FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Schelling games: na neústupných agentech záleží
Student:	Ondřej Nohava
Vedoucí:	Ing. Šimon Schierreich
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Pro studium rezidenční segregace vytvořil Schelling [1,2] následující model. Mějme agentů dvou různých typů umístěných na mřížku. O agentu i řekneme, že je spokojený se svým bydlištěm, pokud je alespoň $t \in (0,1]$ agentů v jeho okolí daném poloměrem r stejného typu, jako agent i. Pokud je tento jeho požadavek splněný, nemá agent i důvod se stěhovat, v případě nespokojenosti bude mít ovšem tendenci své bydliště měnit. V této práci se budeme věnovat variantě, kterou představili Elkind et al. [3]. Zde neumisťujeme agenty na mřížku, ale jako topologie, do které jsou agenti umisťováni, slouží jednoduchý neorientovaný graf. V tomto modelu autoři navíc uvažují dva druhy agentů – strategické a neústupné agenty. Zatímco strategičtí agenti jsou ochotni v případě nespokojenosti se svou lokací své bydliště měnit, u neústupných agentů toto neplatí. Tito nikdy svou lokaci nemění. Výstupem problému je (většinou) takové umístění agentů na graf, při kterém žádný z agentů nemá tendenci své bydliště měnit a při kterém je maximalizován sociální blahobyt. Prozkoumejte variantu problému, kdy je standardním způsobem měřena spokojenost i neústupných agentů. Změní se nějak známé algoritmické a těžkostní výsledky?

[1] Thomas C. Schelling. Models of segregation. Am. Econ. Rev. 59 (2) (1969) 488-493.
[2] Thomas C Schelling. Dynamic models of segregation. J. Math. Sociol. 1 (2) (1971) 143 - 186.
[3] Aishwarya Agarwal, Edith Elkind, Jiarui Gan, Ayumi Igarashi, Warut Suksompong, Alexandros A Voudouris. Schelling games on graphs. Art. Intell. 301 (2021) 103576. DOI: 10.1016/j.artint.2021.103576

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 13. května 2022 v Praze.

Bachelor's thesis

SCHELLING GAMES: WHEN STUBBORN AGENTS MATTERS

Ondřej Nohava

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Šimon Schierreich
June 29, 2023

Contents

List of Figures

List of Tables

I would like to express my deepest gratitude to my supervisor Ing. Šimon Schierreich for his guidance, patience and helpful comments. I would also like to thank VýLet, the summer internship program that supports student research. At last I would like to thank my family and all of my friends for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. I further declare that I have concluded a license agreement with the Czech Technical University in Prague on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act. This fact shall not affect the provisions of Article 47b of the Act No. 111/1998 Coll., the Higher Education Act, as amended.

In Prague on June 29, 2023 .....

Abstract

We study strategic games inspired by Schelling's segregation model. In these games, we study how agents of multiple types are moving on undirected graphs. We consider two types of agents: *strategic* agents aim to maximize the fraction of their neighbors who have the same type, while *stubborn* agents do not move at all. We explore two common variants of the model: *jump games*, where agents can *jump* to empty nodes, and *swap games*, where agents move by swapping positions with other agents. We investigate the computational complexity of these games and the increase in social welfare when considering neighbors of stubborn agents. We propose a novel approach to classify games with maximal social welfare in *equilibrium* using a graph neural network.

Keywords Schelling games, social welfare analysis, stubborn agents, computational complexity, graph neural networks, graph classification

Abstrakt

Zabýváme se studiem strategických her, které jsou inspirovány Schellingovým segregačním modelem. V těchto hrách zkoumáme, jak se agenti, rozdělení do několika typů, pohybují na neorientovaných grafech. Bereme v potaz dva typy agentů: *strategické* agenty, kteří se snaží o maximální počet sousedů stejného typu a *neústupné* agenty, kteří se vůbec nepohybují. V naší práci zkoumáme dvě nejčastější varianty tohoto modelu, které se rozlišují pohybem strategických agentů. Strategičtí agenti mají k dispozici dva druhy pohybů: *skok* a *výměnu*. Na těchto hrách zkoumáme výpočetní složitost a zvýšení sociálního blahobytu při zvážení sousedů i neústupných agentů. Dále představujeme nový způsob detekce maximálního sociálního blahobytu v *ekvilibriu* za pomoci grafových neuronových sítí.

Klíčová slova Schelling hry, analýza sociálního blahobytu, neústupní agenti, výpočetní složitost, grafové neuronové sítě, klasifikace grafů

Acronyms

MSW	Maximal Social Welfare
MSWE	Maximal Social Welfare at Equilibrium
R	Set of Stubborn Agents
S	Set of Strategic Agents
SW	Social Welfare
GNN	Graph neural network
PDF	Portable document format

Introduction

Segregation has been monitored by economists and sociologists for a long time, and their research has shown that a community of people tends to create homogeneous groups over time based on religion, politics, race, etc. This phenomenon's most famous example is racial segregation in the US.[1] For better understanding, we can even show an example of two neighboring cities where one race clearly dominates the others. These are the City of Detroit, which in 2019 was occupied by 78% African Americans and his neighbor, Oakland Country, which had, on the other hand, 75% white residents.

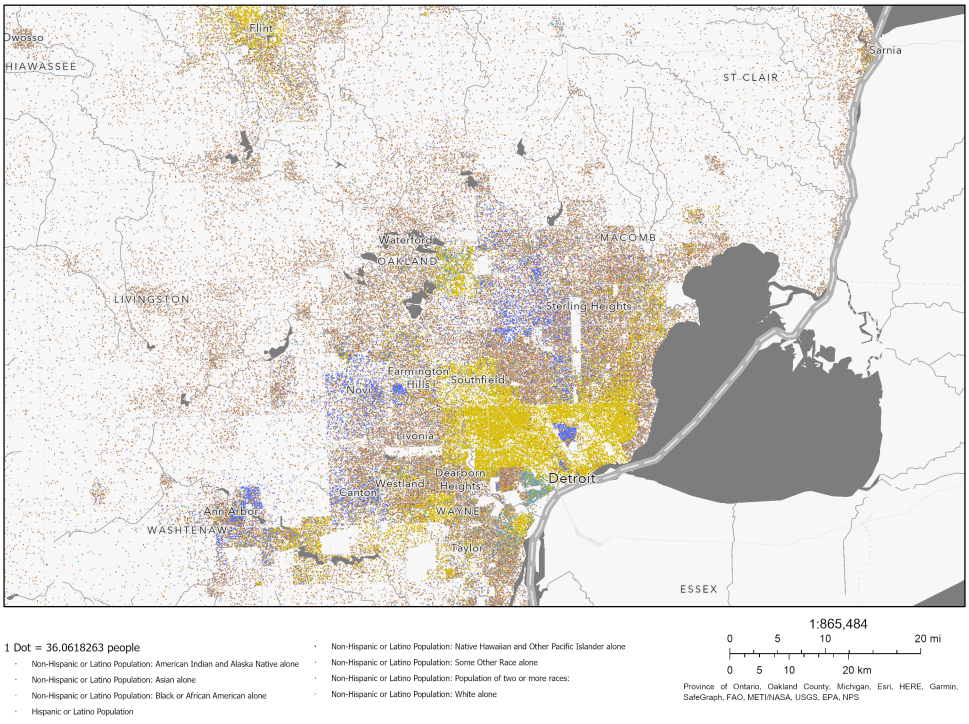

Figure 1 Race and ethnicity in the US by Dot Density (Census 2020) [1].

[1] See the racial dot map [1] for visualization.

One of the first attempts to explain residential segregation was a model presented by Thomas Schelling more than 50 years ago [2, 3]. In this model, there are agents of two different types placed on a line or a grid with each agent having a tolerance threshold parameter $\tau \in (0, 1]$. The agent's happiness is defined by her neighborhood, that is, if at least τ fraction of agents in the agent's radius, meaning agents connected by the edges to the agent's node, are of the same type, then the agent remains at her location. Contrarily, the agent can either swap locations with another unhappy agent or jump to an empty spot to increase her threshold. Schelling showed in relatively simple experiments that even if $\tau \leq \frac{1}{2}$ agents eventually form segregated clusters.

Schelling's work has deepened interest in segregation and inspired many economists, computer scientists and physicists to study related models. Most subsequent works analyzed the problem via agent-based simulations[2] with only two types of agents, where agents move randomly. However, they proved that even when agents behave randomly, they, with high probability, form monochromatic regions, showing that strong segregation is likely to occur; see, for example, works [5, 6, 3].

More recent studies [7, 8] took a different approach and considered agents to be fully deterministic, thus agents now move only when they improve their happiness, which differs from earlier works that moved agents randomly. Most interest in Schelling games followed up after publication of an article from Agarwal et al. [9], which is the first to consider more than just two agent types.

Our contribution

The work presented in this thesis is based on a very recent article by Agarwal et al. [9] that we will follow up on. Their article is based on the model presented by Chauhan et al. [7]. They made some adjustments to make the problem less complex. In their basic model, they also integrate *location preferences*, but in just two basic agent types – *stubborn* and *strategic*. Location preferences try to present the behavior and preferences of agents, but considering them in expanded form makes resolving the problem much harder[3]. For simplification, Agarwal et al. [9] only considered, as mentioned above, strategic agents that move to get to the best possible location, and stubborn agents that stay at the same location without ever changing. Stubborn agents are meant to depict people unwilling to change in any situation (such as indebted people or pensioners). In these games, they also do not consider any kind of threshold, that is, the agent moves as long as she can change to a location with a better happiness ratio (that is, $\tau < 1$) available. Lastly, Agarwal et al. [9] are the first to consider k types of agents.

Although they introduced stubborn agents, they did not consider their utilities, which could highly impact the social welfare of the Schelling game and might not always pay off. We will investigate this possibility on the same model as them and show how the results differ.

In Chapter 2, we prove that maximizing social welfare with consideration of stubborn agents' utilities is NP-complete. We then present an algorithm to compute the maximal social welfare at equilibrium in Chapter 3. Provided algorithm runs in polynomial time for two types of agents, but only works on tree graphs. Based on algorithm outputs, we compare both models, and present two problems that can affect the topology when we do not consider stubborn agents' utilities.

Finally, in Chapter 4, we discuss an alternative approach to evaluating Schelling games, that is, using the new concept of graph neural networks. We implement a graph neural network for classifying maximal social welfare at equilibrium for a given Schelling game and compare our model performance on tree and general graphs.

Overall, this thesis aims to invoke further interest in the role of stubborn agents in Schelling games.

[2] See an agent-based simulation's [4] online implementation.

[3] We can consider, for example, distance, position near institution (school, police), physical disability, etc.

We also try to make use of rapid development in machine learning to combine the usage of machine learning models for easier and faster analysis of computer science problems.

Related work

For an introduction to the Schelling model, we advise interested readers to go to Chapter 4 of the book by Easley and Kleinberg [10], as well as the papers by Brandt et al. [11] and Immorlica et al. [12]. Aside from the work of Agarwal et al. [9] and Chauchen et al. [7], which we previously discussed in detail, several other authors studied similar models. In particular, Chauchen et al. model's results were strengthened by Echzell et al. [13] who confirmed the existence of Nash equilibrium and convergence in their model. Bilò et al. [14] investigated the use of non-monotone utility functions on swap games and showed that moving to non-monotone utilities can bring new structural properties and different equilibrium possibilities. Very recently Friedrich et al. [15] considered non-monotone jump Schelling games and showed that even for simple topologies equilibrium may cease to exist. Another approach to achieve a more general model was recently presented Bilò et al. [16]. In their model, they introduced a new decisive feature that considers non-categorical types to be possible.

Our model is relatively similar to Hedonic games [17, 18, 19]. In Hedonic games, agents form coalitions in order to satisfy their individual preferences. Both Hedonic and Schelling games can be used to model the formation of social networks.

In Chapter 4, we dive into graph neural networks [20], which were also introduced quite recently. We refer interested readers to the article by Zhou et al. [21] and an older article by Scarselli et al. [22].

Chapter 1

Schelling games

In this chapter, we aim to create a solid foundation for understanding Schelling games and related concepts. We start by introducing the notation for graph theory and computational complexity. After that, we define Schelling games. This chapter should create a general understanding of the issue to follow our analysis.

1.1 Graph theory

We begin by defining concepts from graph theory. In our definitions, we follow the basic notation by Diestel [23].

▶ **Definition 1.1** (Graph [23])**.** *A graph is a pair $G = (V, E)$ where V is a non-empty set of vertices and $E \subseteq [V]^2$ is a set of edges.*

Since our definition explicitly prohibits self-loops and multiple edges between the same vertices, it is clear that we only work with simple undirected graphs. Apart from general graphs for our topologies, we will also be using some well-known graph structures, such as *stars*, *paths* or *cycles* in our proofs (see Figure 1.1 for visualization).

A *star*, denoted by S^k consists of a central vertex connected to other vertices, where k represents the number of vertices connected to the central vertex. A *path* (P^k) is a set of linearly arranged vertices where each neighbor vertices is an edge; hence it does not have cycles. Lastly, a *cycle* (C^k) is a path in which the first and last vertex of the path is connected by an edge. In both paths and cycles, k refers to the total number of vertices. In addition to these graphs, we also consider a graph called a *tree*, which is similar to a star except that any two vertices are connected by exactly one unique path; in other words, a tree is a connected acyclic graph.

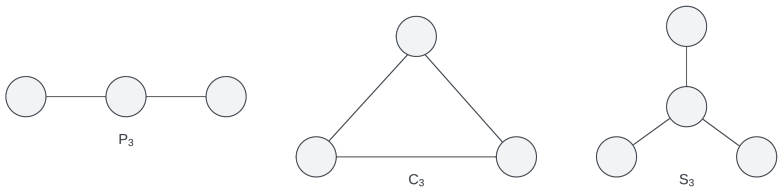

Figure 1.1 An example of a path, a cycle, and a star in this order.

▶ **Definition 1.2** (Complement [23]). *A graph G^C is a* complement *of $G = (V, E)$ if its edges are $[V]^2 \setminus E$.*

▶ **Definition 1.3** (Clique [23]). *A* clique*, C, in an undirected graph $G = (V, E)$ is a subset of vertices, $C \subseteq V$, such that for every pair of distinct vertices $\{u, v\} \in C$, there is an edge $\{u, v\} \in E$.*

▶ **Definition 1.4** (Partition [23]). *A set $\mathcal{A} = \{A_1, \ldots, A_k\}$ of disjoint subsets of a set A is a* partition *of A if $A = \bigcup_{i=1}^{k} A_i$ and $A_i \neq \emptyset$ for every i.*

We use partition to define last type of graph we use called a *bipartite graph.*

▶ **Definition 1.5** (Bipartite graph [23]). *Let $r = 2$ be an integer. A graph $G = (V, E)$ is called* bipartite *if V admits a partition into r classes such that every edge has its ends in different classes.*

Our definition of a bipartite graph can be extended for $r - partite$ graphs by setting $r \geq 2$.

1.2 Computational complexity

Computational complexity investigates problems using various computational models. We base our complexity theory on Turing machines [24].

▶ **Definition 1.6** (Turing machine [25]). Turing machine $\mathcal{M}$ *is a tuple (Γ, Q, δ) that contains:*

- *A set Γ of symbols that $\mathcal{M}$ can work with. We assume that Γ contains two designated symbols, "blank" symbol $\square$ and "start" symbol $\triangleright$ and binary numbers. We call Γ the alphabet of $\mathcal{M}$.*
- *A set Q of finite states of the machine $\mathcal{M}$. We assume that Q contains a start state, denoted q_{start} and a halting state, denoted q_{halt}.*
- *A transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the rule $\mathcal{M}$ uses in performing each step.*

The Turing machine can be viewed as a simplified version of a modern computer, with the set Q corresponding to the computer's memory, and the transition function and symbols from Γ as the computer's central processing unit. However, it is better to think of Turing machines as a formal way of describing algorithms. Although algorithms are often best understood with natural languages, using a formalism can be beneficial when discussing them mathematically. The Turing machine we have defined is known as a *deterministic* Turing machine. It's worth noting that there also exists a *non-deterministic* Turing machine, which was originally used to define complexity class NP [25].

Finally, we can proceed to describe our first complexity class, called P. This is a very important class, as it contains decision problems that can be solved efficiently.

▶ **Definition 1.7** (Complexity class P [25]). *Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We let $DTIME(T(n))$ be the set of all Boolean functions that are computable in $c \cdot T(n)$-time for some constant $c > 0$, then $P = \bigcup_{c \geq 1} DTIME(n^c)$.*

We can now move on to describing a much more comprehensive class called NP. We find the most well-known computational problems in this class.

▶ **Definition 1.8** (Complexity class NP [25]). *A language $L \subseteq \{0,1\}^*$ is NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing machine $\mathcal{M}$ such that for every $x \in \{0,1\}^*$,*

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } \mathcal{M}(x, u) = 1$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $\mathcal{M}(x, u) = 1$ then we call u a certificate for x.

To put it more simply, all the solutions of NP problems can be easily verified. Furthermore, based on these definitions, we can easily see that $P \subset NP$, but it remains one of the most important problems in computer science to prove that $P = NP$ ($P \neq NP$). We capture the difference between P and NP using a characteristic called NP-hardness.

▶ **Definition 1.9** (Reductions, NP-hardness and NP-completeness [25]). *We say that a language $A \subseteq \{0,1\}^*$ is polynomial-time Karp reducible to a language $B \subseteq \{0,1\}^*$ denoted by $A \leq_p B$ if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*, x \in A$ if and only if $f(x) \in B$.*

- *We say that B is NP-hard if $A \leq_p B$ for every $A \in NP$.*
- *We say that B is NP-complete if B is NP-hard and $B \in NP$.*

Several NP-complete problems have been extensively studied in the literature, for example, the Traveling Salesman Problem [26], the Knapsack Problem [27], and the Maximum Cut Problem [28]. These and other problems are not known to be solvable in polynomial time, which led most scientists to believe that $P \neq NP$.

1.3 Schelling game

Schelling games on graphs are a class of games that explore the behavior of agents in unique networks. In these games, the agents are placed on a graph and each agent is assigned *utility function* that calculates their happiness. The most obvious goal of the game is for each agent to have utility that they can no longer increase. We call this *pure Nash equilibrium* (or, simply, an equilibrium).

We have already stated that our model is based on the model by Agarwal et al. [9]; hence our definitions are similar.

▶ **Definition 1.10** (Schelling game). *A* Schelling game *is given by a set $N = \{1, ..., n\}$ of $n \geq 2$ agents partitioned into $k \geq 2$ pairwise disjoint types $T_1, ..., T_k$, and a graph $G = (V, E)$, called the topology.*

Agent types T_i are often identified with colors; for example, in Schelling games with two types T_1 and T_2, agents are referred to as red (T_1) and blue (T_2). In addition to these types, we also divide our agents into two subsets S and R, where agents in the set S are *strategic* and agents in the set R are *stubborn*. Strategic agents move in order to maximize their utility[1] and stubborn agents do not deviate and stay at node they were associated with.

▶ **Definition 1.11** (Assignment). *An* assignment *is a vector $\mathbf{v} = (v_1, ..., v_n)$, where*

$$\forall i \in N : v_i \in V \ \wedge \ \forall i, j \in N, \ i \neq j : v_i \neq v_j$$

Intuitively, the location of agent i is v_i and a single node cannot be occupied by two or more agents. We need to mention that we need to pay attention to the difference between topology and assignment. The topology, as we know, is a simple undirected graph, and the assignment $\mathbf{v}$ defines the position of agents on this graph.

▶ **Definition 1.12** (Utility function). Utility function *(utility), for agent u_i in assignment $\mathbf{v}$ with $N_i(\mathbf{v})$ being a set of all neighbors of agent i and F_i set of agents of the same type as i, is*

$$u_i(\mathbf{v}) = \frac{|N_i(\mathbf{v}) \cap F_i|}{|N_i(\mathbf{v})|}, \ u_i(\mathbf{v}) \geq 0$$

[1] In some articles they are referred to as *fully strategic agents* as they do not consider any kind of threshold.

▶ Note 1.13. You can see that the utility function can contain a division by zero. Therefore, we consider $\frac{0}{0} = 0$. We explicitly state when this convention is necessary.

Strategic agent i in an assignment $\mathbf{v}$ can maximize her utility either by jumping to an empty node or swapping with another agent. Based on agent movement, we call these games *k-jump games* and *k-swap games*. We introduce additional constraints to these movements, that is, in jump games, the agent can only jump if she increases her utility, and in swap games, the agent can only swap if both agents increase utility by swapping. In a given assignment, agents can only move by jumping or swapping. We do not consider games with both types of moves, although it could be an interesting topic to consider in the future work.

▶ **Definition 1.14** (Social welfare)**.** Social welfare *(SW) on an assignment* $\mathbf{v}$ *is*

$$SW(\mathbf{v}) = \sum_{i \in N} u_i(\mathbf{v})$$

▶ Note 1.15. Notice that we take the sum over the set of all agents, not just strategic agents.

▶ **Example.** For illustration of the computation of the utility function and social welfare, we consider an assignment $\mathbf{v}$ in Figure 1.2.

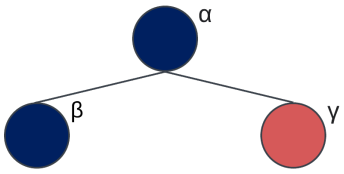

Figure 1.2 Figure depicts a simple swap game with two blue agents α and β and one red agent γ.

First, we calculate utilities based on our definition of utility function and get the following:

- $u_\alpha = \frac{1}{2}$
- $u_\beta = \frac{1}{1} = 1$
- $u_\alpha = \frac{0}{1} = 0$

Now we can immediately see value of social welfare, which is sum of all utilities – $SW(\mathbf{v}) = \frac{3}{2}$. We can also see that no agent can achieve larger utility by swapping; therefore, we are in *pure Nash swap equilibrium*. If we achieve a similar result in a jump game, we call it *pure Nash jump equilibrium*.

▶ **Definition 1.16** (Pure Nash jump equilibrium)**.** *Let* $(z, \mathbf{v}_{-i})$ *be the assignment obtained by changing the location of a strategic agent* i *from* v_i *to empty node* z*. An assignment* $\mathbf{v}$ *is a pure Nash jump equilibrium (or, simply, a jump equilibrium) if and only if for all* $i \in R$ *and for all* $z \in V$ *such that* $z \neq v_j$ *for all* $j \in R \cap S$*, we have* $u_i(\mathbf{v}) \geq u_i(z, \mathbf{v}_{-\mathbf{i}})$*.*

▶ **Definition 1.17** (Pure Nash swap equilibrium)**.** *Let* $\mathbf{v}^{i \leftrightarrow j}$ *be the assignment obtained from* $\mathbf{v}$ *by swapping position between agents* i *and* j*. An assignment* $\mathbf{v}$ *is a pure Nash swap equilibrium (or, a swap equilibrium) if and only if for all* $i, j \in R$ *we have* $u_i(\mathbf{v}) \geq u_i(\mathbf{v}^{\mathbf{i} \leftrightarrow \mathbf{j}})$ *or* $u_j(\mathbf{v}) \geq u_j(\mathbf{v}^{\mathbf{i} \leftrightarrow \mathbf{j}})$*.*

Significance of pure Nash equilibrium in real-life scenarios cannot be overstated, as its discovery has the potential to impact various aspects, for example, reduction of dissatisfied neighborhoods. Recently, Agarwal et al. [9] proved that deciding wether a game admits a pure Nash jump/swap equilibrium is NP-complete for both types of games with $k \geq 2$, where k specifies number of agent types.

▶ **Theorem 1.18** (Agarwal et al., 2021 [9]). *For every $k \geq 2$, it is NP-complete to decide whether a given k-jump game admits an equilibrium assignment, even if all strategic agents belong to the same type.*

▶ **Theorem 1.19** (Agarwal et al., 2021 [9]). *For every $k \geq 2$, it is NP-complete to decide whether a given k-swap game admits an equilibrium assignment.*

Generally speaking, we can say that finding the equilibrium for a given assignment $\mathbf{v}$ in any type of game is a NP-complete problem[2]. Further in this thesis, we do not explicitly write 'jump'/'swap' and only use equilibrium when context is clear.

[2]Later we show that maximizing social welfare with stubborn agents is also NP-complete.

Chapter 2

Maximizing social welfare

In this chapter, we focus on computing maximal social welfare of assignments which consider stubborn agents' utilities. Furthermore, we investigate the complexity of finding such maximal social welfare (MSW). Observe that set of strategic actions available to the agents does not interfere with the complexity of these problems, that is, considering jump games or swap games does not make a difference.

2.1 Considering stubborn agents

Before we show that calculating MSW is NP-complete, we would like to show why it can be beneficial to include stubborn agents in social welfare.

▶ **Lemma 2.1.** *Maximal social welfare for every $k \geq 2$, given a Schelling game with k types and $|R| = L$, could be, without considering stubborn agents' utilities, lowered at least by L, even if we consider a game with an equilibrium assignment on a tree topology.*

We will prove this Lemma 2.1 for jump games and swap games separately as proofs require different topologies.

▶ Note 2.2. Our proofs are not necessarily upper bounds of a possible increase in MSW, for example, in our swap game topology for $k = 2$, we can remove one blue strategic agent and increase MSW by $L + 0.5$.

Jump games

Proof. Given $k \geq 2$, we construct an instance with $k-1$ strategic blue agents, $k-1$ strategic agents, each of different type (not blue) and partitions $A_1, ..., A_{k-1}$ with stubborn blue agents, where $\sum_{i=1}^{k} |A_i| = L$; thus $n = 2(k-1) + L$. We denote root as α; α has k children, where $k-1$ children are parent nodes of partitions A_i. Last child of a root node α recursively contains one neighbor for $k-2$, that is, there is path of length $k-1$ connected to α. We show that there exists an assignment where MSW without stubborn agents' utilities is at least smaller than L. Figure 2.1 depicts topologies for $k = 3$.

Let us have an assignment $\mathbf{v}$, such that α is empty and parents of partitions are all of different types than blue, which means that all $k-1$ strategic blue agents are placed on a path connected to the root node α. All strategic blue agents have utility 1 (for $k > 3$) and as other agents are stubborn or a single agent of its own type, no agent wants to deviate; hence we are in equilibrium and $SW(\mathbf{v}) = k - 1$ for $k > 2$ or $SW(\mathbf{v}) = 0$ for $k = 2$.

On the contrary, we can create an assignment $\mathbf{v}$ in such a way that parents of partitions are blue strategic agents and the rest of the strategic agents are placed on a path. We can now add utility for all stubborn agents, which is $L \cdot 1$ and since the root node is empty, we also have utility 1 for each of our strategic blue agent; thus $SW(\mathbf{v}) = L + k - 1$ for $k > 3$ or $SW(\mathbf{v}) = L$ for $k = 2$.

◀

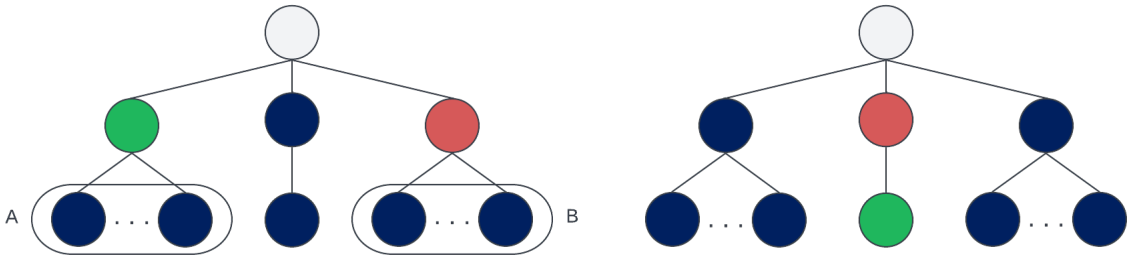

Figure 2.1 Example of the topology used in the proof of Lemma 2.1 for jump games where $k = 3$. The assignment on the left is maximal social welfare without considering stubborn agents' utilities, while the assignment on the right does consider their utilities.

Swap games

Proof. We give a proof for $k = 2$ and we will provide an extension for $k > 2$ at the end. To show that MSW can be lowered at least by L for $k = 2$, we can construct our tree topologies as follows:

- There are two agent types: red and blue.
- There are exactly 3 strategic agents, two blue and one red, rest are stubborn.
- Figure 2.2 depicts both defined topologies.

  The tree topology I_1 with $G_1 = (V, E)$ consists of a root node α, two partitions A and B a path Γ connected to α such that:

  - Partitions are occupied by stubborn blue agents and $|A| + |B| = L$.
  - Γ has length 2 and is occupied by strategic blue agents.
  - Strategic red agent is placed at α.

  We can see that MSW of I_1 is 1.5 and no agent wants to deviate as strategic red agent will not swap as she cannot increase her utility. Let us now move on to the second topology I_2 with $G_2 = (V, E)$ which is similar to I_1 except now are agents placed with considering stubborn agents utilities as follows:

  - Partitions are occupied by stubborn blue agents and $|A| + |B| = L$.
  - Γ has length 2 and is occupied by strategic red agent at leaf node.
  - Strategic blue agents are placed at the last two available nodes.

  We can see that $MSW = L + 1.5$ and still no agent wants to deviate and we can extend our proof by simply adding additional strategic agents of different types to our path Γ; hence we are able to achieve social welfare larger by L for every k which concludes our proof.

◀

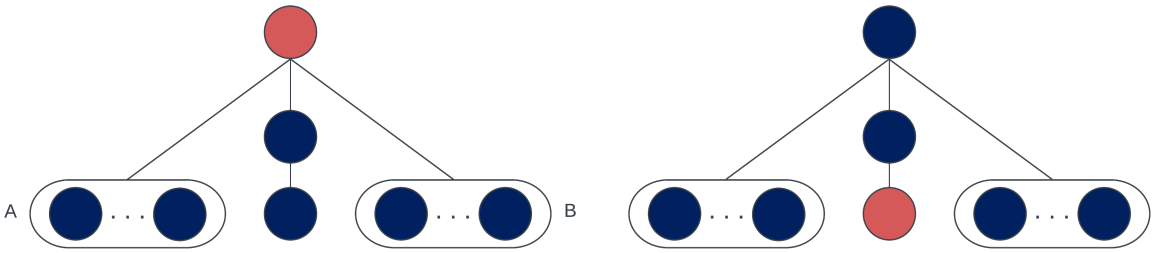

Figure 2.2 The topology used in the proof of Lemma 2.1 for swap games. The assignment on the left is maximal social welfare without considering stubborn agents' utilities, while the assignment on the right does consider their utilities.

We learned that it could be important to consider stubborn agents as it can drastically affect the outcome of social welfare. Now we can move on to showing that maximizing social welfare with considering stubborn agents is NP-complete.

2.2 NP-completeness

▶ **Theorem 2.3.** *For every $k \geq 2$, given a Schelling game with k types and a rational value ξ, it is NP-complete to decide whether the game admits an assignment with social welfare at least ξ.*

Proof. We already mentioned that this problem is in NP: given an assignment, we check whether the SW is at least ξ by summing up the utilities of all agents. We prove NP-hardness using the NP-hard variant of the BICLIQUE problem for bipartite graphs, that is, given a bipartite graph $H = (A \cup B, Y)$ and an integer s, does H contain a complete bipartite subgraph $K_{s,s}$ between A and B? This problem was proven to be NP-hard by David S. Johnson [29]. We assume that $k = 2$ (our proof can be extended for $k > 2$ by adding isolated nodes with agents of different types). Given an instance $\langle H, s \rangle$ of BICLIQUE problem where $H = (A \cup B, Y)$ we construct our social welfare maximization problem like this:

- There are two agent types: red and blue.
- There are $|A|$ strategic red and $|B|$ strategic blue agents. For simplification, we denote $n_a = |A|$ and $n_b = |B|$. All remaining agents are stubborn. We will describe the locations of stubborn agents further while defining the topology.
- The topology $G = (V, E)$ is a complement of H expanded with these vertices and edges:
  - Let α be a stubborn red agent and β be stubborn blue agent. We create edges for α and β such that for $v \in A : \{\alpha, v\} \in E$ and for all $u \in B : \{\beta, u\} \in E$.
  - Furthermore, let agent α be connected to $n_a - s$ empty nodes and similarly agent β to different $n_b - s$ empty nodes.
- Finally, let $\xi = n_a + n_b + 2$.

We will argue that our assignment $\mathbf{v}$ can only have $SW(\mathbf{v}) = \xi$ if and only if graph H contains a biclique of size s.

Let us have agents $a_1, \ldots, a_s$ from partition A and agents $b_1, \ldots, b_s$ from partition B and consider that H contains a biclique between these nodes, thus for all $b_i \in B : \exists \{b_i, a_j\} \in E$, for all $i = \{1, \ldots, s\}$ and $j = \{1, \ldots, s\}$. This means that in our complement there are no edges between these nodes. Observe that the rest of the strategic agents can obtain a maximal utility

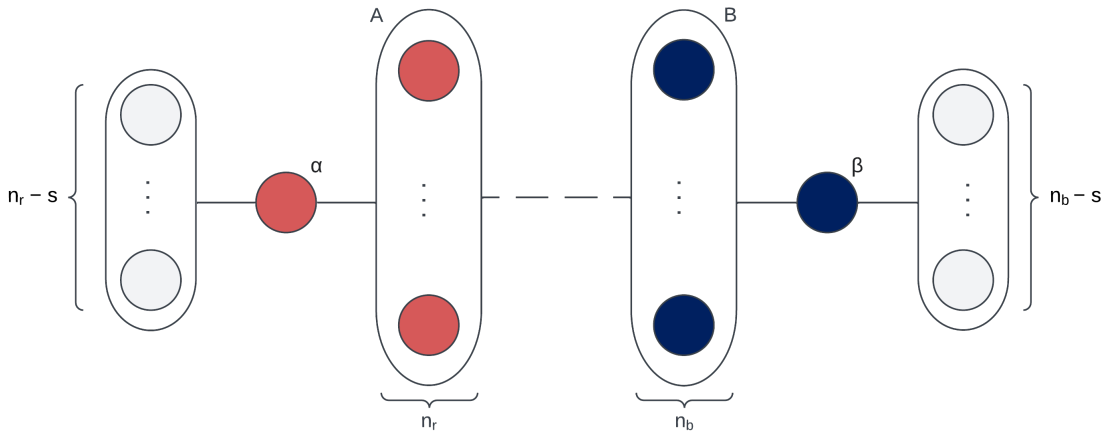

Figure 2.3 The topology G used to prove Theorem 2.3. Solid edge between two components means that all nodes from one component are connected to all nodes in the other component. Dashed edge between two components indicates that there are some edges between nodes in two components such that components form a bipartite graph.

just by moving to empty nodes connected only to stubborn agents of their type. To conclude, there are s agents in each partition, and each of these agents has utility 1. Moreover, the rest of the strategic agents are only connected to a stubborn agent of their own type, hence achieving maximal utility. When summing up the utilities of strategic agents, we get $n_r - s + n_b - s + 2s$. Finally, we add stubborn agents' utilities and achieve $SW(\mathbf{v}) = n_a + n_b + 2$.

Conversely, assume that there exists an assignment $\mathbf{v}$ with $SW(\mathbf{v}) = \xi = n_a + n_b + 2$; hence, for all agents $v \in V : u_v = 1$.

▶ **Lemma 2.4.** *Strategic agents can only be in the partition or additional empty nodes connected to a stubborn agent of the same type to achieve maximal utility.*

Proof. For the sake of contradiction, assume that there is a strategic red agent v that is not in partition A nor in additional empty nodes connected to α and has maximal utility. Hence v is either in partition B or in a empty node connected to β. By our choice of parameters, if v is in partition B it has a neighbor β, which means that v does not have a maximal utility. Additionally, if v is in any empty node η connected to β such that $\eta \notin B$ then her utility is 0. Similarly, we can formulate the proof for strategic blue agents. ◀

Based on Lemma 2.4 we know that there will be at least s agents of the same type in each partition. First, suppose that there are exactly s agents in each partition. All these agents must have utility 1, therefore, for all strategic agents $a \in A$ and $b \in B : \{a, b\} \notin E(H^C)$, which means that $\langle H, s \rangle$ has a biclique of size s. Furthermore, if each partition contains more than s agents there must be a larger biclique to ensure all agents have maximal utility. Trivially, biclique of size s is a subgraph of larger biclique. This completes our proof. ◀

Chapter 3

Maximizing social welfare at equilibrium

In this chapter, we consider the efficiency of equilibrium assignments with and without stubborn agents on k-jump games and provide a polynomial algorithm to calculate maximal social welfare at equilibrium (MSWE) on trees. We analyze how the number of maximal social welfares at equilibrium changes when taking into account stubborn agents' utilities and when disregarding them. Furthermore, we compare MSWE values with and without stubborn agents, and based on our results, we build a graph neural network that can predict MSWE without requiring additional computational costs.

3.1 Motivation

Let us assume that we are a professor at the university and are planning a trip to a nuclear power plant. In our class, there are students from different student clubs, and we know that only students from the same club consider themselves friends. We need to seat the students on a bus in a way that no one wants to change, and the happiness of our group is maximal, as we want to achieve the best possible mood for the excursion.

We know this about our trip:

- The total number of students is less than the number of seats available on the bus.
- The happiness of students depends on the number of their friends sitting around them.
- We assume that there are at least two different student clubs in this class.
- Every student has to be part of exactly one student club.
- Pierre wants to sit in seat 2, because he likes to watch a driver.
- Marie is in love with Pierre, but is afraid to sit next to him, so she will sit behind him in seat 4.
- etc.

For luck of our excursion we are an experienced theoretical computer scientist, and we see the connection between our excursion and maximizing social welfare in a Schelling game (which we had defined in Section 2.2) and we already know that finding such a seating plan is an NP-complete problem.

There is also something else in our problem: stubborn students who will only sit in one seat and will not change. We are a righteous teacher, so we will try to maximize social welfare while taking into account our stubborn students. How does the consideration of stubborn students change our final assignment? Is the MSWE going to be higher or lower? We will investigate these questions and design a polynomial algorithm to calculate our desired assignment. Note that our algorithm only works on tree topologies.

3.2 Creating algorithm

In this section, we present the modification of the equilibrium algorithm proposed by Agarwal et al. [9] to return the SW of the assignment. The modified algorithm allows us to evaluate the impact of stubborn agents on the SW and is an important step towards understanding the role of stubborn agents in Schelling games. The following subsections describe their algorithm and the details of our alternations.

3.2.1 Equilibrium algorithm

Here we will quickly go through their algorithm so that you can understand the changes we made.

▶ **Theorem 3.1** ([9]). *Given a k-jump game with tree topology, we can decide whether there exists an equilibrium (and compute one if it exists) in time poly(n^k), that is, this problem lies in the complexity class XP with respect to the number of types k.*

Proof. We consider instance I with n agents and tree topology $G = (V, E)$ and mark node r as a root. Let us have two functions on node v – *tree(v)* and *child(v)*. Function *tree(v)* denotes all descendants of v, including v, and *child(v)* returns the set of all children of v. We can label the set $U = \{i/j : i \in [n], j \in [n], i \leq j\} \cup \{\emptyset\}$ as the utility of a strategic agent.

The algorithm uses a dynamic programming approach, that is, for each node $v \in V$, the table τ_v is filled. Table τ_v contains the entry $\tau_v(C, n, k, \check{u}, \hat{u})$ for each tuple $(C, n, k, \check{u}, \hat{u})$ where:

- $C \in \{blue, red, empty\}$,
- $n = (n_b, n_r) \in [n]^2$,
- $k = (k_b, k_r) \in [n]^2$,
- $\check{u} = (\check{u}_b, \check{u}_r, \check{u}_{b_+}, \check{u}_{r_+}) \in U^4$, and
- $\hat{u} = (\hat{u_b}, \hat{u_r}, \hat{u_{top}}) \in U^3$.

We can see that the input size is polynomial, precisely $3 \cdot n^4 \cdot |U|^7$.

Furthermore, we define the set of all blue agents as T_b and likewise T_r is the set for the red agents. The entry in each table τ_v can be true or false, depending on whether these conditions are met:

1. If $C = empty$, then node v is empty. Otherwise, the node is assigned to an agent of color C.
2. Exactly n_B nodes of $tree(v)$ are assigned to blue agents, and exactly n_R nodes of $tree(v)$ are assigned to red agents.
3. Exactly k_B nodes of $child(v)$ are assigned to blue agents, and exactly k_R nodes of $child(v)$ are assigned to red agents.

4. Every blue agent in a node of $child(v)$ gets utility at least $u^{\check{}}_{B^\dagger}$ and every red agent in a node of $child(v)$ gets utility at least $u^{\check{}}_{R^\dagger}$.

5. Every blue agent at a node of $tree(v)\backslash(child(v) \cup \{v\})$ gets utility at least $\check{u}_B$ and every red agent at a node of $tree(v)\backslash(child(v) \cup \{v\})$ gets utility at least $\check{u}_R$.

6. If a blue agent that is not already in $tree(v)$ moves to an empty node of $tree(v)\backslash\{v\}$, her utility would be at most $\hat{u}_B$, and if a red agent that is not already in $tree(v)$ moves to an empty node of $tree(v)\backslash\{v\}$, her utility would be at most $\hat{u}_R$.

7. If node v is not empty, then the agent occupying v can obtain utility at most $\hat{u}_{top}$ by moving to an empty node of $tree(v)\backslash\{v\}$.

8. All agents in the nodes of $tree(v)\backslash\{v\}$ have no incentive to deviate to the empty nodes of $tree(v)\backslash\{v\}$.

The algorithm will end when we reach the root table τ_r, then our instance I of a Schelling game admits equilibrium if and only if our τ_r has $n_B = |T_B|$, $n_R = |T_R|$ and $\tau_r(C, n, k, \check{u}, \hat{u}) = true$ and these additional conditions hold:

- if C = blue, then

$$\frac{k_B}{k_B + k_R} \geq \hat{u}_{top}$$

- If C = red, then

$$\frac{k_R}{k_B + k_R} \geq \hat{u}_{top}$$

  Note that these two conditions ensure that if node r is occupied by agent v, the agent v does not want to deviate to any other node. We know move on to the game where root node is empty.

- If C = empty, then for each $X \in \{R, B\}$ with $k_x > 0$ we have

$$\frac{k_X}{k_B + k_R} \geq \check{u}_X,$$
$$\frac{k_X - 1}{k_B + k_R - 1} \leq \check{u}_{X^\dagger}$$

The last condition ensures that if C of table τ_r is empty, no agent wants to deviate there. We know that the validation of table τ_r is possible in polynomial time. It remains to prove that τ_r can be filled in polynomial time.

In every table, we write $1_B(C) = 1$ if C = blue and 0 otherwise; similarly, $1_R(C) = 1$ if C = red and 0 otherwise, and $1_E(C) = 1$ if C = empty and 0 otherwise. We start by filling in the leaf nodes. For every leaf node v, we have

$$\mathrm{T}_v(C, n, k, \check{u}, \hat{u}) = \begin{cases} true, & \text{if } (1_B(C), 1_R(C)) \text{ and } k = (0, 0) \\ false, & \text{otherwise.} \end{cases} \tag{3.1}$$

Now we will look at constructing tables for a parent of L nodes. Suppose that we have the parent node w and have constructed a table for each $v \in child(w)$. Let $|child(w)| = L$. We will construct τ_w by creating intermediate tables θ^l_w for each $l \in \{0, 1, \ldots, L\}$. This table has an entry $\theta^l_w(C, n, k, \check{u}, \hat{u})$ for each tuple and its entry is set to $true$ if and only if the subtree $tree_l(w)$,

$tree_l(w) = tree(w) \backslash \{v_{l+1}, \ldots, v_L\}$, meets our mandatory conditions 1-8. It is important to note that, by construction, $\tau_w(C, n, k, \check{u}, \hat{u}) = \theta^l_w(C, n, k, \check{u}, \hat{u})$.

We construct θ^l_w sequentially for $l = 0, \ldots, L$. Filling in the first node is done using Equation 3.1. Now we will inspect how we can fill the rest of the tables. Assume that we have completed the first tables l, that is, $\theta^0_w, \ldots, \theta^{l-1}_w$. We construct table θ^l_w by combining θ^{l-1}_w and τ_{v_l} in this way: θ^l_w = true if and only if there exist $(C', n', k', \check{u}', \hat{u}')$ and $(C'', n'', k'', \check{u}'', \hat{u}'')$ such that $\theta^{l-1}_w(C', n', k', \check{u}', \hat{u}') = \tau_{v_l}(C'', n'', k'', \check{u}'', \hat{u}'') = true$ and these conditions hold:

In the article by Agarwal et al. [9] you can find precise explanations with the conditions provided; therefore, we do not repeat their explanations.

1. $C' = C$

2. $n'' + n' = n$

3. $1_B(C'') + k'_B = k_B$ and $1_R(C'') + k'_R = k_R$

4. For each $X \in \{B, R\}$,

$$\check{u}'_{X^\dagger} \geq \check{u}_{X^\dagger}$$

Additionally, if $C'' = blue$, then

$$\frac{k''_B + 1_B(C')}{k''_B + k''_R + (1 - 1_E(C'))} \geq \check{u}_{B^\dagger}$$

and if $C'' = red$, then

$$\frac{k''_R + 1_R(C')}{k''_B + k''_R + (1 - 1_E(C'))} \geq \check{u}_{R^\dagger}$$

5. For each $X \in \{B, R\}$,

$$\check{u}'_X, \check{u}''_X, \check{u}''_{X^\dagger} \geq \check{u}_X$$

6. For each $X \in \{B, R\}$

$$\hat{u}'_X, \hat{u}''_X \leq \hat{u}_X$$

and, if $C'' = empty$, then

$$\frac{k''_X + 1_X(C')}{k''_B + k''_R + (1 - 1_E(C'))} \leq \hat{u}_X$$

7. If $C' = blue$, then

$$\hat{u}'_{top} \leq \hat{u}_{top}, \ \hat{u}''_B \leq \hat{u}_{top}$$

and, if also $C'' = empty$, then

$$\frac{k''_B}{k''_B + k''_R} \leq \hat{u}_{top}$$

Similarly, if $C' = red$, then

$$\hat{u}'_{top} \leq \hat{u}_{top},\ \hat{u}''_R \leq \hat{u}_{top}$$

and, if also $C'' = empty$, then

$$\frac{k''_B}{k''_B + k''_R} \leq \hat{u}_{top}$$

8. if $n''_B > 0$, then $\check{u}''_B,\ \check{u}''_{B\dagger} \geq \hat{u}'_B$
   If also $C'' = empty$ and $n''_B > k''_B$ then

$$\check{u}''_B \geq \frac{k''_B + 1_B(C')}{k''_B + k''_R + 1_B(C')}$$

   If also $C'' = empty$ and $k''_B > 0$ then

$$\check{u}''_{B\dagger} \geq \frac{k''_B + 1_B(C') - 1}{k''_B + k''_R + 1_B(C') - 1}$$

   Further, if $C'' = blue$ then

$$\frac{k''_B + 1_B(C')}{k''_B + k''_R + 1_B(C')} \geq \hat{u}'_B,\ \hat{u}''_{top}$$

   Similarly, the red agents must also satisfy these constraints.

9. If $n'_B > k'_B + 1_B(C')$, then $\check{u}'_B \geq \hat{u}''_B$
   If also $C'' = empty$, then

$$\check{u}'_B \geq \frac{k''_B + 1_B(C')}{k''_B + k''_R}$$

   Similarly, if $k'_B > 0$, then analogous constraints must hold with $\check{u}'_{B\dagger}$ which replaces $\check{u}'_B$. Note that these constraints must also hold for red agents.

All of these constraints can be verified in polynomial time, so this completes their proof for instances with no stubborn agents and $k = 2$. They extended their algorithm for stubborn agents by not considering different types of C and evaluating their tables as *false*. ◀

3.2.2 Modification for swap games

We can also find equilibrium assignment for swap games. We use the same algorithm as above, but we do not enumerate empty nodes, thus we can simply remove all conditions that consider them. We also add condition to check if all nodes are covered by agents, otherwise we would not have valid swap game. Notice, that in swap games we consider utilities of both agents during the swap. For example, if we have agents x and y and x does not suffice the conditions, then we can take into account agent y. If all of the conditions are true for agent y then we can still consider equation to be correct as agents in swap games only move if they both increase their utility. We can see that in the worst case we will at most check both agents in every iteration which is still running in polynomial time.

3.2.3 Social welfare at equilibrium algorithm

We already mentioned that our algorithm is a modification of the algorithm above. The main difference is how we manage τ tables and, of course, the return value; our algorithm returns social welfare at equilibrium, which can be modified to return the MSWE achievable based on Equation 3.2.

$$MSW \text{ is a maximal SW of an assignment } I_{eq} \Leftrightarrow (\forall (SW) \in I_{eq}) : \ MSW \geq SW \tag{3.2}$$

▶ **Theorem 3.2.** *Given a k-jump game with tree topology, we can decide whether there exists an equilibrium and, if it does, calculate the maximal social welfare at equilibrium achievable in* $n^{0(k)}$.

Proof. Let us start with τ tables in which they use a boolean value to determine whether the table satisfies their given conditions, which simply means that the agent does not want to deviate, therefore satisfying the essence of equilibrium. We, on the other hand, use integer values for each table which we derive by combining their utility function and values from tables of their descendants. Similarly to their algorithm, we also have only two possibilities in which tables can be evaluated, i.e., given agent v, then her τ_v value is

$$\tau_v(C, n, k, \check{u}, \hat{u}) = \begin{cases} \text{positive integer or } 0 \\ -\infty \end{cases}$$

Also important to say that they do not consider stubborn agents' utilities and deviations, meaning that any table with color that differs from the chosen root's color is defined as false. We also do not consider their deviations, but we keep their value to non -∞ if possible to maintain a clearer view at complete SW of the topology, and since it really does not matter to consider them in equilibrium, we also do not examine their deviations, but considering their utility in SW is crucial for more detailed inspection of real-world segregation. Furthermore, our root r will instead of true/false return the final value of τ_r which is combined from his descendants, hence being the total value of SW in chosen topology. Last but not least, it is important to mention how they combine tables. In their algorithm, they look for two tables that achieve true value and then proceed to the next table based on these values. We will use convention -∞ plus integer equals $-\infty$ as substitution for their true table combined with false table being false. In our algorithm, we use the convention $\frac{0}{0} = 0$.

Furthermore, we now take a more detailed look at our algorithm. Consider the table τ_r being at the root node r. The game admits SW in equilibrium if and only if our τ_r has $n_B = |T_B|$, $n_R = |T_R|$ and $\tau_r(C, n, k, \check{u}, \hat{u}) \neq -\infty$. We get MSW by going through all the SW values and taking their maximum (as stated in Equation 3.2), so our algorithm returns MSWE.

Moreover, we take a closer look at filling and combining leaves, as it is a crucial part of an algorithm. We start by filling leaf nodes; similarly to their Equation 3.1:

$$\mathrm{T}_v(C, n, k, \check{u}, \hat{u}) = \begin{cases} 0, & \text{if } (1_B(C), 1_R(C)) \text{ and } k = (0, 0) \\ -\infty, & \text{otherwise.} \end{cases}$$

Now allow us to move on to the combining of leaves. This would not differ much from their algorithm, but it is crucial to mention that after combining leaf nodes to achieve the SW value for their parent, we also need to account for their utilities, as we cannot evaluate them until we know the color of their parent, i.e.,

$$\tau_w(C, n, k, \check{u}, \hat{u}) = \textit{combined values} \ + \ u(\textit{child}(w))$$

Combined values is sum of all the values from his children's tables. We calculate the utility of each child's node v as:

- If $\tau_v(C) = \tau_w(C)$, this applies,

$$u(v) = \frac{k_C + 1}{k_B + k_R + 1}$$

- If $\tau_v(C) \neq \tau_w(C)$ and $\tau_w(C) \neq empty$, then

$$u(v) = \frac{k_C}{k_B + k_R + 1}$$

- If $\tau_w(C) = empty$, we have

$$u(v) = \frac{k_C}{k_B + k_R}$$

Importantly, when we end up in the root, we repeat calculating his children's utilities, and to them we need to add our root's utility as well.

To sum it up, our algorithm runs in polynomial time, as all our added conditions can be verified in polynomial time, and the polynomial-time complexity of the core algorithm has also been proven. Extending the algorithm for k types is also trivial, and the time complexity would increase exponentially with respect to k. ◀

3.3 Analyzing results

In the previous section, we defined an algorithm for determining MSWE on tree topology. In this section, we use our algorithm to compare the results of Schelling games that consider the utilities of stubborn agents and those that do not. Furthermore, we examine the number of equilibriums obtained and investigate how the topologies with and without stubborn agents vary from one another.

▶ Note 3.3. To ensure the validity of our results, we focus our analysis on jump games with only one empty spot, because adding additional empty spots would not only increase the complexity of finding an equilibrium, but could also lead to the formation of fully separated groups. This would very seriously affect our results. On the other hand, if we only consider one empty spot, the agents are forced to form a topology in the most undesirable environment, providing us with a clearer understanding of the MSWE.

▶ Note 3.4. It is important to note that we use a fixed number of stubborn agents in our graphs. This is because a random number of agents would bring non-linearity to our data, making them relatively random and hard to spot differences between games with and without stubborn agents' utilities.

3.3.1 Comparing social welfare

First, we examine the impact of considering the utility of stubborn agents on the MSWE value. It is clear that the MSWE with stubborn agents' utilities can only be equal to or higher than the MSWE without considering these utilities, and on games with only two types of agents we expect them to be almost always higher. You can see that our results, as shown in Figure 3.1, indeed confirm that the MSWE is higher when stubborn agents' utilities are taken into account.

Before diving deeper into the results, it is crucial to understand why the minimal value of MSWE with stubborn agents' utilities is the same as the MSWE of topology without stubborn agents' utilities.

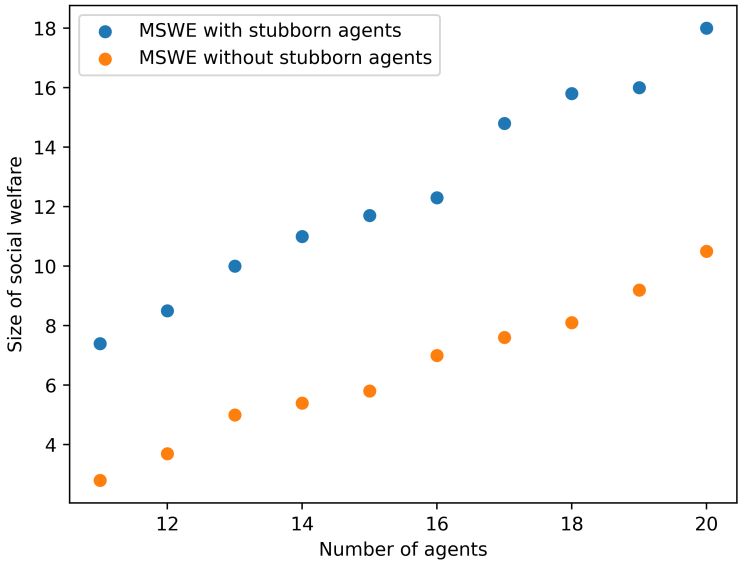

Figure 3.1 Graph showing the number of nodes in random topologies with one empty spot and four stubborn agents against the maximal size of social welfare for games that consider stubborn agents and games that do not consider them.

▶ **Lemma 3.5.** *Maximal social welfare at equilibrium with stubborn agents' utilities can not be less than maximal social welfare at equilibrium without considering stubborn agents' utilities.*

Proof. Suppose, for the sake of contradiction, that there exists a Schelling game I_1 on an assignment $\mathbf{v}$ that considers the utilities of stubborn agents and has a lower MSWE than the Schelling game I_2, also on assignment $\mathbf{v}$, that does not consider them. However, we know that the agents' utility is minimally zero, which implies that in our game I_1 stubborn agents have negative utility. This contradicts our definition of utility function, which states that utility is greater than or equal to zero. ◀

Furthermore, we observe a pattern in Figure 3.1. Based on our observations of the data, we can make the following assumption:

▶ **Assumption 3.6.** *We are able to estimate the maximal social welfare at equilibrium with a simple linear regression model. This assumption was made after analyzing numerous models that showed a similar pattern. Our confidence in this assumption is backed up by the Strong Law of Large Numbers [30], since every measurement was taken from a sample of thousand different examples.*

▶ Note 3.7. Note that our assumption only applies to games with fixed stubborn agents across all topologies. This constraint should be kept in mind when interpreting the results of our analysis and considering the broader implications of our findings.

3.3.2 Comparing number of equilibriums

The number of stubborn agents can significantly affect the equilibrium. This is evident in Figure 3.2, which shows that the topology with two stubborn agents has, on average, more equilibriums compared to the topology with five stubborn agents, and this trend is continuous

as we increase the number of stubborn agents. It is also noteworthy that the number of equilibriums that maximize SW is quite limited, and this number is even smaller when the utility of stubborn agents is considered. Interestingly, the number of equilibriums that maximize SW remains relatively constant, regardless of the size of our topology.

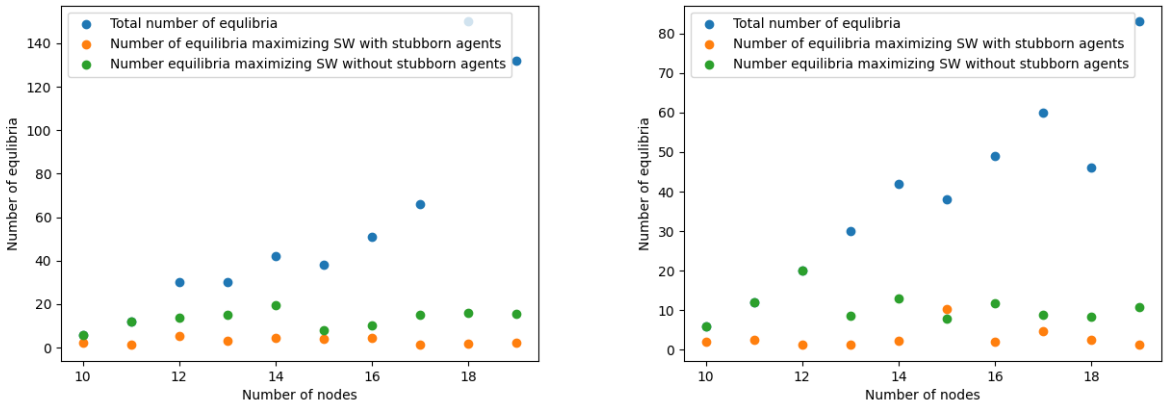

Figure 3.2 Graphs showing the number of equilibriums in topologies with one empty spot. The graph on the left shows random topologies with two stubborn agents and the graph on the right shows random topologies with five stubborn agents.

3.3.3 Comparing topologies

So far we can conclude that Schelling games that consider stubborn agents' utilities profit from higher MSW and due to that have lower total number of assignments in MSWE. To understand the reasons behind this observation, we analyze both assignments (with and without stubborn agents' utilities) and compare their agent layouts.

Our analysis revealed two major problems that stubborn agents encounter in topologies that do not take them into account:

- Unwanted coalition problem
- Cutting off problem

We define these problems as follows:

▶ **Definition 3.8** (Unwanted coalition)**.** *An assignment* $\mathbf{v}$ *with stubborn agent* β *contains Unwanted coalition, if and only if, there is a strategic agent* α *with different type than* β *in* $N_\beta(\mathbf{v})$*, such that there exist an assignment* $\mathbf{v}'$*, where* $\alpha \notin N_\beta(\mathbf{v}')$ *and* $u_\alpha(\mathbf{v}') = u_\alpha(\mathbf{v})$*.*

Unwanted coalition problem is shown in Figure 3.3. We can see that the strategic agent may be positioned next to a stubborn agent of a different color, even though she could be placed at different location without disrupting its stubborn agents' neighborhood. This can lead, in larger topologies, to heterogeneous segregation, which swerves from the typical structure observed in real-world scenarios.

▶ **Definition 3.9** (Cutting off problem)**.** *An assignment* $\mathbf{v}$ *with stubborn agent* β *contains Cutting off problem, if and only if, there are strategic agents* α *and* γ *with same type as* β *and there exist empty node between* α *and* β *such that there exist an assignment* $\mathbf{v}'$*, where* $\gamma \in N_\alpha(\mathbf{v}') \ \wedge \ \gamma \in N_\beta(\mathbf{v}')$ *and* $u_\alpha(\mathbf{v}') = u_\alpha(\mathbf{v})$*.*

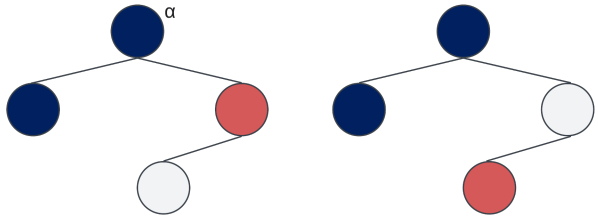

Figure 3.3 The topology of the 2-jump game created as an example of the Unwanted coalition problem from Definition 3.8 with stubborn agent α. The assignment on the left does not consider α's utility, and the assignment on the right does.

The Cutting off problem is shown in Figure 3.4, you can see that stubborn agent α would rather be connected to root node via strategic agent, but as her utility is not consider he could be "left out" from homogeneous group. On larger topologies with more stubborn agents this could result as highly non-profitable, as because of this we may be seeing segregation between same types.

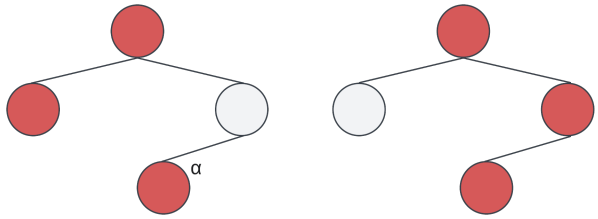

Figure 3.4 The topology of the 2-jump game created as an example for the Cutting off problem from Definition 3.9 with stubborn agent α. The assignment on the left does not consider α's utility, and the assignment on the right does.

To conclude our analysis, we investigated assignments in MSWE. We saw that assignments that consider stubborn agents' utilities not only profit from higher SW, but they can also have a better structure properties. Moreover, it is evident that we can implement a polynomial algorithm to solve both the Unwanted Coalition and Cutting Off problem. However, we have found that applying trivial algorithms to change agents positions may result in isomorphic assignments. One can also see that we may drastically lower SW if we do not consider it with agents' deviations.

Chapter 4

Graph neural networks

In this chapter, we explore the efficiency of graph neural networks (GNNs) in classifying if equilibrium assignments are MSWE. We build GNN and test its accuracy for tree and general graphs. We aim to investigate the differences in accuracies achieved by GNN and to see how these accuracies are impacted by the used topology. For gentle introduction to GNNs, we recommend a monograph of Wu et al. [20].

4.1 Graph classification with GNNs

Graph classification is the task of classifying whole graphs using structural graph properties from a graph dataset. The goal is to embed entire graphs in such a way that they are linearly separable, see Figure 4.1. Graph classification is mostly used for molecular property prediction, but can also be used in settings quite similar to ours, for example, classification of social networks [21].

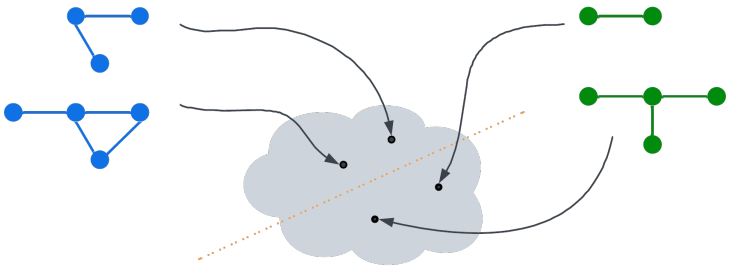

Figure 4.1 Example of embedding two types of graphs such that they are linearly separable.

4.1.1 Mini-batching of graphs

Graph classification datasets typically consist of small graphs, and our datasets are no exception. To improve performance, we use batching, which guarantees full GPU utilization [31]. Unlike traditional batching of images or text data, which typically involves *rescaling* or *padding* the examples to achieve equal-sized shapes [32]. These examples are then grouped into an additional dimension. We typically refer to the length of this dimension as a *batch-size*.

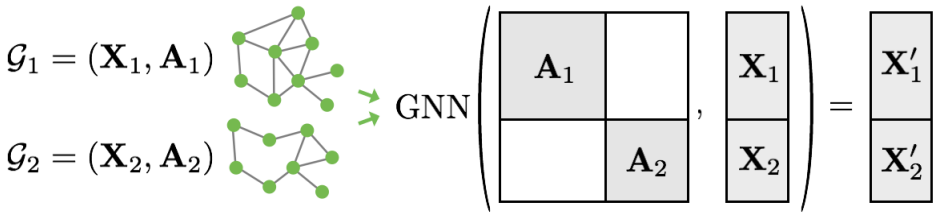

Figure 4.2 Mini-batching in graph neural networks.

However, these approaches are not feasible, or, in case of padding, may result in significantly more memory consumption. Therefore, most of the libraries use different approaches to accomplish parallelization across multiple examples. Common trick is to diagonally put adjacency matrices into one giant sparse matrix, that is, we create a giant graph with multiple components, see Figure 4.2[1]. Notice that adjacency matrices are stored sparsely, thus there is no computational or memory overhead.

4.1.2 Training GNN for graph classification

We will not go into detail of training GNN, but graph classification is mostly done as follows:

- Use message passing to embed each node.
- Aggregate embedded nodes into our graph embedding (**readout layer**).
- Train a final model on the graph embedding.

There exist multiple different types of readout layers in the literature. One of the most common readout layer takes the average of node embeddings:

$$\mathbf{x}_G = \frac{1}{|V|} \sum_{v \in V} x_v^{(l)}$$

We will be using a more complex readout layer that adds a skip connection to the GNN layer to preserve the central node information [33]:

$$\mathbf{x}_v^{(l+1)} = \mathbf{W}_1^{(l+1)} \mathbf{x}_v^{(l)} + \mathbf{W}_2^{(l+1)} \sum_{w \in N(v)} \mathbf{x}_w^{(l)}$$

where:

$\mathbf{x}_v^{(l)}$ = feature vector of node v at the l-th layer of GNN
$\mathbf{W}_1^{(l+1)}$ = weight matrix for feature vector at the l-th layer
$\mathbf{W}_2^{(l+1)}$ = weight matrix for feature vector's neighbors at the l-th layer
$N(v)$ = set of neighbors of v

We used this readout layer rather than using simple readout layer that only takes averages of node embeddings, as we found that this approach leads to faster convergence and higher accuracies.

▶ Note 4.1. We only studied these two readout layers as they are both directly implemented in Pytorch Geometrics library [34].

[1] GOOGLE. *Google colaboratory* [online]. 2021 [cit. 2022-11-29]. Available from: Google colaboratory.

In summary, we introduced the fundamentals of GNNs and showed how we can use them for graph classification. Furthermore, we have discussed graph batching, which is used to improve GPU utilization and can be beneficial for the further usage of our model on much larger graphs. Lastly, we presented the usual steps in training GNN, demonstrated the application of the readout layer, and described the readout layer used in our model.

4.2 Algorithm for general graphs

Before implementing our GNN, we need to create an algorithm that finds MSW for general graphs, so we can generate datasets to train on. As there is no efficient algorithm to decide MSW we will use the brute-force method on small topologies.

Algorithm 1 Compute Equilibrium Assignments

Input: Graph G, Assignment v
Output: *result* – list of equilibrium assignments for G

```
assignments ← permutate(v)                    ▷ Calculate distinct permutations of v
for a in assignments do                       ▷ Iterate through all possible assignments
    G ← a
    isEquilibrium ← True
    for agent in G[strategic agents] do
        for empty in G[empty] do
            u1 ← agent.utility
            swap(agent, empty, G)             ▷ Swaps nodes in G
            u2 ← empty.utility
            swap(agent, empty, G)
            if u1 < u2 then                   ▷ Check if agent does not want to deviate
                isEquilibrium ← False
                break
            end if
        end for
        if isEquilibrium == False then
            break
        end if
    end for
    if isEquilibrium == True then
        result ← a                            ▷ Save equilibrium assignment
    end if
end for
return result
```

We will now prove that our algorithm runs in $O(n^2 \cdot n!)$ and will always return correct result.

▶ **Lemma 4.2.** *Algorithm 1 runs in $O(n^2 \cdot n!)$ with space complexity $O(n!)$ and will always return correct result.*

Proof. The proposed algorithm for finding the equilibrium assignments in a general graph exhaustively goes through all possible deviations for each agent and checks whether no agent wants to deviate. By definition, an equilibrium is reached when no agent can improve her utility. Therefore, the algorithm guarantees to find equilibrium assignments for given G. The worst-case time complexity of the algorithm is $O(n^2 \cdot n!)$ as it generates all possible assignments and checks in each of them whether or not every agent wants to deviate. We need to store all possible assignments which gives us space complexity $O(n!)$. ◀

To extend our algorithm to find MSWE assignments, we can go through the results of our algorithm and select assignments with highest SW[2]. Let us now move to our main experiment.

4.3 Experiment

This section outlines the experimental setup for evaluating the accuracy of GNN on classifying tree and general graphs, that is, we describe used datasets, features and our choice of hyperparameters. We implement our GNN model using the Pytorch Geometrics library by Fey and Lenssen [34]. In addition to this model, we implement a *validator* that decides whether our prediction was correct for a given assignment.

$$\text{Model outputs} = \begin{cases} 1, & \text{MSWE assignment} \\ 0, & \text{otherwise.} \end{cases}$$

Datasets

All graph datasets are in enclosed zip-file and are divided, based on their structure, into two folders. Each of these datasets consists of MSWE assignments and equilibrium assignments that do not have MSW. Each dataset contains the same number of both assignments. We report dataset statistics in Table 4.1. Both datasets have the same amount of graphs to train on and were generated from the same distribution to achieve similar datasets for our model.

Dataset type	# Graphs	# Classes	# Nodes	# Edges	# Node labels
Tree graphs	1150	2	12.16	11.16	6
General graphs	1150	2	12.49	39.33	6

Table 4.1 Information about used datasets.

Notice that our graphs are quite small. This is because to generate a dataset of general graphs, we use a very inefficient brute-force algorithm and as we want to compare accuracies, we limit our tree topologies to similar sizes.

Features

In GNN literature, we can often see an augmentation of node description with additional structural features to improve performance. We also add additional features to our nodes in addition to the agent's *color*. We provide a list of our features and their values in Table 4.2.

Importantly, as we want to compare our topologies consistently, we use the same features in both of our datasets.

	Color	Utility	# Red neighbors	# Blue neighbors	Degree	Stubborn
Value	String	Float	Integer	Integer	Integer	Boolean

Table 4.2 Description of features and their types for each node.

[2]The Algorithm 1 is presented for jump-games, but can be also modified for swap games by going through all agents and calculating their swap values; we omit the details of this implementation.

Hyper-parameters

We perform hyper-parameter tuning via grid search. Furthermore, we select the number of convolutional layers, the number of linear layers, and the learning rate for the optimizer based on either the test accuracy or test loss.

Computational consideration

Our experiment is most complex in generating datasets of general graphs, because we use our brute-force algorithm with $O(n^2 \cdot n!)$ time complexity. Because both of our datasets consist only of 1150 small graphs and we use grid search on basic hyper-parameters, our model is trained within minutes only using CPU. We empathize that for more complex training, there could be need to use parallelism on both CPU and GPU, to conduct the experiment in acceptable amount of time.

Validator

In addition to the implementation of a GNN, we have also implemented a simple validator that verifies the output of our model. Validator either outputs *True*, if our prediction was correct, or returns *False*. In case of an incorrect classification, we provide a potential correct topology as part of the output.

4.4 Comparing predicted accuracies

We now move on to compare our model performance on both datasets, which were both trained for 200 epochs. We provide results of our model in Table 4.3. We can see that our model performs better on tree graphs. This is because tree graphs have specific characteristics that make them not only more efficient for deterministic algorithms, but, as we can see, also more efficient for probabilistic models such as GNNs.

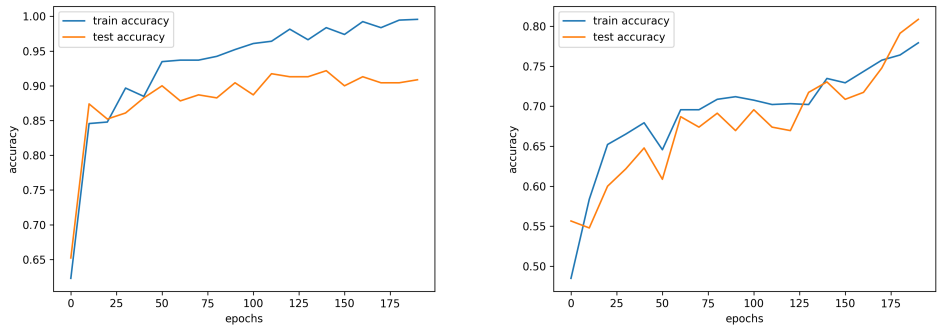

Figure 4.3 Graphs of train and test accuracies achieved by GNN on a dataset of tree graphs and a dataset of general graphs. Performance of GNN on tree graphs is shown in the graph on the left, and performance on general graphs is depicted in the graph on the right.

Furthermore, we compare training accuracy and test accuracy in Figure 4.3. We can see that our GNN not only achieves better accuracies, but also converges faster on tree topologies than on general graphs. This behavior is also caused by the simplicity of tree graphs. Notice that as we converge a lot slower on general graphs, we may be able to increase model accuracy by increasing number of epochs. Unfortunately, even when we doubled the training epochs for general graphs,

Dataset	Accuracy
Tree graphs	0.91 ± 2.6
General graphs	0.82 ± 3.0

Table 4.3 Results of our model on tree and general graphs with mean accuracy and standard deviation.

we were unable to achieve accuracy higher than 0.85 ± 2.4 before overfitting. We know move on to the last section, where we in detail describe appended implementation.

4.5 Implementation details

We used Python to program all our codes and Jupyter notebooks to display our results[3]. We used additional packages for graph representation, visualization and building GNN[4]. All of required packages can be installed by running `install.sh` file[5]. Our implementation can be divided into two main parts:

- Graph neural network
- Algorithms calculating MSWE

4.5.1 Algorithms

We have implemented algorithms to calculate MSWE for both tree topologies and general topologies. The basis of our implementation is the class `Topology`. `Topology` is used to store information about assignment passed to algorithms for calculating MSWE. Users can either create their own graph or they can use available methods in the `Topology` class to generate a random topology. Users can create `Topology` class as follows:

```
topology = Topology(#empty nodes, #blue agents, #red agents, graph, #types)
```

We can then find MSWE for given topology by running either `brute_force(topology)` or `tree_algorithm(topology)`. Furthermore, we implemented additional features for the `Topology` class, such as:

```
Topology.show()   // shows graph with given assignment
Topology.create_general_graph()   // creates random general graph
Topology.create_tree_graph()   // creates random tree graph
```

It is worth noting that generator functions in our `Topology` class also create a random assignment for created graph. Having random assignment on a given topology does not affect our MSWE algorithms, but we can use it to create datasets for our GNN or to test the accuracy of our trained model. This makes our class more compact and easier to use not only for algorithms, but also for GNNs.

[3] All our codes can be find on gitlab.fit.cvut.cz

[4] `https://matplotlib.org/stable/index.html`
`https://more-itertools.readthedocs.io/en/stable/`
`https://networkx.org/documentation/stable/index.html`
`https://numpy.org/doc/stable/index.html`
`https://pytorch-geometric.readthedocs.io/en/latest/`
`https://pytorch.org/docs/stable/index.html`
`https://treelib.readthedocs.io/en/latest/treelib.html`

[5] We have also added `uninstall.sh` to remove all packages.

4.5.2 Jupyter notebooks

Reviewing code can be a difficult process that can sometimes lead to not properly understanding the implementation. To simplify this process, we have created three Jupyter notebooks that summarize and test our algorithms. These notebooks provide an easy-to-follow interface for exploring our code and understanding how it works.

algorithms.ipynb

In this notebook, we look at our polynomial time algorithm and provide small topology to analyze it. Furthermore, we demonstrate achieved results in our thesis using polynomial algorithm on trees. Finally, we show that general graphs behave similarly to tree graphs using brute-force algorithm.

gnn_training.ipynb

Notebook contains a simple dataloader and the `GCN` class that specifies our GNN. We describe each step using a markdown cell for a detailed description of our training. Users can use this notebook to create their own model, tune hyperparameters, etc.

gnn_testing.ipynb

We have created this notebook to enable users to directly interact with our trained GNN. We provide users with multiple simple topologies and analyze the correctness of our GNN's predictions on them. Additionally, we provide a converter from the `networkx` library to the data object that can be processed by our GNN[6]. This enables users to test our model on their own assignments and examine its performance on various graphs. Lastly, we include the validator we have previously mentioned to confirm whether the GNN's predictions were correct.

[6]This converter can be modified for any GNN.

Chapter 5

Conclusion and open problems

In this thesis, we studied games on undirected graphs inspired by Schelling's segregation model. In these games, we partition agents into multiple types and place them on nodes of a graph topology. Agents can improve their utility by jumping to empty nodes or swapping with other agents. We focused on social welfare, maximizing social welfare, and questions related to maximal social welfare at equilibrium. Additionally, we created an effective classification model that determines whether an assignment is in maximal social welfare at equilibrium. We also propose possible topics for further study.

Concerning social welfare, we proved in Theorem 2.3 that even when considering stubborn agents it is NP-complete to maximize social welfare. We showed in Lemma 2.1 that we can significantly impact social welfare by not considering stubborn agents' utilities even in simple topologies such as trees. Moreover, we analyzed potential problems in games that do not consider stubborn agents' utilities. An open question is to design efficient algorithms to detect these problems. Alternatively, one could aim to further investigate the exact lower and upper bounds of social welfare caused by ignoring stubborn agents' utilities.

For efficiently classifying assignments with maximal social welfare at equilibrium, we trained a graph neural network and reported the accuracies achieved in Table 4.3. For further work, it remains to train a more efficient model; for example, a model that can construct assignment in a way that allows for efficient computation of equilibrium or maximal social welfare as both of these problems remain to be NP-complete. Additionally, one can also investigate our model's performance on other types of graphs.

Bibliography

1. ESRI DEMOGRAPHICS TEAM. *Race and Ethnicity in the US by Dot Density (Census 2020)* [online]. Esri Demographics Team: 2021 [cit. 2022-11-09]. Available also from: https://www.arcgis.com/home/item.html?id=30d2e10d4d694b3eb4dc4d2e58dbb5a5.
2. SCHELLING, Thomas C. Dynamic models of segregation. *The Journal of Mathematical Sociology* [online]. 1971, vol. 1, no. 2, 143–186 [cit. 2022-06-19]. ISSN 0022-250X. Available from DOI: 10.1080/0022250X.1971.9989794.
3. SCHELLING, Thomas C. Models of segregation. *The American Economic Review,* [online]. 1969, vol. 59, no. 2, 488–493 [cit. 2022-06-19]. Available also from: https://www.jstor.org/stable/1823701.
4. HART, Vi; CASE, Nicky. *Parable of the polygons* [online]. 2016 [cit. 2022-07-07]. Available also from: http://ncase.me/polygons.
5. BARMPALIAS, George; ELWES, Richard; LEWIS-PYE, Andrew. Digital morphogenesis via Schelling segregation. *Nonlinearity* [online]. 2018, vol. 31, no. 4, 1593 [cit. 2022-06–28]. ISSN 0272-5428. Available from DOI: 10.1088/1361-6544/aaa493.
6. PANCS, Romans; VRIEND, Nicolaas J. Schelling's spatial proximity model of segregation revisited. *Journal of Public Economics* [online]. 2007, vol. 91, no. 1-2, 1–24 [cit. 2022-07-19]. ISSN 0047-2727. Available from DOI: 10.1016/j.jpubeco.2006.03.008.
7. CHAUHAN, Ankit; LENZNER, Pascal; MOLITOR, Louise. Schelling segregation with strategic agents. In: *Proceedings of the 11th International Symposium on Algorithmic Game Theory, SAGT 2018* [online]. 2018, 137–149 [cit. 2022-10-27]. ISBN 978-3-319-99659-2. Available from DOI: 10.1007/978-3-319-99660-8_13.
8. KANELLOPOULOS, Panagiotis; KYROPOULOU, Maria; VOUDOURIS, Alexandros A. Modified schelling games. *Theoretical Computer Science* [online]. 2021, vol. 880, 1–19 [cit. 2022-09-17]. ISSN 0304-3975. Available from DOI: 10.1016/j.tcs.2021.05.032.
9. AGARWAL, Aishwarya; ELKIND, Edith; GAN, Jiarui; IGARASHI, Ayumi; SUKSOMPONG, Warut; VOUDOURIS, Alexandros A. Schelling games on graphs. *Artificial Intelligence* [online]. 2021, vol. 301 [cit. 2022-09-17]. ISSN 0004-3702. Available from DOI: 10.1016/j.artint.2021.103576.
10. EASLEY, David; KLEINBERG, Jon. *Networks, crowds, and markets: Reasoning about a highly connected world*. New York: Cambridge University Press, 2010. ISBN 0-5211-9533-0.
11. BRANDT, Christina; IMMORLICA, Nicole; KAMATH, Gautam; KLEINBERG, Robert. An analysis of one-dimensional Schelling segregation. In: *Proceedings of the 44th Annual ACM Symposium on Theory of Computing, STOC 2012* [online]. 2012, 789–804 [cit. 2022-08-15]. ISBN 9781450312455. Available from DOI: 10.1145/2213977.2214048.

12. IMMORLICA, Nicole; KLEINBERGT, Robert; LUCIER, Brendan; ZADOMIGHADDAM, Morteza. Exponential segregation in a two-dimensional schelling model with tolerant individuals. In: *Proceedings of the 28th annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017* [online]. 2017, 984–993 [cit. 2022-08-15]. ISBN 978-1-61197-478-2. Available from DOI: 10.1137/1.9781611974782.62.

13. ECHZELL, Hagen; FRIEDRICH, Tobias; LENZNER, Pascal; MOLITOR, Louise; PAPPIK, Marcus; SCHÖNE, Friedrich; SOMMER, Fabian; STANGL, David. *Proceedings of the 15th International Conference on Web and Internet Economics, WINE 2019*. Convergence and hardness of strategic Schelling segregation. Cham: Springer International Publishing, 2019. ISBN 978-3-030-35388-9.

14. BILÒ, Davide; BILÒ, Vittorio; LENZNER, Pascal; MOLITOR, Louise. Tolerance is Necessary for Stability: Single-Peaked Swap Schelling Games. *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022* [online]. 2022, 81-87 [cit. 2023-01–17]. ISBN 978-1-956792-00-3. Available from DOI: 10.24963/ijcai.2022/12.

15. FRIEDRICH, Tobias; LENZNER, Pascal; MOLITOR, Louise; SEIFERT, Lars. Single-Peaked Jump Schelling Games. In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023* [online]. 2023, 2899–2901 [cit. 2023-05-15]. ISBN 978-1-4503-9432-1. Available from DOI: 10.48550/arXiv.2302.12107.

16. BILÒ, Davide; BILÒ, Vittorio; DÖRING, Michelle; LENZNER, Pascal; MOLITOR, Louise; SCHMIDT, Jonas. Schelling Games with Continuous Types [article in press]. 2023. Available from DOI: 10.48550/arXiv.2305.06819.

17. AZIZ, Haris; BRANDL, Florian; BRANDT, Felix; HARRENSTEIN, Paul; OLSEN, Martin; PETERS, Dominik. Fractional Hedonic Games. *ACM Transactions on Economics and Computation (TEAC)* [online]. 2019, vol. 7, no. 2, 1–29 [cit. 2023-02-11]. ISSN 2167-8375. Available from DOI: 10.1145/3327970.

18. BILÒ, Vittorio; MONACO, Gianpiero; MOSCARDELLI, Luca. Hedonic Games with Fixed-Size Coalitions. In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022* [online]. 2022, vol. 36, 9287–9295 [cit. 2022-12-11]. No. 9. ISSN 2374-3468. Available from DOI: 10.1609/aaai.v36i9.21156.

19. GANIAN, Robert; HAMM, Thekla; KNOP, Dušan; SCHIERREICH, Šimon; SUCHÝ, Ondřej. Hedonic Diversity Games: A Complexity Picture with More than Two Colors. In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022* [online]. 2022, vol. 36, 5034–5042 [cit. 2023-06-03]. No. 5. ISSN 2374-3468. Available from DOI: 10.1609/aaai.v36i5.20435.

20. WU, Lingfei; CUI, Peng; PEI, Jian; ZHAO, Liang; GUO, Xiaojie. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, 2022. ISBN 978-981-16-6053-5.

21. ZHOU, Jie; CUI, Ganqu; HU, Shengding; ZHANG, Zhengyan; YANG, Cheng; LIU, Zhiyuan; WANG, Lifeng; LI, Changcheng; SUN, Maosong. Graph neural networks: A review of methods and applications. *AI open* [online]. 2020, vol. 1, 57–81 [cit. 2022-12-11]. ISSN 2666-6510. Available from DOI: 10.1016/j.aiopen.2021.01.001.

22. SCARSELLI, Franco; GORI, Marco; TSOI, Ah Chung; HAGENBUCHNER, Markus; MONFARDINI, Gabriele. The graph neural network model. *IEEE Transactions on Neural Networks* [online]. 2008, vol. 20, no. 1, 61–80 [cit. 2022-12-11]. ISSN 1045-9227. Available from DOI: 10.1109/TNN.2008.2005605.

23. DIESTEL, Reinhard. *Graph theory, Graduate texts in mathematics*. 5th ed. New York: Springer International Publishing, 2017. ISBN 978-3-662-53621-6.

24. TURING, Alan Mathison et al. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* [online]. 1937, vol. s2-42, no. 1, 230-265 [cit. 2023-02–26]. ISSN 0024-6115. Available from DOI: 10.1112/plms/s2-42.1.230.

25. ARORA, Sanjeev; BARAK, Boaz. *Computational complexity: A Modern Approach*. 1st ed. USA: Cambridge University Press, 2009. ISBN 978-0-521-42426-4.

26. DAVENDRA, Donald. *Traveling salesman problem: Theory and applications*. InTech, 2010 [cit. 23-01-03]. ISBN 978-953-307-426-9. Available from DOI: 10.5772/547.

27. KELLERER, Hans; PFERSCHY, Ulrich; PISINGER, David; KELLERER, Hans; PFERSCHY, Ulrich; PISINGER, David. *Knapsack Problems*. 1st ed. Berlin: Springer International Publishing, 2004. ISBN 978-3-540-40286-2.

28. PARDALOS, Panos M.; JUE, Xue. The maximum clique problem. *Journal of Global Optimization* [online]. 1994, vol. 4, no. 3, 301-328 [cit. 2023-03–20]. ISSN 0925-5001. Available from DOI: 10.1007/BF01098364.

29. JOHNSON, David S. The NP-completeness column: an ongoing guide. *Journal of Algorithms* [online]. 1984, vol. 5, no. 2, 284–299 [cit. 2023-01-13]. ISSN 0196-6774. Available from DOI: 10.1016/0196-6774(84)90032-4.

30. ETEMADI, Nasrollah. An elementary proof of the strong law of large numbers. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* [online]. 1981, vol. 55, no. 1, 119–122 [cit. 2022-08-07]. ISSN 0044-3719. Available from DOI: 10.1007/BF01013465.

31. ZENG, Hanqing; ZHOU, Hongkuan; SRIVASTAVA, Ajitesh; KANNAN, Rajgopal; PRASANNA, Viktor. Accurate, efficient and scalable training of Graph Neural Networks. *Journal of Parallel and Distributed Computing* [online]. 2021, vol. 147, 166–183 [cit. 2022-12-20]. ISSN 0743-7315. Available from DOI: 10.1016/j.jpdc.2020.08.011.

32. RAWAT, Waseem; WANG, Zenghui. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation* [online]. 2017, vol. 29, no. 9, 2352–2449 [cit. 2023-02-14]. ISSN 0899-7667. Available from DOI: 10.1162/neco_a_00990.

33. MORRIS, Christopher; RITZERT, Martin; FEY, Matthias; HAMILTON, William L; LENSSEN, Jan Eric; RATTAN, Gaurav; GROHE, Martin. Weisfeiler and Leman Go Neural: Higher-order graph neural networks. In: *Proceedings of the 33rd AAAI conference on artificial intelligence, AAAI 2019* [online]. 2019, vol. 33, 4602–4609 [cit. 2022-12-11]. No. 01. ISSN 2374-3468. Available from DOI: 10.1609/aaai.v33i01.33014602.

34. FEY, Matthias; LENSSEN, Jan E. Fast Graph Representation Learning with PyTorch Geometric. In: *Proceedings of the 7th ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019 [cit. 2023-02-16]. Available from DOI: 10.48550/arXiv.1903.02428.

Contents of enclosed zip-file

- `readme.txt` ..... brief description of the media content and its usage
- `src`
  - `gnn_training.ipynb` ..... jupyter notebook for training GNN
  - `gnn_testing.ipynb` ..... jupyter notebook for testing GNN
  - `algorithms.ipynb` ..... jupyter notebook comparing and testing MSWE algorithms
  - `models` ..... trained machine learning models
  - `datasets` ..... datasets for graph neural network
    - `trees` ..... tree graphs
    - `general` ..... general graphs
  - `functions` ..... source codes of additional functions
  - `requirements` ..... bash files to install/uninstall required packages
- `thesis.pdf` ..... thesis text in PDF format