# The Computational Complexity of Diplomacy

**MSc Thesis** *(Afstudeerscriptie)*

written by

**Wouter Vromen**
(born August 10th, 1996 in Leiderdorp, The Netherlands)

under the supervision of **Dr Ronald de Haan**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

**MSc in Logic**

at the *Universiteit van Amsterdam*.

| **Date of the public defense:** | **Members of the Thesis Committee:** |
|---|---|
| *August 29, 2024* | Dr Ekaterina Shutova (chair) |
| | Dr Ronald de Haan (supervisor) |
| | Dr Gregor Behnke |
| | Dr Malvin Gattinger |



INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

# Abstract

In this thesis, we will provide a generalisation for the 1959 board game Diplomacy, remaining close to the original rules of the game. We then formulate some decision problems based on this generalisation, and determine their complexity class.

While the database of generalised games for the purposes of complexity theoretic research is vast, it did not contain a version of generalised Diplomacy yet. Meanwhile the game has attracted the attention of various other fields of research, such as artificial intelligence and automated negotiation.

Further contributions in this thesis are a mathematical framework for the moves of the game and the following move resolution. We construct a polynomial-time deterministic algorithm that resolves the submitted moves by the players. Using the proposed generalisation, we also formulate strategic in-game problems as decision problems, and show in which complexity class they reside. An example of these questions is how hard it is to find a defensive strategy that holds all provinces in a given set, both with or without knowledge about the incoming attack on those provinces.

# Contents


**1 Introduction** **3**

**2 Preliminaries and notation** **5**
2.1 Sets, graphs and propositional logic notation . . . . . 5
2.2 Complexity Theory . . . . . 6
2.3 Flow networks . . . . . 7

**3 The Game** **9**
3.1 Generalised Diplomacy . . . . . 9
3.1.1 The Orders . . . . . 10
3.1.2 Some final useful definitions . . . . . 13
3.2 Order resolution . . . . . 14
3.2.1 The retreat phase . . . . . 20
3.2.2 The winter phase . . . . . 20

**4 The complexity results** **22**
4.1 Defending while knowing the attack . . . . . 22
4.2 Attacking while knowing the defense . . . . . 25
4.3 Defending without knowledge of the opponents . . . . . 30
4.4 Attacking without knowledge of the opponents . . . . . 32
4.5 Working with planar graphs . . . . . 35
4.6 Overview of Results . . . . . 39

**5 Conclusion and Future work** **40**
5.1 Conclusion . . . . . 40
5.2 Future work . . . . . 41

# Chapter 1

# Introduction

*"Diplomacy is the art of telling people to go to hell in such a way that they ask for directions." – not Winston Churchill*

*"Diplomacy is an Avalon Hill (Hasbro) board game for seven players. It is an easy game to learn, but it is impossible to master." – https://www.playdiplomacy.com/*

The board game Diplomacy is released in 1959 in the United States and is not only popular among fans of the game, but also among researchers in various fields. It is a benchmark problem for the field of artificial intelligence [FAI+22]. Not only do the up to seven player in Diplomacy have a large choice of moves at their disposal each turn, but the game also has elements of natural language negotiation, and all players submit their move choice at the same time. The new game state is then computed completely deterministically. This variety of elements in one clearly defined setting has lead to interesting research, see e.g. [Ant+20] [Bak+21] [Jac+22] [Kra+22] [Bak+22].

Diplomacy is also quite different from the usual games of chess, Go or Sudoku when it comes to complexity questions. Computational (in)tractibility has been a hot topic of research ever since before the first functional computer was even invented. The theoretical limits of computers are as high as ever, and they are expanding every day. Contributing to the ever enlarging problem space of NP-complete problems is still a noble cause.

The author is not aware of any research combining computational complexity and Diplomacy. We will thus lay the groundwork for such work by providing a suitable generalisation of the board game. Afterwards, we dive into a selection of decision problems for the generalised game.

We first establish the preliminaries and notation in Chapter 2. We assume familiarity with basic set theory, graphs, and propositional logic. The used notation is described in Section 2.1. Other than that, we aim for this work to be entirely self contained. In Section 2.2 we give an introduction to the field of complexity theory, followed by an introduction to flow networks in Section 2.3.

Afterwards in Chapter 3 we explain more about the original game itself. We go through the relevant rules and combine that with providing the proposed generalisation in Section 3.1. In Section 3.2 we provide a procedure to compute the next board state of the game when all orders are submitted, and argue that this state is computable in polynomial time.

In Chapter 4 we then get to the computational complexity results. In Section 4.1 we investigate the how difficult it actually is to defend a given region while knowing what the opponent will do first. Then, in Section 4.2 we take the role of the attacking player and find out what it takes to construct an attack that conquers all of a given region while knowing the defense of the opponent(s). In Sections 4.3 and 4.4 we again ask the same questions, but then with no knowledge of the moves of the opponent. In Section 4.5, we show that the established results hold when we restrict the underlying graph of the game to a planar graph. A table containing the found complexity results can be found in Section 4.6.

In Chapter 5 you can find a summary of the findings of this thesis and some final remarks. Furthermore, some future lines of work are suggested in Section 5.2.

# Chapter 2

# Preliminaries and notation

The preliminaries of this thesis consist of basic set theory, graph theory and propositional logic. In this Section the notation used throughout the thesis is established, and we introduce the relevant basics of complexity theory and flow networks for those that are unfamiliar with the concept.

## 2.1 Sets, graphs and propositional logic notation

Sets are generally denoted with capital letters, and elements of sets with small letters wherever possible, with additional superscripts or subscripts when desired. We use $\in, \cup, \cap, \setminus, \times$ for set membership, union, intersection, set difference and the cartesian product respectively. $X \subseteq Y$ is used to denote that $X$ is a (possibly strict) subset of $Y$. With $|X|$ we mean the cardinality of set $X$, $\overline{X}$ denotes the complement of $X$, and $\wp(X)$ denotes the power set of $X$. Functions are written with small letters $f, g, h, \ldots$, and $f : X \to Y$ indicates that $f$ maps elements of $X$ to $Y$. The set of natural numbers and real numbers are written as $\mathbb{N}$ and $\mathbb{R}$ respectively. With $\mathbb{R}^+$ we mean the non-negative real numbers. The natural numbers are assumed to start with 1. We define $[n]$ to denote the set $\{x \in \mathbb{N} |\ x \leq n\}$.

Graphs are a tuple $(V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. Whenever a graph is used, it is indicated whether the graph is directed or undirected. In the former case, edges are pairs, and otherwise edges are sets. Graphs can be represented by a drawing, where circles represent nodes, and lines or arrows denote edges. Whenever such a drawing can be drawn without crossing nodes or edges, then the graph is called *planar*.

We use $\varphi, \psi$ to denote propositional or Boolean formulas. The propositional variables are written as $x_1, x_2, \ldots$. We only work with formulas in conjunctive normal form (CNF): so each formula is a conjunction ($\wedge$) of disjunctions ($\vee$) of literals. A literal ($\ell$) is denoted with $\ell_i$ and is either a positive variable $(x_j)$ or the negation of a variable, denoted with $(\overline{x_j})$. A formula is in $k$-CNF if $k$ is the maximum amount of literals in each clause. A (partial) truth assignment is a function from (a subset of the) variables to $\{0, 1\}$ and is usually denoted $\alpha$. The usual semantics for propositional logic apply to the negation, conjunction and disjunction.

## 2.2 Complexity Theory

The definitions in this Section are based on the work of Arora and Barak [AB09], but the notation is in some places adapted to suit the thesis. For a full and comprehensive overview of the field, we refer to this textbook.

The field of computational complexity theory is concerned with computational efficiency −quantifying the amount of computational resources required to solve a given task− according to Arora and Barak [AB09]. The complexity of a task is usually measured by comparing the amount of basic algorithmic steps needed to solve the task with the size of the input of the task. As the definition of "basic algorithmic steps" is not too precise[1], and possibly dependent on the details of the computer, the following definition is commonly used.

**Definition 1** (Big Oh-notation)**.** Let $f, g : \mathbb{N} \to \mathbb{N}$ be increasing functions. We say that $f \in O(g(n))$ iff there exist constants $c \geq 1$ and $n_0 \geq 0$ such that for each $n > n_0$, $f(n) \leq c \cdot g(n)$. △

So whenever $f \in O(g(n)$, then for large enough $n$, the function $g$ provides an upper bound for $f$. Note that this notion is useful for a worst-case analysis, but might be misleading for some other applications. To discuss the theoretical limits of an algorithm though, this definition is most commonly used.

With this definition in hand, we have the basis to compare two functions. However, we were interested in comparing tasks. The specific tasks we are interested are called *decision problems*. These problems consist of an input, and a yes-no question associated with that input. A decision problem is called *decidable* if there is an algorithm that given an instance of the problem, gives the correct yes or no answer. Such an algorithm is then said to decide the problem. The functions we want to compare are the functions mapping input size to the amount of steps the best known algorithm for a decision problem. To do so, we introduce the following definition.

**Definition 2** (DTIME)**.** Let $T : \mathbb{N} \to \mathbb{N}$ be some function. A decision problem $L$ is in $\mathrm{DTIME}(T(n))$ iff there is an algorithm that decides $L$ in at most $c \cdot T(n)$ steps, for some $c > 0$. △

Now we are ready to introduce the first complexity classes. These classes are used to categorize decision problems, based on their computability. The most well known time-complexity classes are P, NP, coNP, and EXP. The first class captures problems with a polynomial time decision algorithms.

**Definition 3** (P)**.** P$= \bigcup_{c \in \mathbb{N}} \mathrm{DTIME}(n^c)$. △

In sharp contrast to polynomial time, there is exponential time. The class of problems that is solvable by exponential time is captured by $\mathrm{EXP}$.

**Definition 4** (EXP)**.** EXP$= \bigcup_{c \in \mathbb{N}} \mathrm{DTIME}(2^{n^c})$. △

[1] Of course, there is a more precise definition that involves Turing Machines and a fixed (usually binary) representation. As that level of detail is not needed for the thesis, we have chosen to not include it here.

Note that as polynomial time algorithms are upper bounded by exponential ones, we have P⊆EXP. Somewhere in between, there are the classes NP and coNP. NP stands for non-deterministic polynomial time. The definition is as follows:

**Definition 5** (NP)**.** A problem $L$ is in NP iff there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ and an algorithm $A$, such that for every candidate instance $x$ of $L$ we have that the following holds: $x \in L$ if and only if there exists some $u$ with $|u| \leq p(|x|)$ such that $A$ outputs "yes" when given $(x, u)$ as input. Such an algorithm is also called a *certifier*. △

Here, $|\cdot|$ is a function that assigns to objects the size of their encoding.

Perhaps unsurprisingly, the class coNP is closely related to NP.

**Definition 6** (coNP)**.** A problem $L$ is in coNP iff $\overline{L}$ is in NP. △

Note that P⊆NP and P⊆coNP by the facts that the certificate does not need to be used by the algorithm in the definition of NP, and P is closed under complementation. Whether these inclusions are strict, we do not know. We also have that NP⊆EXP and coNP⊆EXP, as an exponential-time algorithm can simply simulate $A$ with all possible certificates. Whether these inclusions are strict is also not known. We do have a notion of hardest problems in NP however. To get there, we introduce the following definitions.

**Definition 7** (Polynomial-time reductions)**.** A problem $L$ is *polynomial-time reducible* to a problem $L'$ if there is a polynomial-time computable function $f$ such that for every candidate element of $L$ it holds that $x \in L$ just in case $f(x) \in L'$. △

**Definition 8** (NP-hardness)**.** A problem $L$ is NP-hard when every problem in NP is polynomial-time reducible to $L$. △

The hardest problems in NP are then the problems that are both in NP and are NP-hard. These problems are called NP-complete. The first known NP-complete problem is called SAT, for satisfiability. The input for these problems is a propositional logic formula, and the output is yes just in case the formula is satisfiable, and no otherwise. In this thesis, we mainly use a variant of this problem called 3-SAT. In this variant of SAT, the input is restricted to formulas in 3-CNF.

## 2.3 Flow networks

The last Section of the preliminaries is dedicated to flow networks. A flow network is a special variant of a graph. The idea can be viewed as simulating the flow of water from a source to a sink. This Section is based on the classic textbook by Cormen et al. [Cor+01, Ch. 26]. The notation is adapted to match the notation in this thesis.

**Definition 9** (flow)**.** A $s$-$s'$ *flow network* is a connected directed graph $G = (V, E)$ with edges labeled by a non-negative capacity function $c : E \to \mathbb{R}^+$. Furthermore we have $s, s' \in V$, where $s$ has no incoming edges, and $s'$ no outgoing edges. Node $s$ is called the *source* and node $s'$ is called the *sink*. The other nodes are called inner nodes.

A *flow* is a function $f : E \to \mathbb{R}^+$ with the following properties:

1. For all inner nodes $v \in V \backslash \{s, s'\}$, we have $\sum_{(v,x)\in E} f((v,x)) = \sum_{(x,v)\in E} f((x,v))$. In other words, for all inner nodes we have that the amount of flow that goes in, must be equal to the amount of flow that goes out.

2. For all $(u,v) \in E$ we have $f((u,v)) \leq c((u,v))$, so that the flow respects the capacities of the edges.

The *value of flow*, denoted $v_f$, is defined as $\sum_{(s,v)\in E} f((s,v)) = \sum_{(u,s')\in E} f((u,s'))$. That is all flow that goes from $s$ to $s'$. A flow $f$ is called *maximal* if for all flows $f'$, $v_f \geq v_{f'}$. △

A well known result about flow networks is that any flow network with integral capacities, has a maximum flow where each edge has an integral flow too. Moreover, this maximum flow can be computed in polynomial time, using the Edmond-Karp algorithm. The basic idea of this algorithm is to find a shortest path through a graph based on the flow network, and to increment the flow on this path with the maximum amount allowed by the path.

**Definition 10** (Edmond-Karp)**.** On input $G = (V, E, s, s')$:

1. Initialise the flow $f : E \to \mathbb{R}^+$ to $f(e) = 0$ for all $e \in E$.

2. Build the residual network $G_f = (V, E')$. $E'$ consists of two kinds of edges. The first kind are called forward edges and are edges such that $c_f(e) := c(e) - f(e)$, the capacity of edge $e \in E$ in the residual network, is greater than 0. These are the edges in the original network that still have room for more flow. The second kind of edges are backward edges: $(v,u)$ is a backward edge in $G_f$ if $c_f(v,u) := f((u,v)) > 0$. These edges are used to be able to cancel out already determined flow.

3. Determine a shortest path $p$ (where each capacity unit is viewed as a distance unit) from source $s$ to sink $s'$ in the residual network, using breadth first search. If such a path does not exist, return $f$. Else, determine $f^* := \min_{e\in E'} c_f(e)$. Augment the flow $f$ on all edges of $p$ with $f^*$. Repeat steps 2 and 3.

△

In each iteration of step 3, the total flow of the network increases, eventually terminating at a maximum flow. For a proof of this claim, and a runtime analysis of the algorithm, we again refer to Cormen et al. [Cor+01, Ch. 26].

This concludes the preliminaries of the thesis. In the next Chapter we turn to the game and get to the first results.

# Chapter 3

# The Game

In this chapter, we introduce the game Diplomacy in detail. While going through the rules of the game, we also immediately propose a generalisation. This chapter ends with a newly constructed algorithm for order resolution, following the mentioned rules.

Diplomacy is a strategic board war game, played with up to 7 players. It can be played over the board as well as online. In the game, each player controls a major power, consisting of units. The board is a map of Europe, divided into territories or Provinces. Each territory can be occupied by at most one unit, and each unit has the same strength. Some provinces have a supply centre, granting a way to the players to get more units for their armies. The game is won by the player who is the first to take control of a certain amount of supply centres at once.

What makes Diplomacy unique is that every turn, all players submit their moves, called orders, simultaneously. The set of moves from a player is called a *strategy*. The players submit their strategies after a negotiation phase. Then, there is a set of rules that determines the outcome of the moves, resulting in a new board state. This resolution process is entirely deterministic.

This is the general gist of the game. What will follow is a selection of the rules and details that are relevant to the research questions in this thesis. The author is aware that many interesting aspects of the game will be left out, but believes that the quality of this thesis is improved by these omissions. We will continue with the description of Generalised Diplomacy as proposed by the author. Any diversion from the original game that is not a form of a generalisation will be highlighted. Although the negotiation phase is a key element of the game play, it does not have a roll in this aspect of the game.

## 3.1 Generalised Diplomacy

Generalised Diplomacy is played by $k \in \mathbb{N} \setminus \{1\}$ players on an undirected graph $G = (V, L \cup W)$. The set of vertices $V$ in the graph represent the provinces, or territories. The first set of edges $L$ denote the borders on land, and $W$ represents the neighbourhood relation via water. We can immediately distinguish three types of provinces now.

1. Provinces with incident edges in both $L$ and $W$ are called *coastal provinces*.

2. Provinces with just incident edges in $L$ are called *inland provinces*.

3. Provinces with just incident edges in $W$ are called *water provinces*.

**Remark 1.** Note that in the original game, there are some provinces with so called 'split coasts'. These provinces limit the movement options for fleets. In this generalisation, for the sake of simplicity, we will ignore these map oddities.

Each non-water province can hold a supply centre. The total amount $c$ of centres should be greater than the amount of players, but other than that we leave the exact amounts unspecified. A player has won the game once they control at least $\frac{1}{2}c < s \leq c$ centres at once. The exact amount we choose to also leave underspecified; it should be more than half of the total amount of centres however, because otherwise more players could reach the victory condition at the same time. A player $p$ controls a centre when the last unit that occupied the province in which the centre is located, belongs to $p$.

Even though each unit has the same strength, two types of units are distinguished. The first type is a *fleet* unit, and they can only occupy the water and coastal territories, and can only travel over the edges in $W$. The second type is an *army* unit, and those are land-based. They can only occupy coastal and inland provinces and can only travel over the edges in $L$. The player makes orders to command the units. These will be explained in the next section. Throughout this thesis, the terms *unit* and *province* are sometimes used interchangeably. For example, when we write that $v$ moves to $u$ for $v, u \in V$, we mean that the unit on $v$ moves to the province $u$.

We first go through the orders and their possible intended effects. Later, in the section on resolution, we go through some more details of some more interactions between orders.

The difficulty of computing the new game state lies in determining how many support each action receives.

### 3.1.1 The Orders

There are four kinds of orders. There is *hold*, *move*, *support* and the fleets can also *convoy*. We go through these orders one by one, and afterwards define the necessary concepts to get to the results in the next chapter. As we want to talk about strategies and orders in a formal setting, we will define the orders as tuples. The strategies will then be a set of orders. The order tuple will have information about the player giving the order, the relevant provinces, and the type of order. In the following, we denote the players giving the orders with $p$ or $p'$. For the provinces in $V$, we will use $u$, $v$ and $w$.

#### Hold

This order makes the unit try to defend its province with a strength of 1. It also means that the unit is able to receive defensive support. More on that in the support section. We denote a hold order for $u$ with the tuple $(p, u, \mathbf{h})$. When no order is specified for a unit, it is assumed to hold too.

**Move**

This is the basic attack order. When a fleet unit is in a province $v$, it can only move to provinces $u$ whenever $\{v, u\} \in W$. Likewise, an army unit can only move to provinces adjacent to it via the edges in $L$, unless they are convoyed. The details of the convoy process will be discussed later.

Whenever a unit moves to another province, it attacks that province with a force of 1, and it is open to receive attacking support. The tuple $(p, u, \mathbf{a}, v)$ denotes that the unit on province $u$ attacks, or tries to move to, province $v$. A move to a province is successful if the attacking force is strictly greater than the defending force and stronger than any other attacking force on that province. Whenever that is the case, we also say that the target province is *conquered*. When there are multiple attacks on one target province $v$, and the greatest forces have equal strength, then there is a *standoff*. This means that no unit moves to the target province, so the province is not considered conquered.

If two units try to move to each others province, only the one with the greatest force with succeed. If both move with equal strength, they remain were they were. An exception is when at least one of the units can use a convoy instead. More on that in the convoy order section.

If a province $v$ is conquered, and there is a unit of player $p$ on this province $v$ that did not (successfully) move away, then unit $v$ is *dislodged* (unless the conquering player is $p$ themselves). Whenever $v$ is conquered by $w$, any order to $v$ with target $w$ is cancelled. After the resolution phase, all dislodged units will need to retreat. More on this process in subsection 3.2.1.

**Support**

Via support, a defending or attacking force can get stronger than just a force of 1. We distinguish two kinds of support, also referred to as *direct support*.

1. The first version is *defensive support*. A unit on province $v$ can be successfully be ordered to offer defensive support to a unit on province $u$ if the following conditions are met:

   (a) The unit on $u$ could have also been ordered to move to province $u$, i.e. $\{u, v\}$ is in the correct neighbourhood relation for unit $u$.

   (b) The unit on $u$ is not ordered to move. A unit trying to move cannot receive any defensive support.

   (c) The support is not cut; more on this below.

   Whenever a defensive support order is successful, the strength of the defense of the unit that receives the support goes up by 1 unit of strength. For $u, v \in V$, we write $(p, u, \mathbf{s}, v)$ to denote unit $u$ giving defensive support to $v$.

2. The other form of support is *offensive support*. As the name suggests, this version aims to help the attacking side conquering a province. This command needs to specify 4 components: $(p, u, \mathbf{s}, v, w)$ denotes unit $u$ supporting unit $v$ to attack province $w$. For this command to be successful, we have analogous conditions to the defensive support:

(a) The unit on $u$ could have been ordered to move to $w$.

(b) The unit on $v$ needs to be ordered to move to $w$.

(c) The support is not cut; more on this below.

If all three conditions are met, the attacking force of $v$ on $w$ is increased by 1.

Whenever a unit $u$ is ordered to offer support, it also defends its own province with a strength of 1, and it is open to receive defensive support from other units. Note that a player can choose to support units of other players, and that support cannot be refused[1].

### Cutting support

Although support cannot be refuted, there are other ways to stop support, namely by cutting it. We first discuss when defensive support is cut, and then we go on to offensive support.

Let $(p, u, \mathbf{s}, v)$ be the order by player $p$ that $u$ should provide defensive support to $v$. This support is invalid whenever $v$ is ordered to move anywhere, but it can also be cut when some player $p' \neq p$ orders a province $w$ to move to $u$.

The case for offensive support requires a bit more attention. Let $(p, u, \mathbf{s}, v, w)$ be the given support order. This support is invalid when $v$ does not move to $w$. The support is cut too whenever some player $p' \neq p$ orders a province $x \neq w$ to move to $u$. Furthermore, if $(p', w, \mathbf{a}, u)$ is ordered, then this attack can cut the support of $u$ as long as the attack on $u$ is successful, i.e. when $u$ is conquered by $w$.

When support is cut, the province that were to receive this support, does not receive it. Some example situations are explained in Figures 3.1 and 3.2 below.

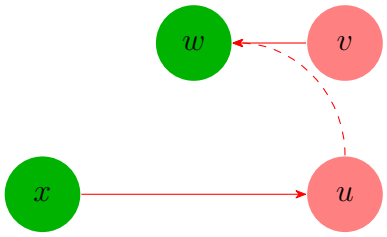



Figure 3.1: In this example, $v$ moves to $w$ with offensive support of $u$. This support is cut however by $x$ moving to $u$. Hence, none of the provinces are conquered here.

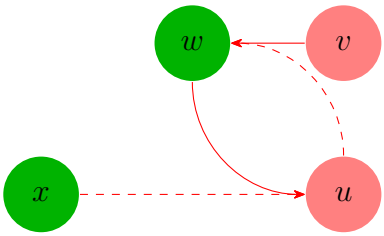



Figure 3.2: In this scenario, $w$ moves to $u$ with the support of $x$, while $v$ moves to $w$ with support of $u$. As $w$ conquers $u$, the support by $u$ is cut. Had $x$ not delivered support to the attack of $w$ against $u$, then the support by $u$ would have gone through.

### Convoy

The last type of order is the convoy order. A convoy order can be given to any fleet province in a water (so not coastal) province. A convoy is used to move armies

[1] Support can in general not be refused. The only exception to this case is when offensive support of a player $p$ directly contributes to the dislodging of a unit of player $p$. In this case the support is refuted. The amount of support that is refuted is the minimum amount that is needed to prevent the dislodging.

(not fleets) over bodies of water. A convoy is ordered with the tuple $(p, u, \mathbf{c}, v, w)$, which indicates that the player $p$ orders the fleet on $u$ to convoy the army on $v$ to province $w$. In the process, the fleets are open for defensive support. The following requirements must be met for a successful transport:

1. Provinces $v$ and $w$ must be coastal, and $v$ must be an army unit. Furthermore, $u$ must be a water province.

2. $(p', v, \mathbf{a}, w)$ is ordered, so the army on $v$ must intend to move to $w$.

3. There is at least one path from $v$ to $w$ in $W$, and all fleets on this path are ordered to convoy $v$ to $w$.

4. On at least one of these paths, no fleet unit is dislodged during the same turn. Note that if the support of unit $w$ is needed to dislodge one of these fleets, and this support is only attempted to be cut by the attack from $v$, then the fleet unit is in fact dislodged. The attack on the fleet thus takes precedence over the transport of unit $v$, but only if there is no other open path from $v$ to $w$. This situation is illustrated by Figures 3.3 and 3.4.

5. Moreover, the same rules for a successful attack apply (i.e. the force of the attack must be greater than the force of the defense). Note that with a convoy however, two armies can switch places in the same turn. This happens if at least one of the armies has a travel option via convoy, no matter which army has the most support.

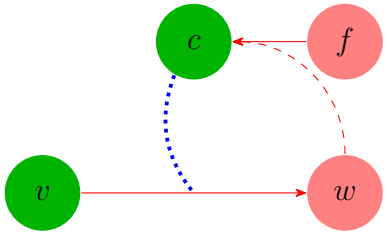



Figure 3.3: Here, $v$ moves to $w$ with the help of convoy $c$. $f$ moves to $c$ with attacking support of $w$. As $v$ is the only attacker of $w$, and the only convoy of this attacker is disrupted, the support by $w$ is not cut.

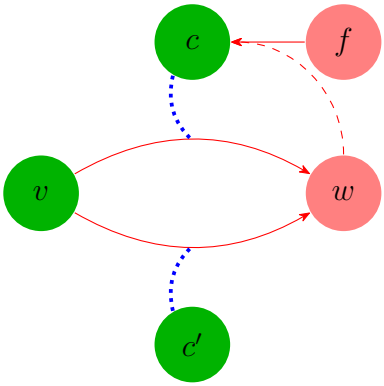



Figure 3.4: In this scenario, the attacking support by $w$ is cut by $v$, as $v$ has an open convoy route via $c'$.

A failed convoy is treated the same as a failed 'regular' move for the army. For the fleets involved, the effect is as though they were ordered to hold.

### 3.1.2 Some final useful definitions

For the results of this thesis, we usually have a set of provinces that we are looking to conquer or defend. This set will be referred to as the target set $T$. The players trying to conquer the targets are called attackers, and the defenders are called defenders. For the attacking players, we distinguish three categories of attacking provinces:

1. Main attackers: these are the units that are ordered to move to the targets.
2. Direct supporters: these are the units ordered to provide offensive support to the main attackers.
3. Indirect supporters or indirect attackers: these are units ordered to move to potential defenders, aiming to cut their defensive support.

For the defenders, we make a similar categorisation.

1. Main defenders: the units ordered to hold the targets, or move to the targets in order to defend it.
2. Direct supporters: units ordered to provide defensive support to the target, or offensively support a main defender when relevant.
3. Indirect supporters: defenders that move to potential attackers, aiming to cut their offensive support.

The following definitions are convenient too for out research questions.

**Definition 11** (Perfect attack)**.** Given a game state, a player $p$, and a target set $T$ consisting of provinces not belonging to $p$, a *perfect attack* by $p$ is a strategy such that no matter the strategies of the other players, $p$ occupies all $t \in T$ the following turn.

**Definition 12** (Perfect defense)**.** Given a game state, a player $p$, and a target set $T$ consisting of provinces belonging to $p$, a *perfect defense* by $p$ is a strategy such that no matter the strategies of the other players, $p$ occupies all $t \in T$ the following turn.

**Definition 13** (Strength)**.** Let $u, v \in V$, $p \in P$, and let $\mathcal{S}$ be a set of orders. If there is an order $(p, u, \mathbf{a}, v) \in \mathcal{S}$, then the strength of the attack of $u$ on $v$ is $1 + |\{(p', x, \mathbf{o}, y, z) \in \mathcal{S} \ : \ \mathbf{o} = \mathbf{s}, y = u, z = v\}|$, i.e. one more than the amount of offensive support $u$ receives.

The strength of the defense of $v$ is defined analogously, and is one more than the amount of defensive supporters of $v$, The strength of the defense of $v$ is only defined if $v$ is not ordered to move.

## 3.2 Order resolution

After all players submitted their orders, a new intermediate game state is computed. In this computation, any conflict caused by orders with the same goal must be resolved. This also happens deterministically. In this section, we provide an algorithm for this, and analyse its runtime. The Let $\mathcal{S}$ denote the union of all strategies by all players. First we formally describe the game state.

**Definition 14** (Game states)**.** A game state $GS$ contains the following information.

1. A turn counter $I$, which is set to 1 at the beginning of the game.
2. The set of players $P = \{p_1, \ldots, p_k\}$, where $k$ is the number of players.

3. The board map $G = (V, L \cup W)$ with the provinces, the land and the water borders.

4. A set $C \subseteq V$ containing the provinces that house a supply centre.

5. A function $c_I : C \to P \cup \{0\}$ indicating which supply centre is controlled by which player, if any, on turn $I$. $c_1$ is determined before the game.

6. A function $occupy_I : V \to [P \times \{\textbf{army}, \textbf{fleet}\}] \cup \{0\}$, assigning to each province which (if any) player occupies the province on turn $I$, and with which type of unit.

We assume that all move and support orders are complete valid. Here, that means that every player gives orders only for all of their own units, and that every order does not require any illegal action to succeed (such as jumping the map) of succeeding.

Now we turn to the resolution algorithm. The input of the algorithm is a game state and a set of strategies, and the output is as stated, an intermediate game state. This is due to the fact that after the resolution phase, some units may have been dislodged. Any such unit has to be either located elsewhere, or could be destroyed. Also, when the turn counter $I$ is an even number, every player must make sure that their amount of units equals the amount of supply centres they own. This is done by disbanding or placing more units on the map. After these intermediate states, the turn counter is increased by 1, and a new turn can begin if nobody has won yet.

The algorithm works in steps. First it figures out the first wave of support that is cut. These are the support orders that are being attacked by other players, and at least one of those attack must not rely on convoys. Then it turns to the provinces of units that are ordered to convoy some other unit. Let $C_v := \{u \in V \ : \ (*, u, \mathbf{c}, *, v)\}$[2] be the set of fleet units that are ordered to convoy some unit to $v$. In each such set $C_v$, there is at most one province that we might not be able to resolve yet. That is the province $u$ that $v$ is ordered to support in. Whether this support is cut or not, can depend on whether some parts of the convoy is dislodged. We therefore first sort out the other provinces in $C_v$. Then we check whether there are still paths from attackers of $v$ to $v$, so that we know how much support is delivered in $u$.

Afterwards, we have enough information to resolve all provinces except the ones that are occupied by units that are ordered to move somewhere. That group is dealt with in the last step.

**Definition 15** ($Res$)**.** $Res(GS, \mathcal{S})$ is computed as follows:

1. Initialise a set $Dislodged := \emptyset$ to track which units are dislodged at the end of the resolution. Initialise a set $Succeed := \emptyset$ to keep track of the successful attacking/move orders. Let $M \subseteq \mathcal{S}$ denote the set of all move orders for later.

2. Next, we remove all support orders from $\mathcal{S}$ that are guaranteed to be cut:

   (a) For all defensive support orders $(p, u, \mathbf{s}, v) \in \mathcal{S}$: if there is a move order $(p', w, \mathbf{a}, u) \in \mathcal{S}$ with $p \neq p'$, $\{w, u\} \notin L$, and no convoy order matching $(*, *, \mathbf{c}, w, u)$ is given, then remove $(p, u, \mathbf{s}, v)$ from $\mathcal{S}$. Add $(p, u, \mathbf{h})$ to $\mathcal{S}$.

[2]The $*$ here denotes a wildcard variable, and its value does not matter. Any instance of $*$ can be the same or different from other instances of $*$.

(b) Analogously, for all attacking support orders $(p, u, \mathbf{s}, v, w) \in \mathcal{S}$: if there is a move order $(p', x, \mathbf{s}, u) \in \mathcal{S}$ such that $p \neq p'$, $x \neq w$, $\{x, u\} \notin L$, and no convoy order matching $(*, *, \mathbf{c}, x, u)$ is given, then remove $(p, u, \mathbf{s}, v, w)$ from $\mathcal{S}$. Add $(p, u, \mathbf{h})$ to $\mathcal{S}$.

3. Let $\mathcal{C} \subseteq \mathcal{S}$ be the set of all convoy orders. Initialise set $Later := \emptyset$. For each $(p, u, \mathbf{c}, v, w) \in \mathcal{C}$, if $(p', w, \int, x, u) \in \mathcal{S}$, then add that order to $Later$ and remove it from $\mathcal{C}$.

   Now do the following for each $(p, u, \mathbf{c}, v, w) \in \mathcal{C}$: Let $d_u = |\{(*, *, \mathbf{o}, x) \in \mathcal{S} : \mathbf{o} = \mathbf{s}, x = u\}| + 1$ be the strength of the defense of $u$. Let $B_u := \{(*, *, \mathbf{o}, u') \in \mathcal{S} : \mathbf{o} = \mathbf{a}, u' = u\}$ be the set of orders to the main attackers of $u$, and let $f : B_u \to \mathbb{N}$ be defined as $f((*, x, *, *) = 1 + |\{(*, *, \mathbf{o}, x', u') \in \mathcal{S} : \mathbf{o} = \mathbf{s}, x' = x, u' = u\}|$. Then $A_u := \arg\max_{x \in B_u} f(x)$ is the set of attacking orders against $u$ with the most offensive support, that being $a_u := \max_{x \in B_u} f(x)$. We distinguish the following two cases:

   (a) $A_u = \{(p', x, \mathbf{a}, u)\}$. Let $b_u := \max_{x \in B_u \setminus A_u} f(x)$ denote the strength of the second strongest attacks. Now consider $a'_u := a_u - |\{(p'', *, \mathbf{o}, x', u') \in \mathcal{S} : p'' = p, \mathbf{o} = \mathbf{s}, x' = x, u' = u\}|$, denoting the strength of the attack without the support from $p$ against their own unit. If $a'_u > \max(b_u, d_u)$ and $p' \neq p$, only then is province $u$ conquered. We set $occupy_{I+1}(u) = occupy_I(x)$, we add $u$ to $Dislodged$, $x$ to $Succeed$ and remove the convoy order $(p, u, \mathbf{c}, v, w)$ from $\mathcal{S}$. Else, we set $occupy_{I+1}(u) = occupy_I(u)$.

   (b) Otherwise, $occupy_{I+1}(u) = occupy_I(u)$. In this case there either was a standoff in province $u$, or $u$ was not moved to. Either way, unit $u$ remains in place and the convoy is not disrupted in this territory.

   Remove all elements of $B_u$ from $\mathcal{S}$.

4. Let $\mathcal{A} \subseteq \mathcal{S}$ be the set of all move/attack orders. Initialise $Convoy := \emptyset$. For each $(p, u, \mathbf{a}, v) \in \mathcal{A}$, remove the order from $\mathcal{S}$ if the following holds:

   $(u, v) \notin L$ and there is no path $(u, w_1, w_2, \ldots, w_n, w_{n+1})$ such that all of the following is true:

   (a) $\{u, w_1\} \in W$ and $w_{n+1} = v$,

   (b) $\{w_i, w_{i+1}\} \in W$ for $i \in [n]$, and

   (c) for each $i \in [n]$, there is an order $(*, w_i, \mathbf{c}, u, v) \in \mathcal{S}$.

   If such a path does exist, even if $(u, v) \in L$, add the set $\{u, v\}$ to $Convoy$.

5. For all defensive support orders $(p, u, \mathbf{s}, v) \in \mathcal{S}$ and offensive support order $(p, u, \mathbf{s}, *, v) \in \mathcal{S}$: if there is a move order $(p', w, \mathbf{a}, u) \in \mathcal{S}$ with $p \neq p'$, then remove the support order from $\mathcal{S}$ and add $(p, u, \mathbf{h})$ to $\mathcal{S}$.

6. Let $\mathcal{D} \subseteq \mathcal{S}$ be the set that contains all defensive support orders, let $\mathcal{O} \subseteq \mathcal{S}$ be the set of all offensive support orders, and let $\mathcal{H} \subseteq \mathcal{S}$ be the set of all hold orders. For each $(p, u, \mathbf{s}, v) \in \mathcal{D}$, each $(p, u, \mathbf{s}, *, v) \in \mathcal{O}$, and each $(p, u, \mathbf{h}) \in \mathcal{H}$, we are going to determine what happens in $u$.

   Find $d_u$, $B_u$ and $A_u$ as defined in step 3. We again distinguish two cases:

(a) If $A_u = \{(p', x, \mathbf{a}, u\}$ is a singleton, also determine $b_u$ and $a'_u$ as in step 3(a). If $a'_u > \max(b_u, d_u)$ and $p' \neq p$, province $u$ is conquered. We set $occupy_{I+1}(u) = occupy_I(x)$, we add $u$ to $Dislodged\ x$ to $Succeed$, and remove the support order $(p, u, \mathbf{s}, v)$ or $(p, u, \mathbf{s}, *, v)$ if still present and relevant from $\mathcal{S}$. Else, we set $occupy_{I+1}(u) = occupy_I(u)$.

(b) Otherwise, $occupy_{I+1}(u) = occupy_I(u)$.

Remove all elements of $B_u$ from $\mathcal{S}$.

7. On this point, there are two types of provinces left that are still being moved to with orders from $\mathcal{S}$. The first type is empty provinces, and the second type is provinces with units that are also ordered to move. This step deals with the former group. This group does not have defenders, but there can still be multiple forces trying to conquer the region.

   Let $\mathcal{M}_0 \subseteq \mathcal{S}$ denote the set of orders that move to provinces $u$ with $occupy_I(u) = 0$. For each $(p, v, \mathbf{a}, u) \in \mathcal{M}_0$, determine $A_u$, $B_u$ and $a_u$ as in step 3.

   (a) If $A_u = \{(p', w, \mathbf{a}, u)\}$, there is a single strongest attacker that takes the province. Set $occupy_{I+1}(u) = occupy_I(w)$ and add $w$ to $Succeed$.

   (b) Else there is a standoff, so no unit involved moves. Set $occupy_{I+1}(u) = 0$.

   Remove all elements of $B_u$ from $\mathcal{S}$ and from $\mathcal{M}_0$

8. Let $\mathcal{M} \subseteq \mathcal{S}$ denote the set of move orders, with as a target a province with a unit that is also ordered to move somewhere. Some of these moves were already successful, some were not, and others may have not been tried yet. Introduce a set $DependsOn := \emptyset$. We iterate over the elements of $\mathcal{M}$.

   For each $(p, v, \mathbf{a}, u) \in \mathcal{M}$, set $current := u$ and do the following.

   (a) Determine $B_u$, $A_u$ and $a_u$ as in step 3. Let $(p', \mathbf{type}) = occupy_I(u)$.

   (b) If $u \in Succeed$, then province $u$ is currently unoccupied.

      i. If $A_u = \{(p'', w, \mathbf{a}, u)\}$, there is a single strongest attacker that takes the province. Set $occupy_{I+1}(u) = occupy_I(w)$ and add $w$ to $Succeed$.

      ii. Otherwise there is a standoff, so no unit involved moves. Set $occupy_{I+1}(u) = 0$.

   (c) Else, if there is no $(p', u, \mathbf{a}, w) \in \mathcal{M}$, then the move by $u$ has failed, and $d_u = 1$.

      i. If $A_u = \{(p'', w, \mathbf{a}, u)\}$, then there is a single strongest attacker $w$. Calculate $b_u$ and $a'_u$ as in step 3(a) again.
      If $a'_u > \max(b_u, d_u)$ and $p' \neq p''$, then province $u$ is conquered. We set $occupy_{I+1}(u) = occupy_I(w)$, we add $u$ to $Dislodged$ and $w$ to $Succeed$. Else, we set $occupy_{I+1}(u) = occupy_I(u)$.

      ii. Otherwise there is a standoff, so no unit involved moves. Set $occupy_{I+1}(u) = 0$.

   (d) Else, there is some $(p', u, \mathbf{a}, w) \in \mathcal{M}$. We distinguish two cases.

      i. If $p = p'$, all of the following must hold.

A. $(p, u, \mathbf{a}, w)$ is successful.

B. $w \neq v$ or $\{u, v\} \in Convoy$,

C. $(p, v, \mathbf{a}, u)$ is the single strongest attacker of $u$.

ii. If $p \neq p'$, all of the following must hold.

A. If $w = v$, either the strength of $v$ into $u$ must be strictly greater than the strength of $u$ into $v$, or both $\{u, v\} \in Convoy$ and $(p', u, \mathbf{a}, v)$ is successful.

B. Either $v$ is the single strongest attacker into $u$, or $w$ is the single strongest attacker into $u$ and $u$ successfully moves to $w$, wins a head to head without convoy against $w$.

C. Conquering $u$ is both independent of the support of $p'$ to $v$ and the attack of $v$ into $u$ has a strength greater than 1 (i.e. $a'_u > \max(b_u, 1)$), or $(p', u, \mathbf{a}, w)$ is successful.

If whether $u$ is conquered depends the success of $(p', u, \mathbf{a}, w)$, add $(v, u)$ to $DependsOn$. Set $current = w$ and go back to 8(a) to directly determine what happens in $w$.

Otherwise, remove all orders in $B_u$ from $\mathcal{M}$.

If $u$ was conquered by some $x \in V$, set $occupy_{I+1}(u) = occupy_I(x)$ and add $x$ to $Succeed$. Set $current := x$.

While there is some $(y, current) \in DependsOn$, repeat the following: add $y$ to $Succeeds$ and set $occupy_{I+1}(current) = occupy_I(y)$. Remove $(y, current)$ from $DependsOn$, remove all orders in $B_x$ from $\mathcal{M}$. Set $current = y$.

For all failed move orders $(*, x, \mathbf{a}, u) \in B_u$, do the following. Set $current := x$. While there is some $(y, current) \in DependsOn$: remove $(y, current)$ from $DependsOn$. Set $current = y$ and remove the given move order if still there.

9. For each province $v \in V$, if $occupy_{I+1}(v) \neq occupy_I(v)$ and $v \notin Succeed$, add $v$ to $Dislodged$.

10. Output $occupy_{I+1}$, $Dislodged$ and $M$.

△

The output will be used by the retreat phase algorithm to determine the new game state. We get the following important result.

**Lemma 1.** *$Res$ is a polynomial-time function*

*Proof.* As the input of $Res$ is a full game state, it is not too hard to see that the function is computable in polynomial time.

Step 1 is done in $O(|\mathcal{S}|)$ for copying the set of move-orders.

Step 2 is done in $O(|\mathcal{S}|^3)$, as we check for each support order whether there is a matching attack order, and for each such attack order whether there is no matching convoy order.

In step 3, for each convoy order of which there are $O(|\mathcal{S}|)$ many, the strength of a defense is calculated, and the strength of all attacks into the relevant province

are calculated. The runtime of this step is thus dominated by the calculation of the strengths of the attacks, so by $O(|\mathcal{S}|^3)$.

In step 4 we look for a convoy path for each move order to army units, which can be done by breadth first search over graph $(V', W)$, where $V'$ consists of the province with the army unit $u$, its target province $v$, and all provinces of fleet units that are ordered to convoy $u$ to $v$. Creating this graph can be done in $O(|GS|)$, and the paths can be found by breadth first search, making this step polynomial-time too.

Step 5 can be done in $O(|\mathcal{S}|^2)$.

Steps 6 and 7 are like step 3 dominated by $O(|\mathcal{S}|)$ for the same reasons.

The worst case in step 8 would be all units being ordered to move in a giant loop, so that all moves depend on the success of the target move. It again takes $O(|\mathcal{S}|^3)$ time to determine for a single move whether it is one of the strongest attacks on their target. Whether all the moves then succeed takes $O(|\mathcal{S}|)$ of those checks. Hence the runtime of step 8 is dominated by $O(|\mathcal{S}|^4)$.

Steps 9 and 10 then take at most $O(|GS|)$ time.

As $O(|\mathcal{S}|) \in O(|GS|)$, the total runtime of the algorithm is bounded by $O(|GS|^4)$, which makes the $Res$ run in polynomial time. □

**Correctness of $Res$**

At https://www.playdiplomacy.com/help.php?sub_page=Game_Rules, there is a short declarative overview of the rules. Rules 12 through 24 there are concerned with order resolution. In order to argue for the correctness of $Res$, we will go through each of these statements, and briefly point out how the algorithm ensures that these statements are obeyed.

Rule 15 states that if units of equal strength move to the same province, neither advances. This is taken care of by $Res$, because we require that there is a single strongest attack in order for a unit to move.

Rule 16 states that two units cannot switch places without a convoy, and that is taken care of by step 8(d)i.B. and 8(d)ii.(A).

A unit that does not advance retains a defensive strength of 1 according to rule 17, and that is ensured by the fact that in step 8(c) we state that the defense ($d_u$) has force 1.

By requiring the single strongest attacks to be strictly greater than the defending forces in order to move, we obey rule 18. The notion of cut support is discussed in rule 19, and that is taken care of by steps 1 and 5. By removing the support orders that are cut, they cannot influence the move orders.

The fact that a dislodged unit cannot support anything is stated in rule 20 and is enforced in $Res$ by step 6(a), through deleting the support order of dislodged units. By not deleting move orders from dislodged units, rule 21 is satisfied.

Rule 22 states that you cannot dislodge or cut support of your own units. This is enforced by $Res$ through the various $p \neq p'$ conditions.

A unit on a province where a standoff occurred is not dislodged correctly too, obliging rule 23. The disruption of a convoy through dislodging one of the fleets is taken care of is taken care of by steps 3 and 4. Hence rule 24 is satisfied as well.

Now we have the resolution algorithm, we are ready to get to the complexity results of the thesis. The units that are assigned by $occupy_{I+1}$ are final for that turn.

However, we do not have the full new game state yet; we do not know where to go with the dislodged units, and what to do if it is winter in the game. For the completeness of the generalisation, we have included short algorithms for these phases of a turn in game too.

### 3.2.1 The retreat phase

After the resolution phase, there is a retreat phase. Fortunately, this phase is short and easy to resolve. Each unit in the $Dislodged$ set must find a new home or be disbanded. This new home must be a region adjacent to where the unit was dislodged, and must not be a province that was the target of any attack that turn, whether that attack was successful or not. Negotiation is not allowed prior to this phase.

Each player that has units in $Dislodged$ submits their orders simultaneously. The orders consist of the player name, the starting province to indicate which unit is ordered, and the target province, or the word **disband**. They are thus of the form $(p, u, v)$ or $(p, u,$ **disband**$)$. An order of the form $(p, u, v)$ is only valid if either the unit $u$ is an army unit and $(u, v) \in L$, or $u$ is a fleet unit and $(u, v) \in W$. Let $\mathcal{R}$ denote the set of valid retreat orders. A retreat to a region $v$ is successful if there was no move order submitted to $v$ in the current turn (that was just resolved, that is why we wanted the set $M$ from $Res$), and there is no other unit aiming to retreat to $v$. If multiple units have a retreat order with the same destination, they are all disbanded. The new intermediate game state is then determined as follows:

**Definition 16** ($Retreat$)**.** $Retreat$ on input $(M, \mathcal{R})$ does:

1. Set $I$ to $I + 1$.
2. For all $(p, u, v) \in \mathcal{R}$ with $v \in V$, check whether $occupy_I(v) = 0$, and whether there was no move of the form $(p', w, \mathbf{a}, v) \in \mathcal{R}$. Also check whether there is no other $(p', w, v) \in \mathcal{R}$. If all these conditions are met, set $occupy_I(v)$ to $occupy_{I-1}(u)$.

△

Now, we almost have the new game state. We just need to reassign supply centres to players. This is done by the algorithm in the next section.

### 3.2.2 The winter phase

If the played turn was a fall turn, which is when the turn counter was even and now set to odd, players must adjust the amount of units they have to match the amount of supply centres they control. This is of course also restricted to some rules. If the turn counter was set to even, the supply centres do not get a new owner. For the completeness of the game state, we do update $c_I$ however.

**Definition 17** (Winter)**.** $Winter$ on input $(occupy_I, c_{I-1})$, does as follows:

1. If $I$ is even, let $c_I = c_{I-1}$ and terminate.

2. Else, if $I$ is odd, we define $c_I$ and keep a counter $U_p$ initialised to 0 for each player $p$. For each $c \in C$, if $occupy_I(c) = (p, *)$, set $c_I(c) = p$. If $occupy_I(c) = 0$, set $c_I(c) = c_{I-1}(c)$.

3. Set $U_p := |\{c \in C \mid c_I(c) = p\}|$.

4. If a player has $k$ more units than they have supply centres under their control, they must select $k$ units to disband.

5. If a player has $k$ less units than supply centres under their control, then they may place up to $k$ units on each province of an unoccupied supply centre in their home country. The home country is predetermined before the game and is left unspecified in this generalisation.

△

This is all you need to know to play generalised Diplomacy. In the next chapter, we will look at some interesting complexity results for generalised Diplomacy.

# Chapter 4

# The complexity results

In this chapter, the main results and proofs of the thesis are described. In section 4.1, we discuss the computational complexity of the case where the player tries to defend a given set of provinces against a known attack. In section 4.2, we reverse the roles and cover the complexity of attacking against a known defense. To do so, we introduce the concept of flow networks there too. Afterwards in sections 4.3 and 4.4, we discuss the cases where there is no knowledge of what the opponents do. In the last section of this chapter, we show that the made reductions hold for planar maps too, without losing the established complexity results.

**Remark 2.** If some unit $u$ moves via convoy to some province $t$, then the last fleet unit of the convoy can also move to $t$. The two main differences are that (1) with the use of a convoy, units $u$ and $v$ could switch places, and (2), an army unit arrives at $v$ instead of a fleet unit. As (1) is not relevant for the questions, and (2) is irrelevant when concerned with just a single turn, we do not consider convoy orders throughout this thesis.

## 4.1 Defending while knowing the attack

These first two sections are dedicated to acting with perfect information of the chosen moves by the opponents in the game. While not too natural in the game setting per se, they help us understand the mechanics of the game more easily. Moreover, the results can be used as building blocks for the results later on this chapter. The first problem we are looking at is formulated as follows:

**Definition 18** (P1)**.** Given a game state $GS$, a set of provinces $T$ all belonging to the player, and the set $M$ of moves made by all opponents, does there exist a strategy extending $M$ such that the player successfully defends all provinces $t \in T$?

**Remark 3.** One remark to make about this definition is that the amount of opponents is left unspecified. As we only care about the results of one player, we will view the opponent as one player for the sake of the readability of the proofs.

We get the following result.

**Lemma 2.** *P1 is NP-complete.*

*Proof.* In order to show that P1 is in NP, we define certifier $C$, that takes as certificate the set $M_p$ of moves by the player that ensure that all provinces are successfully defended.

Certifier $C$, on input $(GS, T, M, M_p)$ does the following:

1. Check whether all $t \in T$ are occupied by the player in $Res(GS, M \cup M_p)$. If yes, accept. Otherwise reject.

△

The certificate is polynomial in size of the input, because $|M_P|$ is polynomial in $|GS|$, as each province receives at most one order. The algorithm runs in polynomial time as $Res$ is the polynomial order resolve function (see 1), and checking which player occupies the provinces of $T$ can also be done in $O(|T|)$.

Hence what is left to show is that $P1$ is NP-hard. We do so by giving a reduction from 3-SAT.

Let $\varphi$ be a formula with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. Let $\mathcal{L}$ denote the set of literals in $\varphi$ and let $f : \mathcal{L} \to \mathbb{N}$ be a function mapping literals to the amount of occurrences of that literal in $\varphi$. For each variable $x_i$, we construct a gadget consisting of multiple components:

1. Three nodes belonging to the player: $\ell_i$, $x_i$, $\overline{x_i}$. Two nodes belonging to the enemy, $a_i$ and $\overline{a_i}$. $\ell_i$ is connected to $a_i$ and $\overline{a_i}$. $a_i$ is connected to $x_i$ and $\overline{a_i}$ is connected to $\overline{x_i}$.

2. Six more sets of nodes, of which two belong to the player, $D_i$ and $\overline{D_i}$, and four belong to the enemy, $A_i, A_i', \overline{A_i}, \overline{A_i'}$. All nodes in $A_i, A_i', D_i$ are connected to $x_i$. Each of these sets have size $f(x_i)$. Analogously, all nodes in $\overline{A_i}, \overline{A_i'}, \overline{D_i}$ are connected to $\overline{x_i}$ and have size $f(\overline{x_i})$.

3. In $A_i$ is one node $m_i$ that is ordered to move to $x_i$. Similarly there is one $m_i' \in A_i'$ ordered to move to $x_i$. Furthermore, both $\overline{A_i}$ and $\overline{A_i'}$ have nodes $\overline{m_i}$ and $\overline{m_i'}$ ordered to move to $\overline{x_i}$. All other nodes in these four groups support the attack of the moving node of their set. Furthermore, $a_i$ supports the attack of $m_i$ on $x_i$ and $\overline{a_i}$ supports the attack of $\overline{m_i}$ on $\overline{x_i}$.

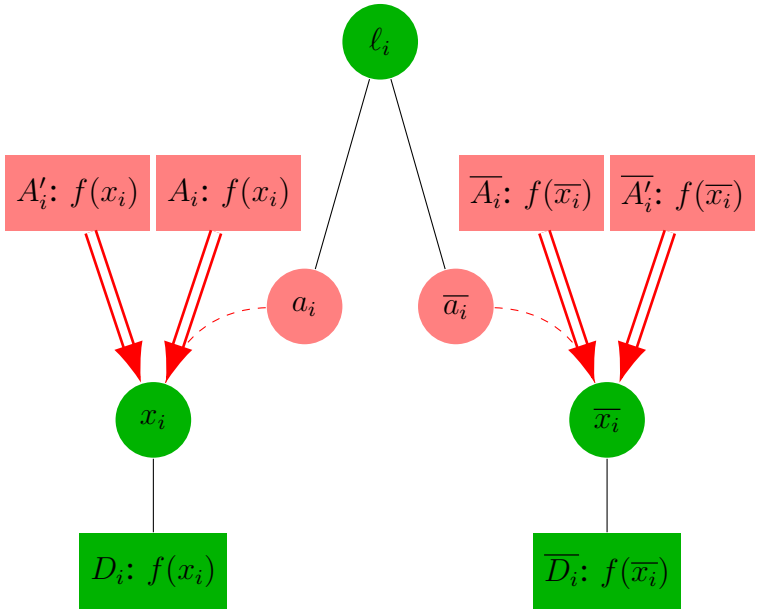


Figure 4.1: Representation of the gadget for variable $x_i$. The green nodes are occupied by the player, the red ones by the opponent. The rectangles denote a collection of provinces, all connected to the node the rectangle is connected with. The amount of nodes in the collection is indicated after the colon. The solid black lines indicate that the nodes connected by them are neighbouring. The dashed red lines indicate attacking support, whereas the double red arrows indicate attacks, in this case of strength $f(x_i)$ or $f(\overline{x_i})$. The other figures in this chapter follow the same representation of neighbourhood and given orders.

See figure 4.1 for a visual representation of the gadget for $\ell_i$.
Each clause $c_j$ also gets a gadget, constructed in the following way.

1. $c_j$ is represented a node currently occupied by the player, which is attacked by some fresh node $a'_j$. This attack is supported by all the literals of the clause, represented by nodes $\ell^j_1, \ell^j_2$ and $\ell^j_3$.

2. Furthermore, there are two nodes connected to $c_j$, occupied by the player.

Furthermore we define $T = \bigcup_{j\in[m]}\{c_j\}$.
See figure 4.2 for a visual representation.

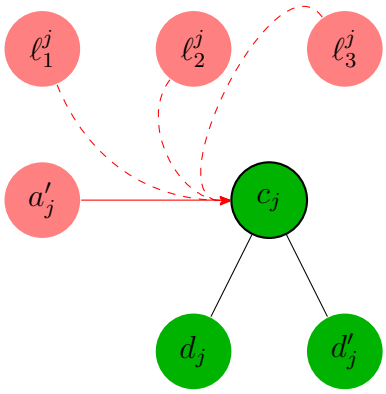



Figure 4.2: Representation of the gadget for clause $c_j$. The solid black border around node $c_j$ indicates that this node is an element of $T$.

The gadgets will be connected in the following natural way: every node in $D_i$ ($\overline{D_i}$) from the literal gadget for $\ell_i$ is connected to one occurrence of the variable $x_i$ ($\overline{x_i}$) in the clause gadgets, where $x_i$ is the variable of $\ell_i$. Each such $x_i$ is thus paired with exactly one element of $D_i$. This concludes the reduction.

The reduction is clearly polynomial, as all components of the gadgets are, and there are polynomially many. For correctness, we show that $\varphi$ is satisfiable iff in the reduction, there is a strategy for the player such that all provinces that currently belong to the player at the beginning of the turn, still belong to the player at the end of the turn.

$\Rightarrow$ Suppose $\varphi$ is satisfiable. Then there is a satisfying assignment $\alpha$ such that $\varphi$ is true under $\alpha$. We define the following strategy $S_\alpha$ as follows. $x_i$, $\overline{x_i}$ and $c_j$ are ordered to hold. $d_j$ and $d'_j$ are ordered to support the holding of $c_j$. Let $\ell_i$ attack $a_i$ if $\alpha(\ell_i) = 1$, or let it attack $\overline{x_i}$ otherwise.

Assume without loss of generality the former. Then, because the support of $a_i$ to $m_i$ is cut, the effective size of the attack of group $A_i$ is the same as the size of the attack from $A'_i$, causing a standoff. Then order all nodes in $\overline{D_i}$ to support the defense of node $\overline{x_i}$. Together with $\overline{x_i}$ holding, this means that the defense is of the size of the largest attack, so $\overline{x_i}$ holds successfully. Now note that there is a surjection $g$ from $D_i$ onto all occurrences of $x_i$ in the clause gadgets. Order $d \in D_i$ to attack $g(d)$.

As $\alpha$ is a satisfying assignment, all clauses $c_j$ have at least one literal $\ell^j$ that is true under $\alpha$. But by construction then in each clause gadget, $S_\alpha$ makes

sure that the support of the node corresponding to $\ell^j$ is cut. But then the size of the defense of $c_j$ is at least the size of the attack on $c_j$, which means $c_j$ is defended successfully.

Hence all holds are successful under strategy $S_\alpha$. Furthermore, all armies that are ordered to move by the player attack with size 1 on some province where the opposing army is ordered to support, which means that their move will not be successful, which means that they remain in the same province they started from.

$\Leftarrow$ Suppose there is a strategy $S$ for the player that ensures all provinces are successfully defended. In particular, this means that every $c_j$ is successfully defended. Hence the support of at least one of $\ell_1^j, \ell_2^j$ or $\ell_2^j$ must be cut. W.l.o.g. assume the support for $\ell_1^j$. We construct a partial assignment $\alpha$ and distinguish two cases:

$(i)$ If $\ell_1^j$ is some positive literal $x_i$, then we know that one unit from the set $D_i$ must attack the node $\ell_1^j$. If we then want the node $x_i$ to be successfully defended, the key observation is that it must happen that the node $\ell_i$ is ordered to attack $a_i$ to cause a standoff at node $x_i$. We then set $\alpha(x_i) = 1$, ensuring that clause $c_j$ has a true literal.

$(ii)$ If $\ell_1^j$ is some negative variable $\overline{x_i}$, with similar reasoning, we get that the unit in node $\ell_i$ must attack $\overline{a_i}$ to cause a standoff at $\overline{x_i}$, so we set $\alpha(x_i) = 0$, again ensuring that clause $c_j$ has at least one true literal.

Observe that this procedure indeed constructs a partial assignment that satisfies $\varphi$: every clause has a literal that is satisfied, as argued for in the case distinction, and no variable is both true and false. The latter holds because there is only one node $\ell_i$ that can attack at most one of $a_i$ or $\overline{a_i}$. Hence $\alpha$ witnesses that $\varphi$ is satisfiable.

□

As a consequence of the above result, one can see that deciding whether a given region can be conquered without information of the other players moves, lies in the complexity class $\Sigma_p^2$; given a successful attack, some nondeterministic algorithm can find that it indeed conquers everything in polynomial time. In section 3.4 we will see that this problem is, perhaps surprisingly, also in $\mathrm{NP}$ and we will show that it is in fact $\mathrm{NP}$-complete. This is mainly due to the fact that when facing more efficient attacks with just one unit making a move on each target, calculating a defense is easier. In the next section this will be expanded on in more detail.

## 4.2 Attacking while knowing the defense

Next, we analyse the complexity with the perspective of the attacking player, looking to conquer some set of provinces. The main question that is under investigation in this section is the following one.

**Definition 19** (P2)**.** Given a game state $GS$, a set of provinces $T$ all belonging to the opponents, and the set $M$ of moves made by all opponents, does there exist

a strategy extending $M$ such that the player successfully conquers all provinces $t \in T$?

Whereas the hardness proof of the previous problem heavily relied on standoffs, as attackers we can try to avoid those. Any other party causing a potential standoff can be viewed as just another defender. It turns out we only need one main attacker per target, which is captured in the following definition and lemma.

**Definition 20** (Simple attack)**.** A strategy by player $p$ is called a *simple attack* on a set of provinces $T$ if for each target $t \in T$ there is exactly one unit of $p$ that is ordered to move to $t$.

We immediately establish the next useful result.

**Lemma 3.** *If there is a perfect successful attacking strategy conquering $T$, then there is a successful simple attack on $T$.*

*Proof.* Consider some perfect successful attacking strategy $S$. For any $t \in T$, let $A_t^S$ be the set of attackers that is ordered to move to target $t$ according to $S$. Now consider any strategy $S'$ that meets the following conditions for all $t \in T$:

1. $A_t^{S'} = \{a_t\}$ for some $a_t \in A_t^S$.
2. All $a'_t \in A_t^S \setminus A_t^{S'}$ support the attack on $t$ by $a_t$.
3. All supporters of any $a'_t \in A_t^S$ support the attack on $t$ by $a_t$.
4. Any other unit is ordered according to $S$.

By condition 1. we have that the attack specified by $S'$ is indeed simple with respect to $T$. As any neighbour of $t$ can support any attacker of $t$, $S'$ is well defined too. We now show that $S'$ is indeed perfect.

Consider any defense $D$ by the opponent and let $t \in T$ be arbitrary. As $S$ is a perfect attack, there is some $a \in A_t^S$ that successfully moves to $t$ with the support of $A_a$. As $A_a$ now supports $a_t$ and the move by $a_t$ cannot be prevented, $a_t$ now has sufficient support to conquer $t$. As $t$ and $D$ were arbitrary, we get that $S'$ is a perfect attack on $T$. □

Finding an optimal attack is not straightforward. One might at first glance think that we just take any attacker to move and let any province that can support this attack do so. Even for one target, it turns out to be not that easy. The figures 4.3 through 4.5 below show that which attacker directly supports and which attacker cuts defensive support, can influence the outcome of an attack.

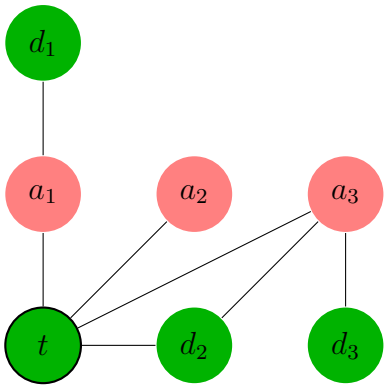



Figure 4.3: An example of a game state. $a_1$, $a_2$ and $a_3$ belong to the attacking party, and $t$, $d_1$, $d_2$ and $d_3$ are occupied by the defenders.

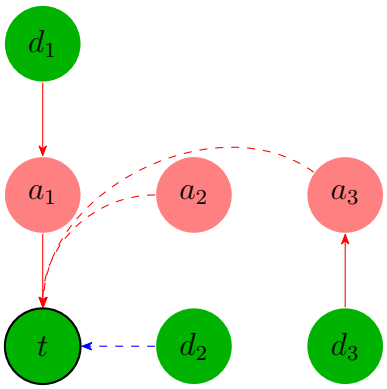



Figure 4.4: Province $t$ holds: it gets direct defensive support from $d_2$. The support from $a_3$ is cut, resulting in a 2 versus 2.

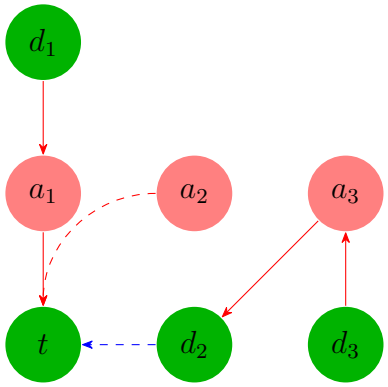



Figure 4.5: Province $t$ is conquered in a 2 versus 1. The defensive support of $d_2$ is cut by $a_3$.

Moreover, we see that the choice of which unit actually moves to a target also might matter: had node $a_3$ moved to $t$ with support of the other attacking nodes, then the attack would have been successful too, regardless of the possible defenses.

Nonetheless, perhaps surprisingly, problem 2 turns out to be in $P$. The key idea to see this is that the defenders only need to successfully defend one target. In order to prove this statement, we make use of flow networks.

**Lemma 4.** *$P2$ is decidable in polynomial time.*

*Proof.* We will construct a flow network and find its maximal flow.

First we partition the attacking provinces $A$ into two groups: $A_1$ and $A_2$. The latter will be moved to by the opponent, where the former will not. We can do so because we assume we know the moves of the opponent. This means that provinces in $A_1$ can deliver any kind of support, whereas $A_2$ will only be able to *move* to contribute to the attack. Furthermore, we denote the set of target provinces with $T$, and all targets and defenders that deliver direct defensive support with $D \supseteq T$. Let $d_t$ be the amount of attackers that is needed to conquer $t$.

The actual $s, s'$-flow network is constructed as follows, where whenever each edge has capacity 1 unless stated otherwise:

1. We add a source node $s$, which we connect with edges to every $a \in A$.
2. Every $a \in A$ is connected to their neighbouring[1] defenders $d \in D \setminus T$ that are not targets.
3. The defenders $d \in D \setminus T$ that deliver defensive support to some $t \in T$ are connected with an edge of capacity 1 to that target $t$.
4. Every $a \in A_1$ is also connected to every target it could possibly attack.
5. For every $t \in T$ we add a node $a_t$, which will have an edge to $s'$. For all $a \in A$ that could be used to directly attack $t$, we add a link to $a_t$.
6. Finally, we add a sink node $s'$, and connect every $t \in T$ to $s'$ with capacity $d_t - 1$.

As the constructed flow network consists of at most the amount of nodes of the original map together with $|T| + 2$ nodes, this construction is polynomial in the size of the input.

To see that the total flow value is $\sum_{t \in T} d_t$ just in case all of $T$ is conquered, observe that each such integral flow can be mapped to a (partial) conquering strategy, in the following natural way.

$\Rightarrow$ Suppose for some integral flow $f$ we have $v_f = \sum_{t \in T} d_t$. We define the attacking strategy as follows.

Let $a \in A$, $t \in T$, and $d \in D \setminus T$. Whenever $f((a, a_t)) = 1$, $a$ is ordered to move to $t$. Observe that as the flow going out of $a_t$ is at most 1, we have exactly one attacker for each target.

When $f((a, t)) = 1$, then $a$ supports the attack on $t$. If $f((a, d)) = 1$, then $a$ is ordered to move to $d$ and thus cut the support provided by $d$.

As each attacker has at most an incoming flow of value 1, every attacker gets at most one order. It is easy to see that these orders are also legal. As each $t$ is attacked by one unit with a support of $d_t - 1$ units, each target is indeed conquered.

$\Leftarrow$ Conversely, any strategy that conquers all of $T$ can be mapped to a maximal flow, by first converting the strategy to a simple one, and then arbitrarily selecting $d_t$ contributing attackers per target $t \in T$.

Therefore there is a winning strategy if and only if the maximum flow is equal to $\sum_{t \in T} d_t$ in the constructed flow network, which makes $P2$ decidable in polynomial time. □

[1] Here it is important to note that the difference between units on sea and units on land does not matter for this problem. With neighbouring we mean reachable for the attacker here. As the defending strategy is already known, the possible asymmetry in this reachability relation makes no difference.

To illustrate the working of the construction above, consider the example from figure 4.6, where $t$ is the target, $d_2$ directly defensively supports $t$ and $d_1$ and $d_3$ move to $a_2$ and $a_3$. respectively. The resulting flow network is depicted in figure 4.7, with the unique maximal flow depicted in figure 4.8.

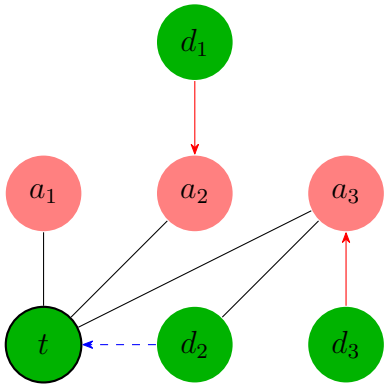



Figure 4.6: The example situation where the strategy of the defending players is known.

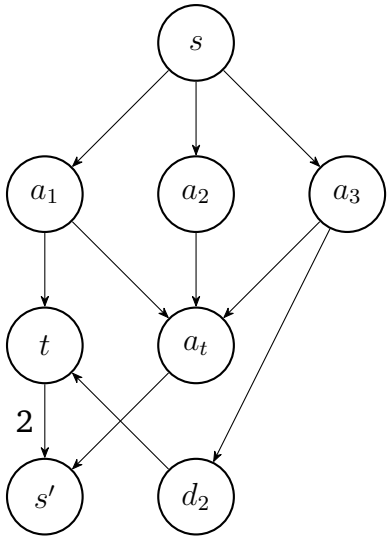



Figure 4.7: The resulting $s$-$s'$ flow network. All edges without label have capacity 1.

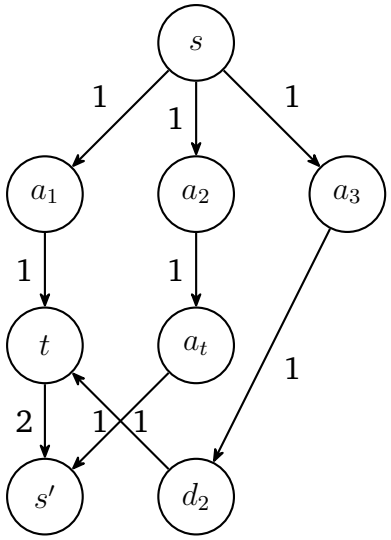



Figure 4.8: The unique maximal flow. The corresponding strategy moves $a_2$ to $t$ with the support of $a_1$, while $a_3$ moves to $d_2$.

At first glance it seems not intuitive that attacking seems less complex than defending. We need to select the right attacker for each target after all, whereas the defending party does not have those choices. In the next section, some more light will be shed on finding an optimal defense. and where this difference is coming from. However, we first look at the following problem, as that result will be used later on as well.

**Definition 21** (P3)**.** Given a game state $GS$, a set of provinces $T$ all belonging to the opponent, and the set $M$ of moves made by all opponents, does there exist a

strategy extending $M$ such that the player successfully conquers any one province $t \in T$?

Observe the crucial difference between problem 3 and problem 2. We are now only interested in being able to defend (at least) one target, while in $P2$ we needed to conquer all of $T$. It turns out $P3$ is in $P$ too.

**Lemma 5.** *$P3$ is in $P$.*

*Proof.* Let $I = (GS, T = \{t_1, t_2, \ldots, t_n\}, M)$ be the input. To decide $I \in P3$, we can for each $i \in [n]$ run the algorithm for $P2$ with input $(GS, \{t_i\}, M)$ to decide whether $t_i$ can be conquered. If any $t_i \in T$ turns out to be conquerable, the instance is in $P3$. Otherwise, it is not. Because we construct at most $|T|$ many polynomially sized flow networks (and find their maximum flow), this whole process takes a polynomial amount of time. □

In the next section we will again focus on the defending side. This time we will not assume anything about the incoming attacks, which more closely represents an actual game situation.

## 4.3 Defending without knowledge of the opponents

As we saw in section 4.1, it is *NP*-complete to decide whether there is a successful defense that keeps all targets, even when given the attacking strategy. This suggests that defending without knowing the attack is at least $\Sigma_P^2$ hard; the question is whether there is **some** defense such that **all** attacks are not successful after all. However, the reduction for Problem 1 heavily relied on the concept of standoffs. As a consequence of lemma 3, when searching for a guaranteed successful attack or defence, we only need to consider simple attacks. When faced with just simple attacks, it is not effective to go looking for a standoff to use possibly less units for the defense of a certain target; all attackers will need to be accounted for now. We have also seen that this is still not too straightforward: in figures 4.3 through 4.5 we have seen that some attackers might need to provide indirect support rather than direct support.

Let us first state the problem of this section formally.

**Definition 22** (P4)**.** Given a game state $GS$, and a set of provinces $T$ belonging to the player, does there exist a strategy $M$ with just moves from this player, such that the player successfully holds all $t \in T$? in all extensions of $M$? △

As any non-simple attack is weakly dominated by some simple attack, when looking for a perfect defense, we only have to consider simple attacks. Perhaps surprisingly, this leads to the following result.

**Lemma 6.** *P4 is* NP*-complete.*

*Proof.* For membership in *NP*, let the certificate $c$ be a set of moves made by the player. We need to check that these moves hold all $t \in T$ against any possible attack. This can be done in polynomial time by using the algorithm $A$ described in lemma 5 and outputting yes if $A$ accepts, and rejecting otherwise.

To show hardness, We again reduce from *3-SAT*. Let $\varphi$ be a formula in 3-CNF with clauses $c_1, c_2, \ldots, c_m$ and variables $x_1, x_2, \ldots, x_n$. Let $C$ denote the set of clauses. There will be two players, but due to a technicality in the rules of the game[2], we also make use of some unoccupied provinces.

For each variable $x_i$ where $i \in [n]$ we introduce the following gadget. Let $d_i$ be a node belonging to the player, and connect it to two fresh nodes nodes $x_i$ and $\overline{x_i}$, both belonging to the opponent. Furthermore, we add two nodes $y_i$ and $z_i$ connected to both $x_i$ and $\overline{x_i}$. Node $y_i$ belongs to the opponent, and node $z_i$ will be unoccupied.

For each clause $c_j$ where $j \in [m]$, we also introduce a gadget. We construct nodes $a_j$, $c_j^1$, $c_j^2$. We also add node $c_j^3$ when $c_j$ has 2 unique literals, and we add $c_j^3$ in case the clause has 3 different literals. Of these nodes, only $a_j$ is occupied by the opponent; the rest belongs to the player. We connect $a_j$ to all nodes, and, when present, $c_j^3$ and $c_j^4$ to $c_j^2$. For a representation of both gadgets, see figures 4.9 and 4.10.

Now connect the gadgets by introducing an edge between nodes $x_i$ and $c_j^2$ iff $x_i$ positively occurs in clause $c_j$. Analogously, connect $c_j^2$ with $\overline{x_i}$ just in case $x_i$ appears negatively in $c_j$. We define helper functions $f : C \to \mathbb{N}$, assigning to each clause the amount of unique literals it has, and $g : C \to \wp(\mathbb{N})$ as $g(c) = \{k + 1 \mid k \in [f(c)]\}$. Let $T = \{c_j^k \mid j \in [m], k \in g(c_j)\}$. This concludes the reduction.

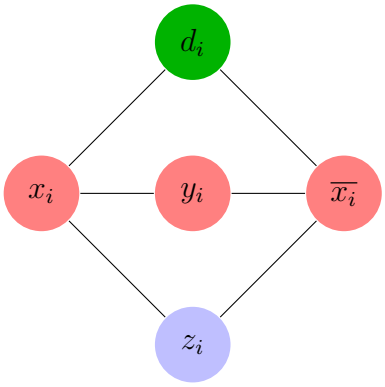



Figure 4.9: A gadget for variable $x_i$. Node $z_i$ is an unoccupied province.

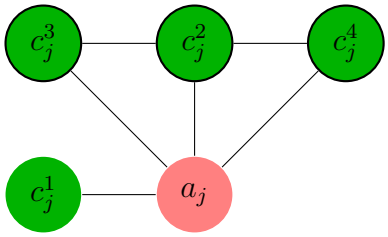



Figure 4.10: A gadget for a clause $c_j$ with 3 unique literals

As we introduced a constant amount of nodes for each clause and for each variable, the reduction is polynomial in the amount of clauses of the formula.

Next, we show the correctness of the reduction.

$\Rightarrow$ Suppose $\varphi$ is satisfied by some assignment $\alpha$. We construct a perfect defense as follows. For each $j \in [m]$, let the unit on $c_j^1$ move to $a_j$, and order both $c_j^3$ and $c_j^4$ to defensively support the holding $c_j^2$ where relevant. This way each node $c_j^2$ has a defense of strength equal to the amount of literals of clause $c_j$. Therefore, to lose any such $c_j^2$, all nodes corresponding to the literals in $c_j$ must help the attack on that province. As $a_j$ is being moved to, for it to contribute to the attack, it must move to $c_j^2$ too, making it so that all nodes corresponding to the literals need to deliver attacking support in order to conquer any $c_j^2$. But as $\varphi$ is satisifiable, we can order $d_i$ to $x_i$ if $\alpha(x_i) = 1$, and to $\overline{x_i}$ otherwise. As every clause has a literal that is true under $\alpha$, at least one

[2]The relevant rule here is that a player is not allowed to support an attack on their own unit.

potential support is cut for each $c_j$, so the defense is successful against every incoming attack. Hence, at the end of the turn, the player will occupy every node in $T$.

$\Leftarrow$ Suppose there is a perfect defense $D$ for $T$. I.e. for all $j \in [m]$, no matter the moves of the other players, the provinces $c_j^2$, $c_j^3$ and $c_j^4$ are occupied by some unit of the player at the end of the turn. We reason that for any $j \in [m]$, $c_j^2$ then cannot move to any literal node according to $D$.

> Suppose $c_j^2$ moves to some literal node $x_i$. The case for $\overline{x_i}$ is completely analogous.
>
> Then $x_i$ can move to the unoccupied $z_i$, so that $y_i$ can support this attack. Any support to attackers of $x_i$ other than $c_j^2$ can be cut in the following way. Support from $d_i$ can be stopped by $\overline{x_i}$, and support from other $c_k^2$ can be cut by $a_k$. Now the attack on $x_i$ is successful, making $c_j^2$ move to province $x_i$. But now there are only $f(c_j) - 1$ units of the player that need to occupy the $f(c_j)$ provinces (in the case of 3 unique literals, these are $c_j^2$, $c_j^3$ and $c_j^4$), which is not possible. Hence $D$ is not a perfect defense in this case. $\triangle$

As $D$ is a perfect defense, it works against any possible attack, in particular one where each $a_j$ moves to $c_j^2$. As any $c_j^2$ can be defended with a strength of at most 3, that is with the direct defensive support of $c_j^3$ and $c_j^4$, this means that for each clause, at least one literal must be stopped from being able to give attacking support. As this stopping can only be done by nodes $d_i$ with $i \in [n]$, we get a satisfying partial assignment for $\varphi$ in a natural way. We let $\alpha_D(x_i) = 1$ iff $d_i$ is ordered to move to $x_i$ according to $D$, and let $\alpha_D(x_i) = 0$ otherwise. Now clearly any extension of $\alpha_D$ is a satisfying assignment for $\varphi$. For every clause, at least one literal node is being moved to by some $d_i$, so this literal is set to true by the assignment. As we also have that for each variable $x_i$, node $d_i$ can only move to one of $x_i$ and $\overline{x_i}$, the assignment is well defined too.

$\square$

So, finding a perfect defense for an arbitrary set of provinces is NP-complete. In the next section, we will look at the same problem from the perspective of the attackers. In section 4.5 we will revisit this proof to ensure the same complexity result when restricting ourselves to planar graphs.

## 4.4 Attacking without knowledge of the opponents

In this section, we will go into the complexity of finding a perfect attack given an arbitrary set of provinces $T$. To remind the reader, that given a set of provinces is an attack that conquers all of $T$ against all possible defenses. Selecting an appropriate target set is up to the player and depends on their strategy in game. For example, it could be important to take provinces with supply centers, or get close to them. For the sake of this thesis, we will not assume anything about the target set. We first formulate the main problem of this section.

**Definition 23** (P5)**.** Given a game state $GS$, and a set of provinces $T$ belonging to the opponent, does there exist a strategy $M$ with just moves of the player, such that the player successfully conquers all $t \in T$ in all possible extensions of $M$?

As one might expect, this problem too turns out to be *NP*-complete. In order to show membership in *NP*, we first establish this useful lemma.

**Lemma 7.** *Given a game state, a set $T$ of provinces belonging to the player, and simple attack by the opponent on $T$, it is decidable in polynomial time whether at least some $t \in T$ can be defended successfully.*

*Proof.* Given some simple attack, we do the following. For each $t \in T$, we are going to construct a flow network to see whether $t$ can be held.

Because the attack is known, we can divide the possible defenders up into two disjoint groups. $D_1$ will be all defenders that are not attacked and can reach $t$ in one move, together with $t$ itself. $D_2$ will be all defenders that are not in $D_1$, but are able to move to some support supplying attacker.

Observe that for $D_1$, it is always good to provide direct defensive support to $t$ whenever we only care about $t$. This way they do not block other defenders from cutting support, and still take care of one unit of attacking power.

The flow network is then constructed as follows: the set of nodes consists of a source $s$, $D_2$, $t$, and the group of support supplying attackers $A$. The source $s$ will have a direct edge to $t$ with capacity $|D_1|$, and an edge with capacity 1 to each $d \in D_2$. Each such $d$ will be connected to each neighbouring support supplying attacker in $A$. All $a \in A$ will have an edge of capacity 1 to $t$. For an example, see figures 4.11, 4.12 and 4.13 below.

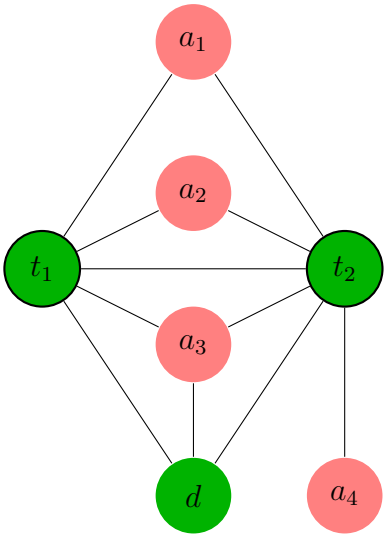



Figure 4.11: An example map

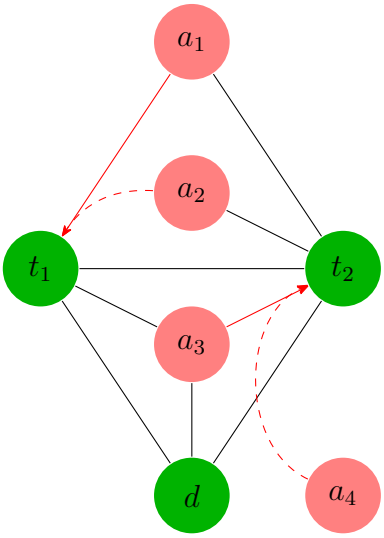



Figure 4.12: A simple attack on $\{t_1, t_2\}$ where $a_1$ moves to $t_1$ with the support of $a_2$, whereas $a_3$ moves to $t_2$ with the support of $a_4$.

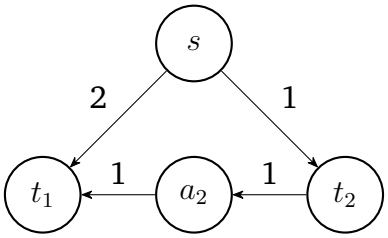



Figure 4.13: The resulting $s$-$t_1$ flow network. The maximal flow $f$ is given by $f(e) = c(e)$. As $3 = v_f \geq |A| = 2$, $t_1$ can be held.

We then obtain the following result: $t$ can be held just in case the value of a maximal flow is at least $|A|$. Note that as all capacities are integral, we get that there is an integral maximal flow too (see section 2.3). This flow can then naturally be translated to a defending strategy: all $d_1 \in D_1$ will deliver direct defensive support to $t$. $d_2 \in D_2$ will move to some $a \in A$ whenever there is flow from $d_2$ to $a$. Any other unit owned by the player can hold. As an $s$-$t$ flow maximises the amount of defenders for $t$, if the value of the maximal flow is less than $|A|$, we get that $t$ cannot be held. Observe that the constructed flow networks are instances maximal matching in a bipartite graph, with one extra arc from $s$ to $t$.

□

This immediately gives us a proof for membership of *NP*.

**Corollary 8.** *P5 is in NP.*

*Proof.* Given a game state and a set of provinces, use a simple attack as a certificate and use the algorithm from lemma 7 with the aforementioned game state, set of provinces and simple attack as input. If no province can be held, then all are successfully conquered. □

Now we are ready to show *NP*-hardness too.

**Lemma 9.** *P5 is NP-hard.*

*Proof.* We reduce from *3SAT* again, with a similar construction as before. Let $\varphi$ be a 3-CNF formula with clauses $c_1, c_2, \ldots, c_m$ and variables $x_1, x_2, \ldots, x_n$.

For each variable $x_i$, we introduce a gadget consisting of three nodes. A node $a_i$ occupied by the attacker, which will be connected to nodes $x_i$ and $\overline{x_i}$, representing the positive and the negative literals. These will be occupied by the player.

For each clause $c_j$, we will introduce a 6-node gadget. Two vertices will be occupied by the player: nodes $d_j$ and $c_j$. There is a node $a_j^1$ occupied by the attacker, connected to both of these provinces. Furthermore, there are nodes $a_j^2, a_j^3$ and $a_j^4$ connected to $c_j$ whenever there are three unique literals in clause $c_j$. If there are only two, $a_j^4$ is left out. If there only one unique literal in the clause, then both $a_j^3$ and $a_j^4$ will be removed. We connect $c_j$ to each $x_i$ whenever $x_i$ occurs positively

in $c_j$, and we connect $c_j$ to $\overline{x_i}$ whenever $x_i$ occurs negatively in the clause. Let $T = \bigcup_{j \in [m]} c_j$. For a visiual representation of the gadgets, see figures 4.14 and 4.15.

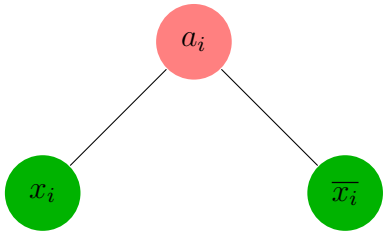



Figure 4.14: A gadget for variable $x_i$.

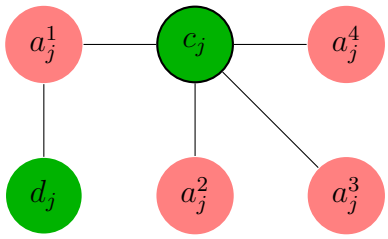



Figure 4.15: A gadget for some clause $c_j$ with 3 unique literals. Node $c_j$ is an element of $T$.

As we introduced a constant amount of nodes for each literal, and the amount of nodes introduced for each clause is bounded by 6 too, the reduction can be constructed in a polynomial amount of time. What is left to do is argue for the correctness of the reduction.

$\Rightarrow$ Suppose there is some satisfying assignment $\alpha$ for the formula $\varphi$. We construct the following attacking strategy. Move $a_j^1$ to $c_j$ with the support of $a_j^2$, $a_j^3$ and $a_j 4$. Move $a_i$ to $x_i$ whenever $x_i$ is true according to $\alpha$, and move $a_i$ to $\overline{x_i}$ otherwise. As $\varphi$ is true, each clause has at least one variable that is now under attack. That means that each $c_j$ has at most $k$ possible defenders left ($c_j$ itself and up to two other literals), which is not enough to withstand the incoming $k+1$ attacking nodes. Hence all $c_j$ are conquered, as desired.

$\Leftarrow$ Suppose $\varphi$ is unsatisfiable. Then no matter where each $a_i$ moves to, there will always be a clause $c_j$ such that all of its literals are not under attack. That means that all of these $k$ literals are free to provide $c_j$ with direct defensive support. Then $c_j$ has $k+1$ defenders (including $c_j$ itself), and an equal amount of attackers. Hence $c_j$ can be successfully defended, which means that there is no perfect attack whenever $\varphi$ is unsatisfiable.

□

**Corollary 10.** *P5 is NP-complete.*

*Proof.* Immediate from the previous two lemmas. □

With this proof, we have all the main results of this thesis. In the next section, we will see that the NP-hardness results still hold when restricted to planar graphs. In the discussion section we delve a bit into the area between having a perfect defense and having a perfect attack.

## 4.5 Working with planar graphs

As the game Diplomacy is typically played in a world map that is divided into provinces, it makes sense for us to look at how restricting ourselves to planar

graphs only affects the complexity of the researched problems. As the problems that were shown to be in *(N)P* were solved by algorithms that work on any graph, it is clear to see that those results still hold. For the *NP*-hardness proofs however, When reducing from any *3-SAT* formula, we have no guarantee at all that the resulting graph is in fact planar. In this section we will show that we do not lose any of the complexity results when working with the restricted class of planar graphs representing the game map. To do so, we introduce *planar monotone 3-SAT*, a variant of *3-SAT*, first presented in [BK10].

**Definition 24** (Planar monotone *3-SAT*)**.** Let $\varphi$ be a 3-CNF formula over a set of variables $U := \{x_1, \ldots, x_n\}$. Let $C := \{c_1, \ldots, c_m\}$ denote the set of clauses of $\varphi$. Then $\varphi$ is called *monotone* if each clause either contains only positive variables, or only contains negative literals. In [Har82] it is shown that deciding satisfiability of monotone 3-CNF formulas is still NP-complete.

A 3-CNF formula $\varphi$ is called *planar* if the bipartite graph $G = (U \cup C, E)$ where $E := \{(x, c) \mid x \text{ occurs in } c\}$ can be drawn without crossing edges. A well known result by Lichtenstein [Lic82] is that deciding satisfiability of planar formulas remains NP-complete[3] and . Furthermore, Knuth and Raghunatan [KR92] have shown that each such formula has a rectilinear representation (see figure 4.16 for an example of a formula in rectilinear representation): The variables and clauses are all drawn as rectangles. The variables are drawn on one horizontal line, the clauses are drawn above or below the variables, and the edges representing that a variable is part of a clause are straight vertical lines. Moreover, the drawing is crossing-free.

A *monotone rectilinear representation* of a monotone 3-CNF formula is a rectilinear representation where each clause with only positive formulas is drawn above the variables, and each clause with just negative variables is drawn below the variables in the middle. △

[3]At first glance, this result looks like a good starting point for our problem. However, with the current gadgets we introduced in lemmas 2, 6 and 9, we need to separate the positive and negative instances of variables too.

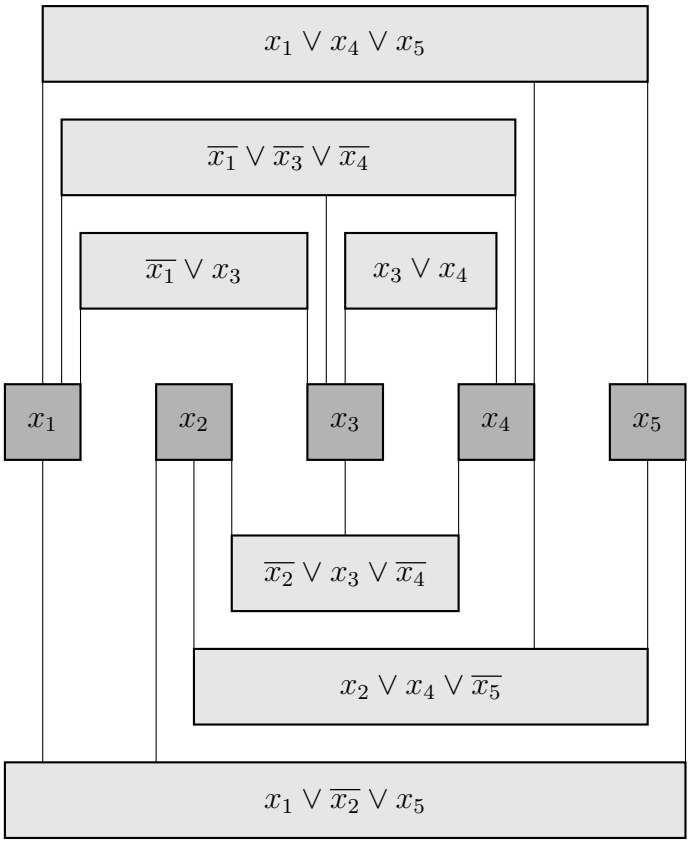



Figure 4.16: A rectilinear representation of a formula in 3-CNF. The darker shaded nodes represent the variables, and the lighter shaded rectangles represent the clauses.

The corresponding decision problem of this *3-SAT* variant is then formulated as follows:

**Definition 25** (Planar monotone 3-SAT satisfiability problem)**.** Given a monotone 3-CNF formula $\varphi$ and a corresponding monotone rectilinear representation of $\varphi$, is the formula satisfiable? △

De Berg and Khosravi have shown that this problem too is *NP*-complete [BK10]. With this result we can show that finding a perfect defense, a perfect attack, or an optimal defense against any given attack still remains *NP*-hard. As the correctness of the reductions is shown for any 3-CNF formula, they will still work for formulas with a monotone rectilinear representation. What is left to show is that the product of these reductions result in planar graphs. In order to do so in one go, we introduce the following abstraction of the introduced gadgets.

**Definition 26** (Battery variable gadget)**.** A *battery variable gadget* is a graph gadget for a variable with the following properties:

1. The gadget can be drawn as a planar graph.

2. The part of the gadget that is connected to positive occurrences of the variable can be drawn as a horizontal line of nodes. Analogously, the nodes that are connected to the negative occurrences of the variable can be drawn in horizontal a line. These two lines can be drawn on the outside of the gadget and do not cross.

3. Let $a, b$ be two elements in one such line. Then if $a$ is connected to $x$ and $b$ is connected to $y$ where $x, y$ are in the clause gadgets and the reduction is correct w.r.t. satisfiability, then, whenever $a$ is connected to $y$ and $b$ is connected to $x$ instead, we still have a valid reduction regarding satisfiability.

The name of this abstraction refers to a way of representing the variable gadget like a battery. We get a positive side, a middle piece, and a negative side. The third condition is there to make sure it does not matter which node in the literal gadget is paired with which occurrence.of the variable in the clause gadgets.

**Definition 27** (Single receiver clause gadget)**.** A *single receiver clause gadget* is a graph gadget for a clause with the following properties:

1. The gadget can be drawn as a planar graph.
2. The part of the gadget that is connected to the variable gadgets is on the outside, and can be drawn as a horizontal line of nodes. The order in this line has no influence on the correctness of the reduction w.r.t. satisfiability, and no influence on the internal planarity of the gadget.
3. There is at most one edge going out of the gadget for each variable in the clause.

**Lemma 11.** *When starting from a planar monotone 3-CNF formula and using just gadgets that fit the criteria from Definitions 26 and 27, there is a reduction that results in a planar graph.*

*Proof.* Consider the monotone rectilinear representation of a monotone 3-CNF formula. We can now substitute the variable rectangles in the representation by the battery variable gadget. The positive plate of the battery can be placed on the top of the rectangle, and the negative side on the bottom. The nodes can be aligned with the outgoing edges in the original representation.

The clause rectangles can be replaced by a clause gadget within the rectangle of the corresponding clause, with the receiving line facing the variable gadgets. By conditions 2 and 3 of definition 12, the size and order of the line of nodes that are connected to the variables can be readjusted so that the edges of the original rectilinear representation line up with these nodes.

Connecting the variable gadgets with the clause gadgets is then done by using the original vertical lines from the monotone rectilinear representation between the variables and clauses, and hence the resulting graph is planar. □

So with some copy pasting and adjusting the size of some nodes, we can make use of planar graphs.

**Corollary 12.** *The complexity results of lemmas 2, 6 and 9 hold when restricting the game to planar maps only.*

*Proof.* For the gadget in 4.1, observe that it is indeed planar, and that the order in $D_i$ and $\overline{D_i}$ does not matter. As both of these groups are on the outside of the gadget, one can be placed above, and the other below. Hence this variable gadget is a battery variable gadget. Observe that the clause gadget 4.2 has three nodes that must be connected to the variable gadgets, and the order of those nodes does

not matter. Hence the NP-completeness result of lemma 2 holds for planar graphs too. It is easy to see that the gadgets used in Lemmas 6 and 9 fit the requirements of Definitions 26 and 27 too. □

## 4.6 Overview of Results

An overview of the established results can be found in the following table.

| | some $t \in T$ | all of $T$ |
|---|---|---|
| Defense with knowledge of attack (any) | - | NP-complete |
| Defense with knowledge of attack (simple) | P | - |
| Defense without knowledge of the attack | - | NP-complete |
| Attack with knowledge of the Defense | P | P |
| Attack without knowledge of the defense | - | NP-complete |

Table 4.1: An overview of the established complexity results

# Chapter 5

# Conclusion and Future work

## 5.1 Conclusion

In this thesis, we provided a generalisation for the game Diplomacy. We remained close to the original game: the rules of the original game are directly applicable to the generalised version. The exact starting positions, supply centre locations and winning locations are left open for discussion. We chose to make the generalisation as general as possible.

We provided a mathematical framework for the orders and the move resolution, and then constructed a detailed polynomial-time deterministic algorithm that resolves the submitted strategies, which was not done before as far as the author is aware[1]. We then used this algorithm as a subroutine to determine the complexity classes of some of the natural questions in this game.

First, in Section 4.1, we established that finding a defense that holds every province in a given region is NP-complete, even when we are given the attacking strategy. The main difficulty lies in the cases where the defense could make use of standoffs in the game.

Afterwards in Section 4.2, we defined the concept of a simple attack and used that to show that when given a set of provinces and a defensive strategy, it is decidable in polynomial-time, using a constructive proof, whether all or some provinces of that region can be conquered.

In Section 4.3 we showed that constructing a perfect defense given a region and a game state is NP-complete. This may be surprising in light of the result of Section 4.1, as one might expect that the adding another unknown factor should increase the possible solution space. That would lead one to expect that this problem is $\Sigma_p^2$-hard. This difference in expectation and reality can be explained by the fact that it can be hard to find an optimal defense against a non-optimal attack.

When faced with an optimal attack, there is a certain amount of units you need to use in order to stop the incoming threat. When opposed by a non-optimal attack, one might need to use fewer units. These spare units can then be used elsewhere, to help a different province. This creates more possibilities for strategies, which makes the search space for a solution larger. When one aims to find a perfect defense, one needs to only prepare against the most optimal attacks. In Section 4.4 we showed that when faced with a simple attack and a region, it is decidable in

[1] After writing down the whole algorithm, the author has also seen why this has not been done in full detail before.

polynomial-time whether at least one province from that region can be defended successfully. Using that result we have shown that it is NP-complete to decide whether there exists a perfect attack against any given region.

Finally, in Section 4.5, we established that all results still hold when we restrict the game to a planar graph, using a suitable variant of the well known *3-SAT* problem.

All in all, this thesis contributes more decision problems to the vast collection of NP-complete problems.

## 5.2 Future work

The framework also invites researching more of these questions. For example, one could look at variants of the problems presented here, like establishing what it takes to conquer or hold at least $k$ provinces in a given region $T$. For the attacking variant for example, the question then becomes to find out whether there is an attack such that for any defense, we can find a subset $T' \subseteq T$ of size at least $k$ such that all of $T'$ is conquered. If the universal quantifier was not in this question, the problem would clearly be in NP. If we compare this to the original question of whether there is an attack such that for any defense, all of $T$ is conquered, we see the same universal quantifier, but it does not affect the complexity: the problem is still in NP.

This framework also lends itself for game theoretic research. Deciding on an optimal defense when there is no perfect defense for example, (see Figure 5.1); or figuring out strategies for groups of players. Finding out which players you should form a coalition with to achieve your in game goals also poses various challenges.

Finally, it would also be interesting to try to characterise which configurations of game states lead to easy to find perfect defenses or perfect attacks, and look for phase transitions [CKT91] in the game.

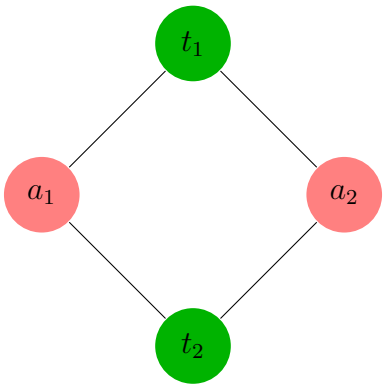



Figure 5.1: If the green player wants to hold both $t_1$ and $t_2$, and the red player want to conquer at least one of them, then it is not sufficient for the green player to hold in both provinces. Instead, they should try to move to one of $a_1$ or $a_2$ to have a chance.

# Bibliography


[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[Ant+20] Thomas Anthony et al. "Learning to Play No-Press Diplomacy with Best Response Policy Iteration". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 17987–18003. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/d1419302db9c022ab1d48681b13d5f8b-Paper.pdf.

[Bak+21] Anton Bakhtin et al. *No-Press Diplomacy from Scratch*. 2021. arXiv: 2110.02924 [cs.LG]. URL: https://arxiv.org/abs/2110.02924.

[Bak+22] Anton Bakhtin et al. *Mastering the Game of No-Press Diplomacy via Human-Regularized Reinforcement Learning and Planning*. 2022. arXiv: 2210.05492 [cs.GT]. URL: https://arxiv.org/abs/2210.05492.

[BK10] Mark de Berg and Amirali Khosravi. "Optimal Binary Space Partitions in the Plane". In: *Computing and Combinatorics*. Ed. by My T. Thai and Sartaj Sahni. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 216–225. ISBN: 978-3-642-14031-0.

[CKT91] Peter C. Cheeseman, Bob Kanefsky, and William M. Taylor. "Where the Really Hard Problems Are". In: *International Joint Conference on Artificial Intelligence*. 1991. URL: https://api.semanticscholar.org/CorpusID:6634061.

[Cor+01] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937. URL: http://www.amazon.com/Introduction-Algorithms-Thomas-H-Cormen/dp/0262032937%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0262032937.

[FAI+22] Meta Fundamental AI Research Diplomacy Team (FAIR)† et al. "Human-level play in the game of Diplomacy by combining language models with strategic reasoning". In: *Science* 378.6624 (2022), pp. 1067–1074. DOI: 10.1126/science.ade9097. eprint: https://www.science.org/doi/pdf/10.1126/science.ade9097. URL: https://www.science.org/doi/abs/10.1126/science.ade9097.

[Har82] Juris Hartmanis. “Computers and Intractability: A Guide to the Theory of NP-Completeness (Michael R. Garey and David S. Johnson)”. In: *SIAM Review* 24.1 (1982), pp. 90–91. DOI: 10.1137/1024022. eprint: https://doi.org/10.1137/1024022. URL: https://doi.org/10.1137/1024022.

[Jac+22] Athul Paul Jacob et al. “Modeling Strong and Human-Like Gameplay with KL-Regularized Search”. In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 17–23 Jul 2022, pp. 9695–9728. URL: https://proceedings.mlr.press/v162/jacob22a.html.

[KR92] Donald Ervin Knuth and Arvind U. Raghunathan. “The Problem of Compatible Representatives”. In: *SIAM J. Discret. Math.* 5 (1992), pp. 422–427. URL: https://api.semanticscholar.org/CorpusID:9974756.

[Kra+22] János Kramár et al. “Negotiation and honesty in artificial intelligence methods for the board game of Diplomacy”. In: *Nature Communications* 13.1 (Dec. 2022), p. 7214. ISSN: 2041-1723. DOI: 10.1038/s41467-022-34473-5. URL: https://doi.org/10.1038/s41467-022-34473-5.

[Lic82] David Lichtenstein. “Planar Formulae and Their Uses”. In: *SIAM J. Comput.* 11 (1982), pp. 329–343. URL: https://api.semanticscholar.org/CorpusID:207051487.