&0&0&0&1&1\\ 1&0&1&1&0&0&0&1&0\\ 1&0&1&0&1&1&1&0&1\\ 1&1&0&1&1&1&0&1&0 \end{bmatrix} \quad \begin{bmatrix} 0&0&0&0&1&1&0&1&1\\ 0&0&0&0&0&1&1&1&1\\ 0&0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&0&1&1\\ 1&0&0&0&0&0&0&0&1\\ 1&1&0&0&0&0&0&1&1\\ 0&1&1&0&0&0&0&0&0\\ 1&1&1&1&0&1&0&0&1\\ 1&1&0&1&1&1&0&1&0 \end{bmatrix} \tag{5}$$

The main takeaway is that (at least for $n \leq 9$) the vast majority of essentially controllable graphs are neither perfectly vertex resilient, nor perfectly edge resilient. This is important from an engineering perspective – if one wishes to retain essential controllability under failure, they should intentionally incorporate resilience in the design of the network's topology.

5 Random Asymmetric Graphs and Essential Controllability

An important fact about LLFD is that any non-trivial graph automorphism prevents essential controllability. Hence, essentially controllable graphs are a subset of asymmetric graphs.

Proposition 9. *(Proposition 4.2 in [2]) All essentially controllable graphs with $n \geq 3$ are asymmetric.*

The converse of the preceding statement, however, is not true. On the other hand, as the authors of [2] point out, "The data in Table I suggests that the ratio [of the number essentially controllable to the number of asymmetric graphs] is monotonically increasing as n increases. It is an interesting problem to investigate if almost all asymmetric graphs are essentially controllable on $\{0,1\}^n$ as $n \to \infty$." That suggestion inspires this section.

n	Asymmetric graphs	Essentially controllable graphs	Ratio
6	8	4	50.00%
7	144	84	58.33%
8	3552	1992	56.08%
9	131452	94084	71.57%

Table 3: Ratio of the number of asymmetric to essentially controllable unlabelled connected graphs, as implied from TABLE I in [2].

As one can see in Table 3, the suggestion made by authors of [2] is not an entirely accurate description of their data. The ratio is lower for $n = 8$ than it is for $n = 7$. Nevertheless, this decrease at $n = 8$ might be an anomaly and whether this ratio increases for larger n is an interesting question, which we will investigate in this section.

As the number of unlabelled asymmetric connected graphs increases very fast with n, it is not feasible to go through all of them.[9] We will therefore sample random graphs to determine the ratio of the number of essentially controllable to the number of asymmetric graphs for a given n.

5.1 Generating Random Graphs

A straight forward way to create a random graph with n vertices is to consider each pair of vertices and create an edge between them with a given probability $p = 1/2$. This method of generating random graphs is called Erdős-Rényi random graph model[14].

In our case, we would like to sample unlabelled graphs uniformly, as it is the underlying structure of the graph that interests us. But the Erdős-Rényi model samples the labelled graphs uniformly. To illustrate the difference, we will examine labelled and unlabelled graphs for $n = 3$.

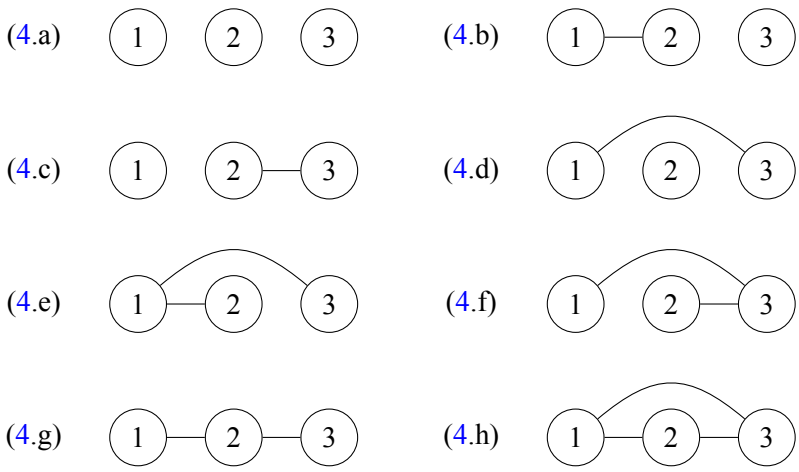

Figure 4: All graphs with $n = 3$ with labelled vertices.

Using the Erdős-Rényi model, every labelled graph in 4h as the same probability of being chosen.

[9] The number of labelled graphs with n vertices is $2^{\frac{n(n+1)}{2}}$. The number of unlabelled asymmetric connected with n vertices is difficult to work out exactly.[8]

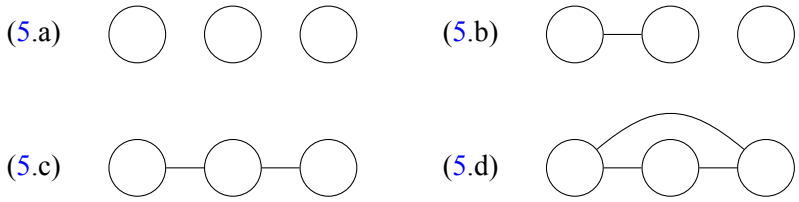

Figure 5: All graphs with $n = 3$ with unlabelled vertices.

Notice, that both (4.f), and (4.g) correspond to (5.c); but only (4.h) corresponds to (5.d). This means that (5.c) is twice as likely to be generated with the described procedure than (5.d). This contradicts our aim to sample unlabelled graphs uniformly.

How likely is then a given unlabelled graph to be generated?. The number of labelled graphs that correspond to a particular unlabelled graph is $n!/S$, where S is the number of automorphisms of the given graph – see equation 1.1.3 in [8] for details. The probability of a graph (with fixed size n) being generated with this algorithm is hence proportional to $1/S$.

For our application, we need to generate random graphs that are connected and asymmetric, as those are prerequisites for essential controllability. The probability of both of those properties increases with n[2]. We can simply generate a new graph if the one we obtained from the algorithm is disconnected or symmetric. By definition, the size of the automorphism group of asymmetric graph is 1, hence they have the same probability of being generated with the described algorithm.

5.2 Results

We can use the following formula to determine the number of times that a Bernoulli variable has to be sampled in order to achieve a given absolute error ε with confidence $1 - \alpha$ (Algorithm 1 in [9]):

$$n = \left\lceil \frac{\log(2/\alpha)}{2\varepsilon^2} \right\rceil$$

We will therefore sample 76820 graphs to find results to two significant digits, which corresponds to relative error of .005, with 95% confidence.

As one can see in the table below, the ratio of Essentially controllable graphs to asymmetric graphs does not increase with every increment in n, but it seems to tend to 1 as n grows.

n	9	10	11	12	13	14	15	16
Essentially Controllable	72%	72%	89%	86%	96%	95%	98%	99%

Table 4: Ratio of Essentially controllable graphs to asymmetric graphs.

We noticed that whenever the ratio of essentially controllable graphs to asymmetric graphs decreases when incrementing n, it is always when going from an odd n to even n.

6 Conclusion

In Section 4 we defined the notions of perfect vertex and edge resilience, we examined all graphs $2 \leq n \leq 9$ and observed that the complete graph with three vertices (the triangle graph) is the only

perfectly vertex resilient graph that is also completely uncontrollable. Furthermore, all 19 conditionally controllable graphs with $n \in \{8, 9\}$ that are either perfectly vertex resilient or perfectly edge resilient were controllable with the majority of the control vectors. This leads us to state the following conjectures:

Conjecture 1. *The complete graph with three vertices is the only graph of any size that is both completely uncontrollable and perfectly vertex resilient.*

Conjecture 2. *There are no graphs that are completely uncontrollable and perfectly edge resilient.*

Surprisingly, we found a graph does not have to be essentially controllable to be perfectly vertex resilient or perfectly edge resilient. The question remains if it has to be essentially controllable to be both perfectly vertex and perfectly edge resilient.

Conjecture 3. *The ratio of the number of essentially controllable to the number of asymmetric graphs, approaches* 1 *as* $n \to \infty$.

The former conjecture inspired Section 5, where we provide strong statistical indication it holds. However, to prove or disprove this notion remains an unsolved problem. Furthermore, our method sampled all unlabelled asymmetric graphs with equal probability. But in many practical scenarios, one would favour some topologies over others. For example, when controlling a swarm of drones, the hardware will have a limit on the number of connection each agent can make – the vertices in our topology have a maximum degree. For a given practical application, research should be done investigating this question using more appropriate random graph sampling.

References

[1] Jeffε (https://cstheory.stackexchange.com/users/111/jeff\%ce\%b5). *What is the actual time complexity of Gaussian elimination?* _eprint: https://cstheory.stackexchange.com/q/3921 Published: Theoretical Computer Science Stack Exchange. url: https://cstheory.stackexchange.com/q/3921.

[2] Cesar O. Aguilar and Bahman Gharesifard. "Graph Controllability Classes for the Laplacian Leader-Follower Dynamics". In: *IEEE Transactions on Automatic Control* 60.6 (2015), pp. 1611–1623. doi: 10.1109/TAC.2014.2381435.

[3] Jeff Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM review* 59.1 (2017). Publisher: SIAM, pp. 65–98. url: https://doi.org/10.1137/141000671.

[4] Demetres Christofides. *The asymptotic complexity of matrix reduction over finite fields*. June 23, 2014. doi: 10.48550/arXiv.1406.5826. arXiv: 1406.5826[cs,math]. url: http://arxiv.org/abs/1406.5826 (visited on 01/17/2024).

[5] Soon-Jo Chung et al. "A Survey on Aerial Swarm Robotics". In: *IEEE Transactions on Robotics* 34.4 (Aug. 2018). Conference Name: IEEE Transactions on Robotics, pp. 837–855. issn: 1941-0468. doi: 10.1109/TRO.2018.2857475. url: https://ieeexplore.ieee.org/abstract/document/8424838 (visited on 11/14/2023).

[6] D. Coppersmith and S. Winograd. "On the Asymptotic Complexity of Matrix Multiplication". In: *SIAM Journal on Computing* 11.3 (Aug. 1982). Publisher: Society for Industrial and Applied Mathematics, pp. 472–492. issn: 0097-5397. doi: 10.1137/0211038. url: https://epubs.siam.org/doi/abs/10.1137/0211038 (visited on 01/19/2024).

[7] James Fairbanks et al. *JuliaGraphs/Graphs.jl: an optimized graphs package for the Julia programming language*. 2021. url: https://github.com/JuliaGraphs/Graphs.jl/.

[8] Frank Harary and Edgar M. Palmer. *Graphical Enumeration*. Academic Press, Jan. 1, 1973. isbn: 978-0-12-324245-7. doi: 10.1016/B978-0-12-324245-7.50005-8. url: https://www.sciencedirect.com/science/article/pii/B9780123242457500058 (visited on 12/20/2023).

[9] Lan Jiang and Fred J. Hickernell. *Guaranteed Monte Carlo Methods for Bernoulli Random Variables*. Nov. 4, 2014. doi: 10.48550/arXiv.1411.1151. arXiv: 1411.1151[math,stat]. url: http://arxiv.org/abs/1411.1151 (visited on 12/05/2023).

[10] Brendan D. McKay and Adolfo Piperno. "Practical graph isomorphism, II". In: *Journal of Symbolic Computation* 60 (Jan. 2014), pp. 94–112. issn: 07477171. doi: 10.1016/j.jsc.2013.09.003. url: https://linkinghub.elsevier.com/retrieve/pii/S0747717113001193 (visited on 12/18/2023).

[11] Victor Y. Pan and Zhao Q. Chen. "The complexity of the matrix eigenproblem". In: *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*. STOC '99. New York, NY, USA: Association for Computing Machinery, May 1, 1999, pp. 507–516. isbn: 978-1-58113-067-6. doi: 10.1145/301250.301389. url: https://dl.acm.org/doi/10.1145/301250.301389 (visited on 01/19/2024).

[12] Jan Willem Polderman and Jan C. Willems. *Introduction to Mathematical Systems Theory: A Behavioral Approach*. Ed. by Jan Willem Polderman and Jan C. Willems. Texts in Applied Mathematics. New York, NY: Springer, 1998. isbn: 978-1-4757-2953-5. doi: 10.1007/978-1-4757-2953-5_4. url: https://doi.org/10.1007/978-1-4757-2953-5_4 (visited on 12/21/2023).

[13] Wei Ren and Yongcan Cao. *Distributed Coordination of Multi-agent Networks: Emergent Problems, Models, and Issues*. Communications and Control Engineering. London: Springer, 2011. isbn: 978-0-85729-168-4 978-0-85729-169-1. doi: 10.1007/978-0-85729-169-1. url: https://link.springer.com/10.1007/978-0-85729-169-1 (visited on 10/18/2023).

[14] Alfred Rényi and Paul Erdős. "On random graph". In: *Publicationes Mathematicate* 6 (1959), pp. 290–297.

[15] H.G. Tanner. "On the controllability of nearest neighbor interconnections". In: *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*. 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601). Vol. 3. ISSN: 0191-2216. Dec. 2004, 2467–2472 Vol.3. doi: 10.1109/CDC.2004.1428782. url: https://ieeexplore.ieee.org/abstract/document/1428782 (visited on 11/22/2023).

[16] James Usevitch and Dimitra Panagou. "Resilient Leader-Follower Consensus to Arbitrary Reference Values in Time-Varying Graphs". In: *IEEE Transactions on Automatic Control* 65.4 (Apr. 2020), pp. 1755–1762. issn: 0018-9286, 1558-2523, 2334-3303. doi: 10.1109/TAC.2019.2934954. url: https://ieeexplore.ieee.org/document/8795564/ (visited on 11/10/2023).

Parts of this text were originally used by the author as part of the bachelor thesis preparatory assignments (information literacy, research plan, and epistemic analysis).

Code for colour refinement was ported from the Python version developed for a group assignment on graph isomorphism.

AI tools were employed while writing the code, creating the figures, and to identify grammar, spelling and stylistic mistakes in the text.

7 Appendix (Code)

The latest version of the code can be found at github.com/MikiVanousek/bcs-thesis.

7.1 main.jl

```
"""
LLFD Graph Classification algorithm and other code written with LLFD theory insight.
"""
module LLFD

using Graphs
using GraphIO.Graph6
using GraphIO.EdgeList
using Graphs.LinAlg
using SparseArrays
using LinearAlgebra

include("util.jl")
using .LLFDUtil

@enum ContClass essentially conditionally uncontrollable
export ContClass, essentially, conditionally, uncontrollable

cont_vecs_cache = Dict{Int64,Vector{Vector{Int64}}}()
function cont_vecs(n::Int64)
    if !(n in keys(cont_vecs_cache))
        cont_vecs_cache[n] = generate_cont_vecs(n)
    end
    return cont_vecs_cache[n]
end

struct LaplacianGraph
    g::SimpleGraph
    cont::Vector{Bool}
    score::Float64
    class::ContClass
    L::Matrix{Int64}
end
function LaplacianGraphBF(g, is_ctr=is_controllable)
    L = Matrix(laplacian_matrix(g))
    cont = [is_controllable(L, B) for B in cont_vecs(nv(g))]
    score = average(cont)
    if score == 0.0
        class = uncontrollable
    elseif score == 1.0
        class = essentially
    else
        class = conditionally
    end
    return LaplacianGraph(g, cont, score, class, L)
end
function LaplacianGraphSmart(g)
    L = Matrix(laplacian_matrix(g))

    vals, vecs = eigen(L)
    if repeated(vals)
        score = 0.0
        cont = [false for _ in 1:length(cont_vecs(nv(g)))]
        class = uncontrollable
```

```
        return LaplacianGraph(g, cont, score, class, L)
    end
    U = vecs^-1
    has_zero(k) = any(almost_zero.(k))
    ctrl(B) = !has_zero(U * B)
    cont = [ctrl(B) for B in cont_vecs(nv(g))]

    score = average(cont)
    if score == 0.0
        class = uncontrollable
    elseif score == 1.0
        class = essentially
    else
        class = conditionally
    end
    LaplacianGraph(g, cont, score, class, L)
end
export LaplacianGraph, LaplacianGraphBF, LaplacianGraphSmart


function edge_modifications(g)
    res = []
    for e in edges(g)
        gm = copy(g)
        rem_edge!(gm, e)
        push!(res, gm)
    end
    return res
end
edge_resilience_essentially(lg) = average(gm.class == essentially for gm in
    LaplacianGraph.(edge_modifications(lg.g)))
function find_perfect_edge_resilience(lgs)
    perfect = []
    for (i, lg) in enumerate(lgs)
        if i % 1000 == 0
            println("perfect_progress ", 100i / length(lgs))
        end
        if edge_resilience_essentially(lg) == 1.0
            push!(perfect, lg)
        end
    end
    return perfect
end

function vertex_modifications(g)
    res = []
    for v in vertices(g)
        gm = copy(g)
        rem_vertex!(gm, v)
        push!(res, gm)
    end
    return res
end
vertex_resilience_essentially(lg) = average(gm.class == essentially for gm in
    LaplacianGraph.(vertex_modifications(lg.g)))
function find_perfect_vertex_resilience(lgs)
    perfect = []
    for (i, lg) in enumerate(lgs)
        if i % 1000 == 0
            println("perfect_progress ", 100i / length(lgs))
        end
```

```
        if vertex_resilience_essentially(lg) == 1.0
            push!(perfect, lg)
        end
    end
    return perfect
end
export find_perfect_edge_resilience, find_perfect_edge_resilience

function random_graph(n, p=0.5, condition=(g) -> true)
    while true
        g = SimpleGraph(n)
        for (i, v1) in enumerate(vertices(g))
            for v2 in view(vertices(g), i+1:nv(g))
                if rand() < p
                    add_edge!(g, v1, v2)
                end
            end
        end
        if condition(g)
            return g
        end
    end
end

export random_graph, random_graph_connected_asymmetric

graphs_dict(n) = loadgraphs("./graphs/connected-$n", Graph6Format())
graphs_list = values  graphs_dict
function separate(lgs::Vector{LaplacianGraph})
    res = Dict(i => [] for i in instances(ContClass))
    for g in lgs
        push!(res[g.class], g)
    end
    @assert sum(length.(values(res))) == length(lgs)
    return res
end
export graphs_dict, graphs_list, separate

end
```

7.2 util.jl

```
"""
Utilities and helper function, that do not contain essential insights from LLFD
    theory.
"""
module LLFDUtil
using LinearAlgebraX

almost_eq(a, b, tol=10^-9) = abs(a - b) < tol
almost_zero(v) = all(almost_eq.(0, v))
orth(a, b) = almost_eq(0, a  b)
repeated(v) = any(almost_eq(v[i], v[i+1]) for i in 1:length(v)-1)
export almost_eq, almost_zero, orth, repeated

function controllability_matrix(A, B)::Matrix{Int64}
    @assert (n = length(B)) == size(A, 1) == size(A, 2)
    res = zeros(Int64, n, n)
    res[:, 1] = B
    for i in 2:n
        res[:, i] = A * res[:, i-1]
```

```
    end
    return res
end
function is_controllable(A, B)
    @assert (n = size(A, 1)) == size(A, 2)
    rank(controllability_matrix(A, B)) == n
end
function is_controllable_x(A, B)
    @assert (n = size(A, 1)) == size(A, 2)
    rankx(controllability_matrix(A, B)) == n
end
export is_controllable, is_controllable_x

function all_bin_vecs(n::Int64)
    if n == 0
        return []
    elseif n == 1
        return [[0], [1]]
    end
    smaller = all_bin_vecs(n - 1)
    return vcat([[s; 0] for s in smaller], [[s; 1] for s in smaller])
end
generate_cont_vecs(n::Int64) = [push!(b, 0) for b in all_bin_vecs(n - 1) if sum
    (b) > 0]
export generate_cont_vecs

average(iter) = sum(iter) / length(iter)
export average

class_counts(dict) = Dict(k => length(v) for (k, v) in dict)
export class_counts

function track_progress(i, k, reports=10)
    if i % floor(k / reports) == 0
        println("progress $(round(100i/k, digits=2))%")
    end
end
export track_progress

end
```

7.3 test.jl

```
"""
Here we test functions from main.jl and use them to find numerical results to
    some of our research questions.
"""

using Pkg
Pkg.activate(".")

using Plots
using GraphRecipes
using Graphs
using CSV
using DataFrames

include("main.jl")
using .LLFD
include("util.jl")
using .LLFDUtil
```

```
include("color_refinement.jl")
using .ColorRefinement

show(g::Graph) = graphplot(g, show=true)

function assert_classification(expected, actual)
    total = sum   values
    if total(expected) != total(actual)
        prinln("Expected and actual disagree on the number of graphs!")
    end
    if !(expected == actual)
        println("actual - expected")
        diff = Dict(e => actual[e] - expected[e] for e in instances(ContClass))
        display(diff)
        println("Percentage wrong: ", sum(abs.(values(diff))) / 2 / sum(value(
    expected)) * 100)
        error("Wrong answer!")
    end
end

function test_classification(n)
    correct_classes = Dict(
        2 => Dict(
            essentially => 1,
            uncontrollable => 0,
            conditionally => 0,
        ),
        3 => Dict(
            essentially => 0,
            uncontrollable => 1,
            conditionally => 1,
        ),
        4 => Dict(
            essentially => 0,
            uncontrollable => 4,
            conditionally => 2,
        ),
        5 => Dict(
            essentially => 0,
            uncontrollable => 11,
            conditionally => 10,
        ),
        6 => Dict(
            essentially => 4,
            uncontrollable => 59,
            conditionally => 49,
        ),
        7 => Dict(
            essentially => 84,
            uncontrollable => 264,
            conditionally => 505,
        ),
        8 => Dict(
            essentially => 1992,
            uncontrollable => 2764,
            conditionally => 6361,
        ),
        9 => Dict(
            essentially => 94084,
            uncontrollable => 29750,
            conditionally => 137246,
```

```
        ),
    )
    separated = separate(LaplacianGraphSmart.(graphs_list(n)))
    cc = class_counts(separated)
    display(cc)
    assert_classification(correct_classes[n], cc)
end

function test_resilience()
    resilience_scores = [edge_resilience_essentialy(es) for es in separated[
    essentially]]
    display(resilience_scores)
    println(maximum(resilience_scores))
    resilience_scores
end

function save_graphs(lgs, filename, graphs_dict)
    gs = getproperty.(lgs, :g)
    d = Dict(k => v for (k, v) in graphs_dict if v in gs)
    println("number of perfect: ", length(d))
    if length(d) > 0
        savegraph(filename, d)
    end
end
function find_perect_vertex(n)
    graphs_dict, graphs, separated, lgraphs = init(n)
    save_graphs(find_pefect_vertex_resiliance(separated[essentially]), "
    vertex_perfect-$n", graphs_dict)
end

function find_perfect(n)
    graphs_dict, graphs, separated, lgraphs = init(n)
    edge_perfect = []
    vertex_perfect = []
    for e in (essentially, conditionally, uncontrollable)
        println(e)
        ep = find_pefect_edge_resiliance(separated[e])
        println("edge ", length(ep))
        append!(edge_perfect, ep)
        vp = find_pefect_vertex_resiliance(separated[e])
        println("vertex ", length(vp))
        append!(vertex_perfect, vp)
        println()
    end
    println("Total edge perfect: ", length(edge_perfect))
    println("Total vertex perfect: ", length(vertex_perfect))
    save_graphs(edge_perfect, "edge_perfect", graphs_dict)
    save_graphs(vertex_perfect, "vertex_perfect", graphs_dict)
end
function test_color_ref()
    n = 9

    graphs_dict, graphs, separated, lgraphs = init(n)
    test_color_ref()

    for e in instances(ContClass)
        println(e)
        average(has_discrete(lg.g) for lg in separated[e]) |> println
    end
end
function test_asymetric_color_ref()
```

```
137     include("color_refinement.jl")
138     n = 9
139     asgs = loadgraphs("graphs/asymetric-$n", Graph6Format())
140     println("asymetric grahps: ", length(asgs))
141     average(has_discrete(g) for g in values(asgs)) |> println
142     @assert all(asymmetric(g) for g in values(asgs))
143 end
144
145 function test_random(n, k)
146     include("color_refinement.jl")
147     ess = 0
148     con = 0
149     unc = 0
150     color_refined_ess = 0
151     color_refined_cond = 0
152
153     for i in 1:k
154         if 100i % k == 0
155             println("progress $(100i/k)%")
156         end
157         g = random_connected_graph(n)
158         lg = LaplacianGraph(g)
159         if lg.class == essentially
160             ess += 1
161             hd = has_discrete(g)
162             if !hd
163                 println("Counter example!")
164                 display(Matrix(adjacency_matrix(g)))
165             end
166             color_refined_ess += hd
167
168         elseif lg.class == conditionally
169             con += 1
170             color_refined_cond += has_discrete(g)
171         else
172             unc += 1
173         end
174     end
175     println("Probability essentially: ", ess / k)
176     println("Probability conditionally: ", con / k)
177     println("Probability color refined (essentially): ", color_refined_ess /
        ess)
178     println("Probability color refined (conditionally): ", color_refined_cond /
         con)
179
180 end
181 function test_random_asymetric(n, k)
182     include("color_refinement.jl")
183     found = 0
184     for i in 1:k
185         if 100i % k == 0
186             println("progress $(100i/k)%")
187         end
188         g = random_connected_graph(n)
189         if asymmetric(g) && !has_discrete(g)
190             found += 1
191         end
192     end
193     println("Rebels ", found)
194 end
195 function test_random(n, decimal_prec=2)
```

```
196     function random_graph_connected_asymmetric(n, p=0.5)
197         @assert n >= 6 "There are no asymmetric graphs with less than 6
        vertices!"
198         random_graph(n, p, (g) -> is_connected(g) && asymmetric(g))
199     end
200
201     signif = 0.95
202     a = 1 - signif
203     abs_error = 4.9 * 10.0^(-(decimal_prec + 1))
204     trials = ceil(Int64, log(2 / a) / (2abs_error^2))
205
206     println("Trials needed for $decimal_prec (abs error $abs_error) at
        significance $signif:\n$trials")
207     ess = 0
208     for i in 1:trials
209         track_progress(i, trials)
210         g = random_graph_connected_asymmetric(n)
211         lg = LaplacianGraphSmart(g)
212         if lg.class == essentially
213             ess += 1
214         end
215     end
216     prob_ess = round(ess / trials, digits=decimal_prec)
217     println("For n=$n")
218     println("Probability of essen. cont.: \t$prob_ess")
219     println()
220     return prob_ess
221 end
222
223 function analyze_random(range=9:16, decimal_prec=2)
224     df = DataFrame(n=Int[], ess=Float64[])
225
226     for n in range
227         ess = test_random(n, decimal_prec)
228         push!(df, (n, ess))
229     end
230     CSV.write("random_graph_analysis.csv", df)
231 end
```

7.4 graphs/gen-graphs.sh

You have to run this script to generate all files with unlabelled and asymmetric graphs.

```
1  # Install Nauty before use for the geng and pickg binaries!
2  # https://pallini.di.uniroma1.it/
3  # Check if exactly one argument is provided
4  if [ $# -ne 1 ]; then
5      echo "Usage: $0 [n]"
6      echo "   [n] is the number of vertices in the graph.
7      exit 1
8  fi
9
10 touch connected-$1
11 geng -c $1 > connected-$1
12 pickg -a1 < connected-$1 > asymetric-$1
```