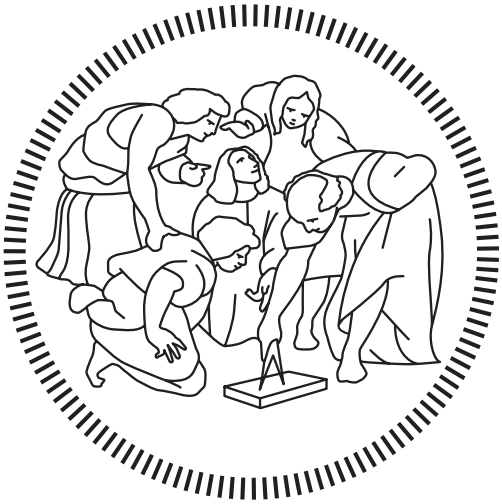
Politecnico di Milano
SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science – Computer Science and Engineering



Implementation of algorithm for optimal ex ante persuasive signaling scheme in Bayesian Network Congestion Games

Supervisor: **Prof. Nicola Gatti**
Co-Supervisor: **Dott. Matteo Castiglioni**
Co-Supervisor: **Dott. Alberto Marchesi**

Candidate
Edoardo Disarò – 920069

Academic Year 2019 – 2020

Ringraziamenti

Vorrei ringraziare il professor Nicola Gatti per la sua cortesia e disponibilità durante tutto lo sviluppo del lavoro. Ringrazio inoltre Matteo e Alberto, per il loro continuo sostegno e i numerosi consigli che mi hanno aiutato a migliorare e completare la tesi.
Un ringraziamento va inoltre a tutte le persone che mi hanno accompagnato in questi anni e che mi sono state vicine durante il percorso universitario.
Un ringraziamento speciale agli amici e compagni di viaggio incontrati durante il semestre in Norvegia presso l' NTNU.
Infine ringrazio la mia famiglia, in particolare mia madre, che mi ha sempre sostenuto con immensa pazienza.

Sommario

I Bayesian Network Congestion Game (BNCG) sono un modello della Teoria dei Giochi adatto a rappresentare le interazioni tra una moltitudine di agenti. Applicando l'ambiente Bayesiano al ben noto modello di Congestion Game, i BNCG descrivono situazioni del mondo reale in cui lo stato della rete è sconosciuto e determina i costi sostenuti dai giocatori. Consideriamo il caso in cui sia presente un giocatore che può svelare informazioni sullo stato di natura agli altri giocatori. Questo modello è noto come persuasione Bayesiana. Il nostro obiettivo è studiare il problema del calcolo di uno schema di segnalazione ottimale ex ante persuasivo in BNCG simmetrici con costi affini. Sotto questi presupposti, è stato dimostrato che il problema è calcolabile in un tempo polinomiale ed è stato sviluppato un algoritmo. Lo scopo del presente lavoro è dunque analizzare in dettaglio l'algoritmo, proponendo un'implementazione efficiente nel linguaggio di programmazione Python ed analizzando dal punto di vista sperimentale le prestazioni dell'algoritmo in varie istanze create sinteticamente.

Abstract

Bayesian Network Congestion Games (BNCGs) are a model used in Game Theory to represent multi-agent interactions. Applying the Bayesian Framework to the well-understood model of network congestion game, BNCGs describes real-world situations in which the state of the network is unknown and determines the costs incurred by the players. We consider the case where there is a player who can reveal information about the state of nature to the other players. This model is known as Bayesian persuasion. Our goal is to study the problem of computing an optimal *ex ante* persuasive signaling scheme in symmetric BNCG with affine costs. Under these assumptions, the problem has been proved to be computable in polynomial time and an algorithm was developed. The aim of the present work is to study the framework and analyze the algorithm, proposing an efficient implementation in the Python programming language and experimentally analysing the performance of the algorithm in various synthetically created instances.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we present a general overview of the research field and the main contribution of this work.

1.1 General overview

Game Theory provides a theoretical framework to formulate, structure, analyse and eventually understand different strategical scenarios. The interaction among agents is formalized as a *game*, an abstract formal description of a strategic situation in which players behave according to certain rules. Game Theory as a discipline has its historical origin in 1944 with the publication of the book *Theory of Games and Economic Behavior* by John von Neumann and Oskar Morgenstern. Since then, the applications of Game Theory have increased, allowing us to describe situations and problems relating to economics and mathematics as well as in biology, political science, computer science and philosophy.

Congestion Games are a classical type of games studied in Game Theory and proposed by the economist Robert Rosenthal in 1973, with the aim of describing strategic behavior of players in choosing among available resources. They can model different situations as processor scheduling, routing, and network design. In the case in which the resources correspond to simple paths in a graph, *e.g.* representing routing options from a source to a target, we are talking about Network Congestion Games. Congestion Games have good computational properties, admitting in many cases algorithms requiring polynomial time.

Information structure design studies the effects of information on the outcomes of strategic interactions. In this context the most important model is the Bayesian Persuasion framework introduced by Kamenica

and Gentzkow [6]. This framework involves an informed *sender* trying to influence the behavior of one or multiple self interested players, the *receivers*. The sender is able to commit to a policy called *signaling scheme* which is a mapping from network states to action recommendations, *i.e.* route suggestions. Since their development, Congestion Games have been a powerful tool and an active area of research but they show limits in describing real-life situations. We introduce Bayesian Network Congestion Games (BNCGs) that capture uncertainty in Network Congestion Games. Finally, we present an algorithm developed to compute an optimal *ex ante* signaling scheme in BNCGs in polynomial time, under specific assumptions.

1.2 Contributions

In the present work, we study the Bayesian framework applied to the game theory model of the Network Congestion Game. Specifically, we are interested in the computation of an optimal signaling scheme in Bayesian Network Congestion Games. We focus on the notion of *ex ante* persuasiveness where the receivers are incentivized in following the signal by only observing the signaling scheme. We also consider symmetric BNCGs, in which all the players share the same source and destination pair and we assume BNCGs have affine costs function, *i.e.* the cost of each edge in the graph can be decomposed in a linear function of the number of players passing through the edge.

In our work, we present and analyze the algorithm designed by Castiglioni, Celli, Marchesi and Gatti [4], which exploits the Ellipsoid Method in order to compute an optimal *ex ante* persuasive signaling scheme in polynomial time for a symmetric BNCG, with affine costs. Therefore, in the light of recent work, we can present the implementation of the algorithm we accomplished and discuss the results we obtained. The implementation of the algorithm was carried out by studying and considering the various steps involved and dealing with mathematical tools and formulations such as the *ellipsoid method*, the *row generation* algorithm and the *integer min-cost problem*. The crucial point is that the ellipsoid algorithm is not usable in practice. Therefore, the question is if, in practice, it is still possible to have an efficient algorithm. Finally, we can empirically evaluate the algorithm as different parameters change and thus verify its efficiency, in particular considering the computational time taken in different game instances. This makes it possible to assess the importance of different parameters and their weight on the computational time. We performed the experiments from a dataset of random

Bayesian Network Congestion Games we implemented, considering networks with 30, 60, 90 and 120 nodes. The number of players considered in the games varies between 10 and 100 at intervals of 10, while the number of states of nature varies between 10 and 40 at intervals of 10. To sum up, we generated and resolved in total 1600 Bayesian Network Congestion Games, setting a time limit of 3600 seconds for each game instance. The programming language used is Python and the Linear Programming problems are solved with Gurobi [9] optimization solver, invoked in our code through the Python module `gurobipy`.

1.3 Structure of the thesis

The work is organized into different chapters as follows.

Chapter 2 introduces the theoretical starting point necessary to understand our work. A brief summary of Computational Complexity Theory is presented. Most of the chapter is then dedicated to Congestion Games and Bayesian Persuasion framework.

Chapter 3 presents the description and formalization in mathematical terms of the research problem faced in our work. Specifically, a formal definition of Bayesian Network Congestion Game is presented, along with the assumptions considered.

Chapter 4 exposes in detail the algorithm developed in order to compute an optimal *ex ante* persuasive signaling scheme in BNCGs in polynomial time.

Chapter 5 illustrates the process we follow to implement the algorithm for computing an optimal *ex ante* persuasive signaling scheme in BNCGs in polynomial time.

Chapter 6 presents and discuss the experimental results we obtained.

Chapter 7 concludes the work with a summary of the results obtained and the proposals for future discussions.

Chapter 2

State of the Art

This chapter analyses all the concepts and definitions necessary to entirely understand the work carried out in this paper. At first, a brief introduction to *Computational Complexity Theory* is presented. Then, the general framework of *Game Theory* is introduced, analysing its basic concepts and notions and focusing our attention to the class of *Congestion Games*. Information structure design is subsequently presented, focusing on the *Bayesian Persuasion* framework. The Bayesian Persuasion framework is the theoretical heart of our work and we analyze in detail the various models and main solutions provided by the literature. Finally, we describe the mathematical tools and algorithms we need to approach the problem in this work.

2.1 Basics of Complexity Theory

Since our work is mainly focused on the implementation and description of a procedure to solve a problem efficiently, we need to start by introducing some basics concerning the idea of algorithm and its computational aspects. *Computational Complexity Theory* is a field of theoretical Computer Science which has the goal to compare and classify the difficulty of solving problems, with the practical aim to obtain bounds and quantitative relations for complexity. In this section, we briefly introduce the basic concepts and notions of this theory.

2.1.1 Complexity of algorithms

An algorithm is a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation and that, given an input, it produces an output. Complexity theory investigates the solvability of individual problems, in terms of the

quantity of computational resources an algorithm takes to solve the problem. The most interesting and considered resources in complexity literature to measure the efficiency of an algorithm are time and space. We can define:

- *Time complexity function* for an algorithm expresses the time requirements by giving, for each possible input size, the largest amount of time the algorithm takes to solve the problem;
- *space complexity function* of an algorithm expresses the amount of storage an algorithm takes to solve the problem in terms of input size.

Although space complexity is usually very important in the implementation of an algorithm, in the present work we focus more on time complexity because studying it allows us to understand whether a problem can be solved in an appropriate time.

To describe algorithm's running time based on the size of the algorithm's input, the Big-O notation is introduced.

Definition 2.1. *Let $f,\ g : \mathbb{N}^+ \to \mathbb{R}^+$ and suppose g(n) is the running time of an algorithm on an input size n. We denote the asymptotic running time of an algorithm by $O(f(n))$. This is called the Big-O notation, meaning there exists some positive constant c such that for all $n \in \mathbb{N}$, $g(n) \leq c \cdot f(n)$; $c \cdot f(n)$ bounds g(n) from above.*

The Big-O notation provides an asymptotic upper bound, since it bounds from above the growth of the running time of the algorithm for input sizes.
Although different algorithms have different time complexity functions, using the given Big-O definition, computer scientists recognized two main simple classes of algorithms.

Definition 2.2. *An algorithm is polynomial-time (or efficient) if it has a worst-case time complexity $f(n) = O(n^k)$, where k is a constant and n is the size of the input.*

Definition 2.3. *An algorithm is exponential-time if it has a worst-case time complexity $f(n) = O(2^n)$, where n is the size of the input.*

As expected, a polynomial-time algorithm is more efficient and faster than an exponential-time algorithm. In Complexity theory literature a problem is considered "well-solved" only when a polynomial-time algorithm to solve it is known. The notions of *tractable* and *intractable* for a problem are so introduced.

Definition 2.4. *A problem is tractable if, and only if, there exists a polynomial-time algorithm that solves all instances of it. On the other hand, a problem is intractable if, and only if, there is no known polynomial-time algorithm for solving all instances of it.*

2.1.2 Complexity of problems

Central to the development of Computational Complexity Theory is the concept of *decision problem.* Intuitively, a decision problem is any problem to which a solution can either be given as an answer *Yes* or *No*. An example is the Traveling Salesman Problem (TSP), which gives a list of cities and distances between each pair of cities and asks "Is this a route that visits all cities under cost c?".
Decision problems can belong to different Complexity classes.

- **P** is class of decision problems that can be solved by polynomial time algorithms;
- **NP** is the class of decision problems such that, for every *Yes*-instance there is a concise certificate which allows to verify, in polynomial time, that the instance really admits a *Yes* answer. We say that these problems can be verified in polynomial time;
- **coNP** is class of decision problems whose complements are in NP. Complement of a decision problem is obtained reversing the *Yes* and *No* answers.

The other important category in Complexity theory is composed of the *functional problems*. Functional problems are problems where a single output, if it exists, is expected for every input value. The output in functional problems is more complex than the one of a decision problem.

- **FP** is the class of functional problems that can be solved by a polynomial time algorithm;
- **FNP** is the class of functional problems such that there exists an algorithm that, given a problem instance I and a solution y, can verify, in polynomial time, whether y is a solution of I.

The relationship between the classes P and NP is fundamental for the Computational Complexity Theory. A famous conjecture states that P is properly contained in NP, *i.e.* $\mathrm{P} \subset \mathrm{NP}$, and it means that every problem belonging to P is also in NP. There is no formal proof of this conjecture and in fact demonstrating if $\mathrm{P} = \mathrm{NP}$ remains an open question in complexity theory. However, it seems more reasonable to operate

under the assumption that P $\neq$ NP. Every decision problem solvable by a polynomial-time deterministic algorithm is indeed solvable by a polynomial-time nondeterministic algorithm.

The distinction between P and NP is suggested by the presence of a subclass of problems in NP, the so called *NP-complete problems*. Informally, a decision problem π is NP-Complete if $\pi \in$ NP and for all other decision problems $\pi' \in$ NP exists a polynomial time transformation to π.

2.1.3 Hard problems

In this section, we introduce the notion of NP-hardness. Informally, the NP-Hard is a class of problems that are at least as hard as the hardest problems in NP. More precisely, as stated by the following definition, a problem π is NP-hard when every problem π' in NP can be reduced in polynomial time.

Definition 2.5. *A problem π is* NP-Hard *if every problem $\pi' \in$ NP is such that $\pi' \leq_P \pi$.* [1]

The notion of NP-hardness is useful to investigate the complexity of problems. The following definition presents the relationship between NP-Complete and NP-Hard classes.

Definition 2.6. *A problem π is in* NP-Complete *if it is in* NP-Hard *and belongs to* NP.

It is not difficult to see that if a problem is in NP-Hard, then it can not be solved in polynomial time unless P = NP. It would mean that any NP or NP-Complete problem can be solved in polynomial time. On the other hand, proving that does not exist a polynomial algorithm for a NP-Complete problem implies P$\neq$NP. Despite the fact that the most widely accepted hypothesis in the literature is that P$\neq$NP, the question is still open and the Figure 2.1 shows the two possible relationships.

[1] notation $\pi' \leq_P \pi$ means that problem π is *at least as complex as* π'.

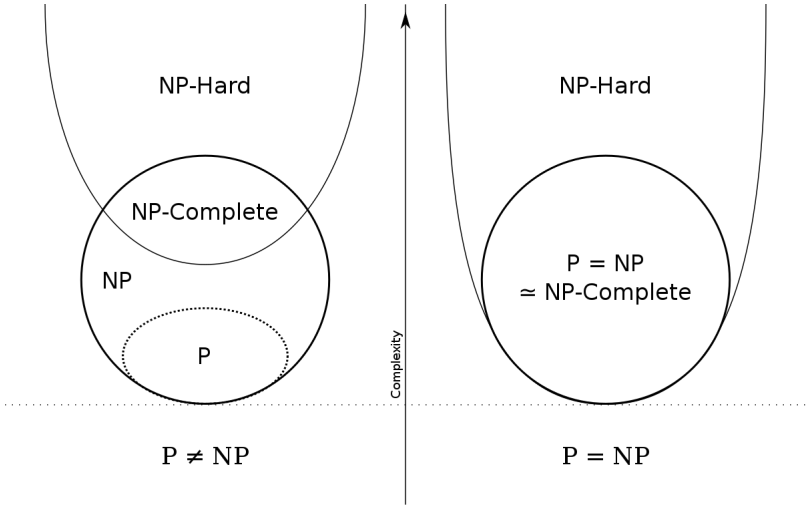


Figure 2.1. *Possible relations between P and NP classes.*

2.2 Introduction to Game Theory

In this section, we introduce the notions and definitions related to the *Game Theory* field. Game Theory is a theoretical framework for studying strategic interactions among competitive players, who have a set of possible choices. Initially developed in economics, thanks to its flexibility and its high level of abstraction of real-life situations, Game Theory has now a wide range of applications, from mathematics and political science to biology and psychology. The subject covered is the analysis of conflict and cooperation in a context in which the result for a player does not only depend on his own decisions, but also involves the behavior of other players.

In this section we introduce the formal definition of *game*, the basic notion in Game Theory to model highly abstract representation from real-life situations. Then, we mainly focus on a specific class of games called *Congestion Games*.

2.2.1 General notion of game

A game, as defined by Osborne and Rubinstein, is "a description of strategic interaction that includes the constraints on the actions that the players *can* take and the players' interests, but does not specify the actions that the players *do* take (Osborne, Rubinstein, 1994, p. 2). Players therefore interact with each other and, depending on the chosen strategy, each player obtains a final outcome which may depend on the

actions of other players. As mentioned, each player has interests and during the game he tends to act according to them.

Game Theory provides several representations of a game to describe all the formal aspects involved. We present the *normal-form representation*, which is able to capture all the features of *simultaneous game*, a situation in which players make their moves simultaneously, making a player unaware of other players' moves.

Definition 2.7. *The normal-form representation of a game is defined by a tuple (N, A, U) where :*

- $N = \{1, 2, ..., n\}$ *is the set of players;*
- $A = \{A_1, A_2, ..., A_n\}$ *is the set of action spaces, where* $A_i = \{a_1, a_2, ..., a_{m_i}\}$ *is the set of player i's actions;*
- $U = \{U_1, U_2, ..., U_n\}$ *is the set of utility functions, where* $U_i : A_1 \times A_2 \times ... \times A_n \to \mathbb{R}$ *is the utility function of player i.*

Players, in the model we consider, are supposed to be *rational*, in the sense that they aim at maximizing their utility function, and *selfish*, because each player does not care about the effects of his action on other players.

Now we introduce the fundamental notions of *action profile*, as a collection of actions, and *strategy* in order to describe the behaviour of the players in a game.

Definition 2.8. *Action profile* $\mathbf{a}$ *is a tuple* $(a_1, a_2, ..., a_n)$ *with* $a_i \in A_i$*, containing one action per player. Action profile* $\mathbf{a}_{-i}$ *is a tuple* $(a_1, a_2, ..., a_{i-1}, a_{i+1}, ..., a_n)$ *with* $a_j \in A_j$*, containing one action per player, except for player i. We denote by* $A_{-i} = A_1 \times A_2 \times ... \times A_{i-1} \times A_{i+1} \times ... \times A_n$ *the space of* $\mathbf{a}_{-i}$*.*

Definition 2.9. *A strategy* $\sigma_i : A_i \to [0, 1]$ *with* $\sigma_i \in \Delta(A_i)$ *is a function returning the probability with which action* $a_i \in A_i$ *is played by player i. We denote with* $\Delta(\cdot)$ *the simplex over* $\cdot$*.*

Similarly as we defined the notion of action profile, a tuple $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, ..., \sigma_n)$ containing one strategy per player denotes a *strategy profile*. A strategy profile completely captures the behavior of all the the players in the game. It is also important to notice that, given an action profile, the outcome of a game and the utilities of each player are universally identified.

2.2.2 Congestion Games

Congestion Games is a class of noncooperative games, introduced for the first time by Robert Rosenthal [16]. In these games, each player is allowed to choose a subset of global set of available resources, for example drivers have to choose among different paths to reach a destination. As the time taken by a driver to reach a destination depends on the traffic on the road, in a Congestion Game a player cost depends on the number of the total players using the same resources. Congestion Games have been an important research area, since they can model different situations such as scheduling, routing and network design.
The definition of Congestion Game and its basic related notions are introduced.

Definition 2.10. *A Congestion Game is a tuple* $(N, M, (A_i)_{i\in N}, (c_j)_{j\in M})$ *where:*

- $N = \{1, 2, ..., n\}$ *is the set of players;*
- $M = \{1, 2, ..., m\}$ *is the set of resources;*
- $A_j \subseteq \wp(M)$ *is the set of action of player i, where each action a is a subset of the set of resources;*
- $c_j : N \to \mathrm{R}$ *is a function returning the cost related to resource j when it is used by a given number of players.*

We can introduce more notions in order to better define the framework of Congestion Games: the *congestion* of a resource, the *cost* of a player and the *social cost*.

Definition 2.11. *Given an action profile* $\mathbf{a} \in \boldsymbol{A}$*, the congestion of resource j, denoted by* $\mathrm{cong_j(a)}$*, is the number of players using resource j in* $\mathbf{a}$*. Formally,* $\mathrm{cong_j(a)} = \sum_{i\in N} \mathrm{use}(a_i, j)$ *where:*

$$\mathrm{use}(a_i, j) = \begin{cases} 1 & \text{if } j \in a_i, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2.12. *The cost of a player i, given action profile* $\mathbf{a}$*, is returned by function* $C_i : \boldsymbol{A} \to \mathbb{R}$ *where:*

$$C_i(a_i, \mathbf{a}_{-i}) = \sum_{j\in a_i} c_j(\mathrm{cong_j}(a_j, \mathbf{a}_{-i})).$$

Definition 2.13. *The social cost is given by:*

$$\sum_{i\in N} C_i(\mathrm{a}).$$

The following example helps to understand all the notions we have introduced.

Example 2.1. *Considers the following congestion game:*

- $N = \{1, 2, 3\}$;
- $M = \{1, 2, 3, 4, 5\}$;
- $A_1 = A_2 = A_3 = \{\{1, 4\}, \{1, 3, 5\}, \{2, 5\}\}$;
- $c_1(x) = \begin{cases} 2 & x = 1 \\ 3 & x = 2 \\ 5 & x = 3, \end{cases}$ $c_2(x) = \begin{cases} 4 & x = 1 \\ 6 & x = 2 \\ 7 & x = 3, \end{cases}$ $c_3(x) = \begin{cases} 1 & x = 1 \\ 2 & x = 2 \\ 8 & x = 3, \end{cases}$
  $c_4(x) = \begin{cases} 2 & x = 1 \\ 3 & x = 2 \\ 6 & x = 3, \end{cases}$ $c_5(x) = \begin{cases} 1 & x = 1 \\ 5 & x = 2. \\ 6 & x = 3 \end{cases}$

The example represents the situation in which there are 3 players moving from the same source (node A*) to the same target (node* B*). The graphical interpretation is given in Figure* 2.2*. The actions available to the players are all the possible paths connecting source* A *to target* B*. Player* 1 *and* 2 *choose actions* $\{1, 3, 5\}$ *while player* 3 *chooses action* $\{1, 4\}$*. Accordingly to the definition reported above, we have:*

- $\text{cong}_1(\mathbf{a}) = 3$ *and* $c_j(\mathbf{a}) = 5$,
- $\text{cong}_2(\mathbf{a}) = 0$ *and* $c_j(\mathbf{a}) = 0$,
- $\text{cong}_3(\mathbf{a}) = 2$ *and* $c_j(\mathbf{a}) = 2$,
- $\text{cong}_4(\mathbf{a}) = 1$ *and* $c_j(\mathbf{a}) = 2$,
- $\text{cong}_5(\mathbf{a}) = 2$ *and* $c_j(\mathbf{a}) = 5$.

Furthermore, we have

- $C_1(\mathbf{a}) = 12$,
- $C_2(\mathbf{a}) = 12$,
- $C_3(\mathbf{a}) = 7$.

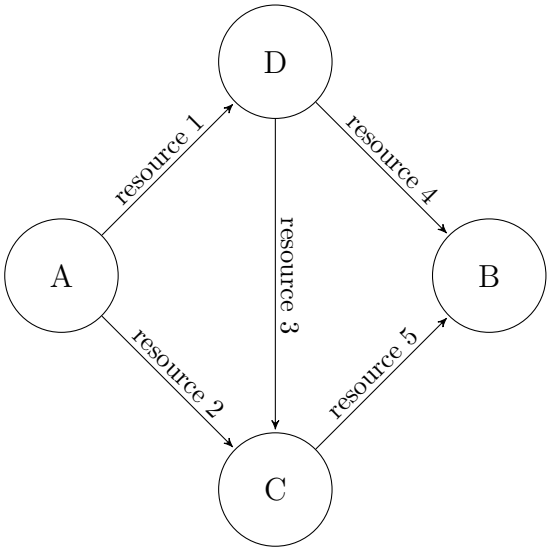


Figure 2.2. *Example of Congestion Game.*

2.3 Information structure design

The analysis of information design problems has been a crucial point of the recent literature. *Information structure design* studies how to influence the behavior of self-interested agents by providing incentives or by manipulating their beliefs, in order to achieve desired outcomes. In the present paper we focus on the manipulation of beliefs, which concerns in the selective arrangement of the information provided to agents. The interaction among agents is modelled as a game of incomplete information, in which the players have only partial information about the status of the game and about the other players. The information structure of a game determines "who knows what" and describes how the knowledge is distributed among players.
The *persuasion*, defined as act of exploiting an informational advantage in order to optimize a given objective, is the main topic studied in Information structure design.

The technical and conceptual starting point of our research regarding persuasion is the Bayesian persuasion framework. To introduce the basic concepts and assumptions, we therefore present the most influential model in Information structure design, the Bayesian persuasion model introduced by Kamenica and Gentzkow [6], regarding the problem of single agent persuasion.

2.3.1 Bayesian persuasion: single agent case

The model proposed by Kamenica and Gentzkow [6] is probably one of the simplest and most applied in Information structure design. Here we have two players: a *sender* and a *receiver*. In Bayesian persuasion, we adopt the perspective of the sender looking to persuade the receiver to take an action that is desirable for the sender himself. The receiver can choose one action a from a set of actions A and, together with the *state of nature* θ, it determines the utilities of both the sender and the receiver. The state of nature is a parameter, or a set of parameters, which determines the payoff function of the game. In particular, the utility of the receiver is specified by the function $u : A \times \Theta \to \mathbb{R}$. In turn, the utility of the sender is $f : A \times \Theta \to \mathbb{R}$, which highlights how the receiver's choice of the action influence the payoff of the sender. The states of nature θ is drawn from a finite set Θ of potential realizations of nature, according to a common prior distribution μ. As in persuasion more generally, the sender leverage is in his informational advantage, in his access to the realization of the state of nature, which contributes to determine the payoff for both sender and receiver.

In the Bayesian persuasion it is assumed that the sender can commit to a policy, known as *signaling scheme*, that strategically reveals the information to the receiver before the realization of the state of nature. Formally, a signaling scheme $\phi : \Theta \to \Delta(\Sigma)$, is a mapping from states of nature Θ to a family of distribution over *signals* Σ. For $\theta \in \Theta$, we denote with $\phi(\theta, \sigma)$ the probability that the sender selects signal $\sigma \in \Sigma$ after observing the states of nature θ. Given a signaling scheme ϕ with Σ set of signals, each signal $\sigma \in \Sigma$ is realized with probability $\Pr(\sigma) = \sum_\theta \mu(\theta)\phi(\theta, \sigma)$. Upon receiving signal σ, receiver performs a Bayesian update and infers posterior belief over the state of nature. The realized state is θ with posterior probability $p_\theta = \mu_\theta \phi(\theta, \sigma)/\Pr(\sigma)$. The general interaction between sender and receiver is described by Table **??**.

Fundamental purpose in Bayesian persuasion is to find the sender's optimal signaling scheme, *i.e.* the signaling scheme maximizing his expected utility. A simple revelation-principle style argument built by Kamenica and Gentzkow [6] shows that there exists an optimal signaling scheme which is *direct* and *persuasive*. A *direct* signaling scheme means that each signal $\sigma_i \in \Sigma$ can be interpreted as a recommendation to play action $a_i \in A$. The straightforward consequence of the argument is that $|\Sigma| = |A|$. Instead, a signaling scheme is *persuasive* if the receiver has no incentive in deviating from the recommended action.

Sender-Receiver interaction

1. The sender commits to a signaling scheme ϕ;
2. the sender observes the realized state of nature $\theta \sim \mu$;
3. the sender draws a signal $\sigma \sim \Sigma$ and communicates it to the receiver;
4. the receiver observes the signal σ and rationally updates his beliefs over Θ according to the *Bayes* rule;
5. the receiver selects an action maximizing his expected utility.

Table 2.1. *Sender-Receiver interaction.*

Considering direct and persuasive signaling scheme, it is possible to write the following linear program (LP) to find the optimal signaling scheme in Bayesian persuasion with a single receiver.

$$\begin{aligned}
\max_{\phi(\theta,a)} \quad & \sum_{\theta\in\Theta} \mu_\theta \sum_{a\in A} \phi(\theta,a) f(\theta,a) & \\
\text{s.t.} \quad & \sum_{\theta\in\Theta} \mu_\theta \phi(\theta,a)(u(\theta,a) - u(\theta,a')) \geq 0 & \forall a, a' \in A \\
& \sum_{a\in A} \phi(\theta,a) = 1 & \forall \theta \in \Theta \\
& \phi(\theta,a) \geq 0 & \forall \theta \in \Theta, a \in A
\end{aligned}$$

The only variable in this LP is $\phi(\theta, a)$, for each state of nature $\theta \in \Theta$ and action $a \in A$, that represents the conditional probability of recommending action a in state of nature θ. Actually, it is not practical to solve this LP, unless the prior distribution μ is of small support and given explicitly, but it serves as a useful structural characterization.
The following example illustrates the Bayesian model described above.

Example 2.2. (Dughmi [4])
Consider an academic adviser (the sender) who is writing a recommendation letter (the signal) for her graduating student to send to a company (the receiver), which in turn must decide whether or not to hire the student. The adviser gets utility 1 *if her student is hired, and* 0 *otherwise. The state of nature determines the quality of the student, and hence the company's utility for hiring the student. Suppose that*

the student is excellent with probability $\frac{1}{3}$*, and weak with probability* $\frac{2}{3}$*. Moreover, the company gets utility* 1 *for hiring an excellent student, utility* -1 *for hiring a weak student and utility* 0 *for not hiring. Consider the following signaling schemes:*

- *No Information: Given no additional information, the company maximizes his utility by not hiring. The adviser's expected utility is* 0*.*
- *Full information: Knowing the quality of the student, the company hires if and only if the student is excellent. The adviser's expected utility is* $\frac{1}{3}$*.*
- *The optimal (partially informative) scheme: The adviser recommends hiring when the student is excellent, and with probability just under* 0.5 *when the student is weak. Otherwise, the adviser recommends not hiring. The company maximizes his expected utility by following the recommendation, and the adviser's expected utility is just under* $\frac{2}{3}$*.*

2.3.2 Bayesian persuasion: multiple agents

The model proposed by Kamenica and Gentzkow [6] presents the main principles of the Bayesian persuasion but it has some limitations. Using the model as a building block for more complex descriptions, the most interesting development in literature considers a setting which involves multiple receivers. Clearly, it leads to a more complicated model description, as the interaction between the receivers must be taken into account. Arieli and Babichenko [12] introduced the so called *no-externality* assumption to simplify the model. This assumption implies that one receiver's action does not impose an externality, making each receiver's utility independent from the actions selected by the others. It means that each receiver's utility can be written as a function of state of nature and his own particular action, as in the single-agent persuasion. However, it is not always possible to rely on the *no-externality* assumption and in more realistic contexts the receiver's payoff can be instead closely dependent on the actions of other receivers. The sender's utility, on the other hand, always depends on the state of nature and the profile of receiver actions. The case with no externalities is often applied to voting problems [27].

In multi-receivers setting is possible to have different policies through which information is distributed, to create asymmetry among the re-

ceivers. The literature has mainly dealt with the following two basic signaling schemes.

Private signaling scheme: the sender can reveal different information to different receivers through a private communication channel.

Public signaling scheme: all receivers in the game obtain from the sender the same information through a public communication channel.

The key difference between the two types of signaling schemes lies in their different impacts on receivers' beliefs. Signals from public persuasion lead to common belief about the unknown state of the world for all receivers; but signals from private persuasion do not.
We now analyze in detail the two different revelation policies.

2.3.3 Private signaling scheme

A private signaling scheme is a general policy that allows the sender to reveal privately different information to the receivers. This policy creates asymmetry of information and coordinate the behaviour of the receivers, providing more flexibility to the sender.
More formally, a private signaling scheme $\phi : \Theta \to \Delta(\Sigma_1 \times \Sigma_2 ... \times \Sigma_n)$ is a map from the set of states of nature Θ to a set of *signal profiles* $\Sigma = \Sigma_1 \times \Sigma_2 ... \times \Sigma_n$, where Σ_i identifies the signal set of receiver i. Therefore, the output of a private signaling scheme is a signal profile $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_n) \in \Sigma$ where the particular signal σ_i is sent to receiver i via a private channel. Given a private signaling scheme, $\phi(\theta, \boldsymbol{\sigma})$ denotes the probability of selecting the signal profile $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_n)$, given the state of nature θ. For each state of nature $\theta \in \Theta$ and receiver $i \in [n]$, each signal $\sigma_i \in \Sigma_i$ is realized with marginal probability $\mathrm{P}(\sigma_\mathrm{i}) = \sum_{\theta \in \Theta} \mu_\theta \ \phi_i(\theta, \sigma_i)$. Upon receiving signal σ_i, each receiver i performs a Bayesian update and infers a posterior belief $p^i_\theta = \mu_\theta \phi_i(\theta, \sigma_i)/\mathrm{P}(\sigma_\mathrm{i})$ over the state of nature θ.
Applying the same argument proposed by Kamenica and Gentzkow [6] for the single-agent persuasion, it is possible to restrict our attention to an optimal private signaling scheme which is both direct and persuasive. As a consequence, each signal profile $\boldsymbol{\sigma} \in \Sigma$ can be expressed as an action profile $\mathbf{a} \in \boldsymbol{A}$. So, for each receiver i, we denote with A^i his set of actions.

Finally, the sender optimization problem can be computed by the following exponentially-large linear program. The notation $\phi(\theta, \mathbf{a})$ represents the probability of the sender selecting action profile $\mathbf{a}$ in state of nature θ,

while $\phi_i(\theta, a)$ denotes the marginal probability of recommending action $a \in A^i$ to receiver i.

$$
\begin{aligned}
\max \quad & \sum_{\theta \in \Theta} \mu_\theta \sum_{\mathbf{a} \in \boldsymbol{A}} \phi(\theta, \mathbf{a}) f(\theta, \mathbf{a}) \\
\text{s.t.} \quad & \sum_{\mathbf{a} \in \boldsymbol{A}: \mathbf{a}_i = a} \phi(\theta, \mathbf{a}) = \phi_i(\theta, a) && \forall i \in N, \forall a \in \boldsymbol{A}^i, \forall \theta \in \Theta \\
& \sum_{\theta \in \Theta} \mu_\theta \ \phi_i(\theta, a)(u_i(\theta, a) - u_i(\theta, a')) \geq 0 && \forall i \in N, \forall a, a' \in \boldsymbol{A}^i \\
& \sum_{\mathbf{a} \in \boldsymbol{A}} \phi(\theta, \mathbf{a}) = 1 && \forall \theta \in \Theta \\
& \phi(\theta, \mathbf{a}) \geq 0 && \forall \theta \in \Theta, \forall \mathbf{a} \in \boldsymbol{A}
\end{aligned}
$$

2.3.4 Public signaling scheme

Constraining the sender to communicate through a public communication channel simplifies considerably the model but introduces some obvious limitations to the sender's ability to discriminate between receivers and correlate their actions. A public signaling scheme π can be considered as a special type of private signaling scheme in which each receiver must receive the same signal. Hence, only one public signal is sent.
Formally, a public signaling scheme $\pi : \Theta \to \Delta(\Sigma_1 \times \Sigma_2 ... \times \Sigma_n)$ is a map from states of nature Θ to distributions over *public signals* Σ. Given a public signaling scheme, $\pi(\theta, \sigma)$ denotes the probability of sending signal $\sigma \in \Sigma$ at state of nature θ. Moreover, $\Pr(\sigma) = \sum_{\theta \in \Theta} \pi(\theta, \sigma)$ indicates the probability with which a receiver receives the signal σ. Each receiver then performs the same Bayesian update and infers the same posterior belief $p_\theta = \mu_\theta \pi(\theta, \sigma)/\Pr(\sigma)$ over the state of nature.
A simple revelation-principle style argument allows us to consider an optimal public signaling scheme which is direct and persuasive. As a consequence, each signal profile $\boldsymbol{\sigma} \in \Sigma$ can be expressed as an action profile $\mathbf{a} \in \boldsymbol{A}$, recommending an action to each receiver.

Finally, the sender's optimization problem can be expressed as the

following exponentially-large LP.

$$
\begin{aligned}
\max \quad & \sum_{\theta\in\Theta} \mu_\theta \sum_{\mathbf{a}\in\boldsymbol{A}} \pi(\theta,\mathbf{a}) f(\theta,\mathbf{a}) \\
\text{s.t.} \quad & \sum_{\theta\in\Theta} \mu_\theta \; \pi(\theta,\mathbf{a})(u_i(\theta,\mathbf{a}_i) - u_i(\theta,a')) \geq 0 \qquad \forall i \in N, \forall \mathbf{a} \in \boldsymbol{A}, a' \in \boldsymbol{A}^i \\
& \sum_{\mathbf{a}\in\boldsymbol{A}} \pi(\theta,\mathbf{a}) = 1 \qquad \forall \theta \in \Theta \\
& \pi(\theta,\mathbf{a}) \geq 0 \qquad \forall \theta \in \Theta, \forall \mathbf{a} \in \boldsymbol{A}
\end{aligned}
$$

The following example illustrates the results from both the revelation policies.

Example 2.3. (Dughmi [4])
Consider an academic adviser (the sender) who is writing a recommendation letter (the signal) for his student. The student has applied for two fellowship programs (the receivers), each of which must decide whether or not to award the student a fellowship funding as part of his graduate education. Suppose that the student can accept one or both fellowship awards. The adviser gets utility 1 *if his student is awarded at least one fellowship, and* 0 *otherwise. A student is excellent with probability* $\frac{1}{3}$ *and weak with probability* $\frac{2}{3}$*, and a fellowship program gets utility* 1 *from awarding an excellent student,* -1 *for awarding a weak student, and* 0 *from not awarding the student. Rationally, a fellowship program makes an award if and only if it believes its expected utility for doing so is non negative.*
Consider the following signaling schemes:

- *No Information: Neither program makes the award, and the adviser's utility is* 0*.*
- *Full information: Both programs make the award if the student is excellent, and neither makes the award if the student is weak. The adviser's expected utility is* $\frac{1}{3}$*.*
- *Optimal public scheme: If the student is excellent, the adviser publicy signals* "award"*. If the student is weak, the adviser publicy signals* "award" *or* "don′t award" *with the same probability. Therefore, both programs are simultaneously persuaded to award the student the fellowship with probability* $\frac{2}{3}$*, and neither makes the award with probability* $\frac{1}{3}$*. The adviser's expected utility is* $\frac{2}{3}$*.*
- *Optimal private scheme: If the student is excellent, the adviser recommends* "award" *to both fellowship programs. If the student is*

> *weak, the adviser recommends* "award" *to one fellowship program chosen uniformly at random, and recommends* "don′t award" *to the other. The result is that both fellowship programs make the award when the student is excellent, and exactly one of the programs makes the award when the student is weak. The utility for the adviser is then* 1.

We can assert that public persuasion is usually more difficult computationally to deal with [29].

2.4 Algorithms for Linear Programming

After introducing the theoretical notions that help us to define the problem we are dealing with, this section presents the mathematical concepts and algorithms that constitute the toolbox for the resolution of the problem. The literature regarding resolution methods for linear problems is extremely vast. Specifically, we focus on the definitions of so-called *Ellipsoid Method* and its role in developing a polynomial time algorithm. Then, an overview of the *Row-Generation* technique to iteratively solve a mathematical program is proposed.

2.4.1 Ellipsoid Method

The Ellipsoid algorithm for linear programming is a specific application of the Ellipsoid method, developed for the first time by mathematicians D. B. Yudin and A. S. Nemirovskii [14] and independently by N. Z. Shor. Afterwards, Khachiyan [13] adapted the Ellipsoid method to derive a polynomial-time algorithm for linear programming. In this section the Ellipsoid algorithm and its importance in developing polynomial time algorithms for optimization problems is presented.

We focus our attention on a linear optimization problem with a bounded feasible region. *i.e.* a polytope P. The problem setting considered by the Ellipsoid Algorithm is the following:

$$\text{Given a polytope } P \subseteq \mathbb{R}^n, \text{ find a point } x \in P, \text{ or decide that } P = 0 \tag{2.4}$$

We can assume that P is a full-dimensional polytope. Then, a linear programming over some full-dimensional polytope Q

$$\max w^\top x \tag{2.5a}$$

$$x \in Q \tag{2.5b}$$

can be reduced to finding a point $x \in P = Q \cap \{w^\top x \geq b\}$.
The main advantage of the Ellipsoid Method, compared to other methods for linear programming, is that we just need to be able to solve a definitely simpler sub problem, called the *separation problem.*
A Separation Problem is defined as following:

Definition 2.14. *Given a polytope $P \subseteq \mathbb{R}^n$ and a point $y \in \mathbb{R}^n$:*

- *Decide whether $y \in P$, or if it is not true,*
- *find a non-zero vector $c \in \mathbb{R}^n$ such that $P \subseteq \{x \in \mathbb{R}^n \mid c^\top x \leq c^\top y\}$.*

A procedure able to solve the separation problem in polynomial time is known as *separation oracle.* The formal definition of separation oracle is given below.

Definition 2.15. *A polynomial-time separation oracle for a convex set P is a procedure which given y, either tells that $y \in P$ or returns an hyperplane that separates y and all of P. The procedure runs in polynomial time.*

Note that an hyperplane is defined as $H = \{x \in \mathbb{R}^n \mid c^\top x = \alpha\}$ with $c^\top y \geq \alpha$ and $P \subseteq \{x \in \mathbb{R}^n \mid c^\top x \leq \alpha\}$.

The fact that the Ellipsoid Algorithm is based on a separation oracle allows us to solve in polynomial time many linear programs. As we see in the next chapters, to solve the Linear Problem considered in the present work, characterised by an exponential number of constraints, it was necessary to exploit the Ellipsoid Method by designing a polynomial-time separation oracle.

2.4.2 Cutting-Planes

A linear program in which the variables are restricted to take integer values is called *integer linear program* (ILP). Integer linear problems are more difficult to solve than linear problems and require specific algorithms.
An important algorithms category for integer programming problems is the *exact* algorithms. Exact algorithms guarantee to find an optimal solution for the problem, with the disadvantage that they may need an exponential number of steps. The *Cutting-Planes* algorithm belongs to the exact algorithms class.

The idea behind the Cutting-Planes algorithm is to iteratively cut off parts of the feasible region adding valid inequalities to the LP relaxation. The optimal solution will eventually become and extreme point an found by a polynomial-time algorithm.
Given an integer programming problem ILP = $\max\{c^T x \mid Ax \leq b,\ x \in \mathbb{Z}^n\}$, and its continuous relaxation LP = $\max\{c^T x \mid Ax \leq b,\ x \in \mathbb{R}^n\}$ a generic cutting plane algorithm is defines as follows:

1. solve the linear programming relaxation LP. Let x^* be an optimal solution;
2. if x^* is integer, stop. x^* is an optimal solution to ILP; otherwise,
3. add a linear inequality constraint to LP that all integer solutions in ILP satisfy, but x^* does not; go to Step 1;

2.4.3 Row Generation

The LP relaxation becomes intractable as the number of constraints grows. Then, the Cutting-Planes algorithm can be generalized to solve LP problems with a large number of constraints (*e.g.* exponential). This generalization is named *dynamic constraint generation* or *Row Generation*.
Intuitively, Row Generation is a classical technique to solve a mathematical program by iteratively adding a limited number of constraints to the model. The idea underelying row generation is that typically, only a subset of the original set of constraints is needed to prove optimality. Row generation algorithm is described by the following steps:

1. initialize *reduced master problem* RMP $\tilde{A}x \leq \tilde{b}$ with a limited subset of constraints $Ax \leq b$;
2. solve the LP associated to the *reduced master problem* RMP = $\max\{c^T x \mid \tilde{A}x \leq \tilde{b},\ x \in \mathbb{R}^n\}$ and obtain x*;
3. if x* satisfies all constraints in $Ax \leq b$, stop; otherwise add a violated constrain, or a set of violated constraints, to $Ax \leq b$ and go to Step 2;

Figure 2.3 outlines the typical flow of a Row Generation algorithm. Iteratively, only those constraints that are violated by the current solution are added to the model. The Step 3 has the task of testing the feasibility of the optimal solution x* of the reduced master problem. Subsequently, we have to identify the violated constraints or prove that

there are no more violated constraints. This step is the most important and it is what we called *separation problem*.
In circumstances when, for example, the number of constraints is exponentially large, it is not possible to enumerate all the constraints explicitly and check them one by one. So we need to formulate the separation problem as an LP, or more likely as and ILP. The idea of solving an ILP in an algorithm to solve a LP is common in literature and efficient in practice. We describe in detail the procedure and its relationship to our implementation in the next chapters.

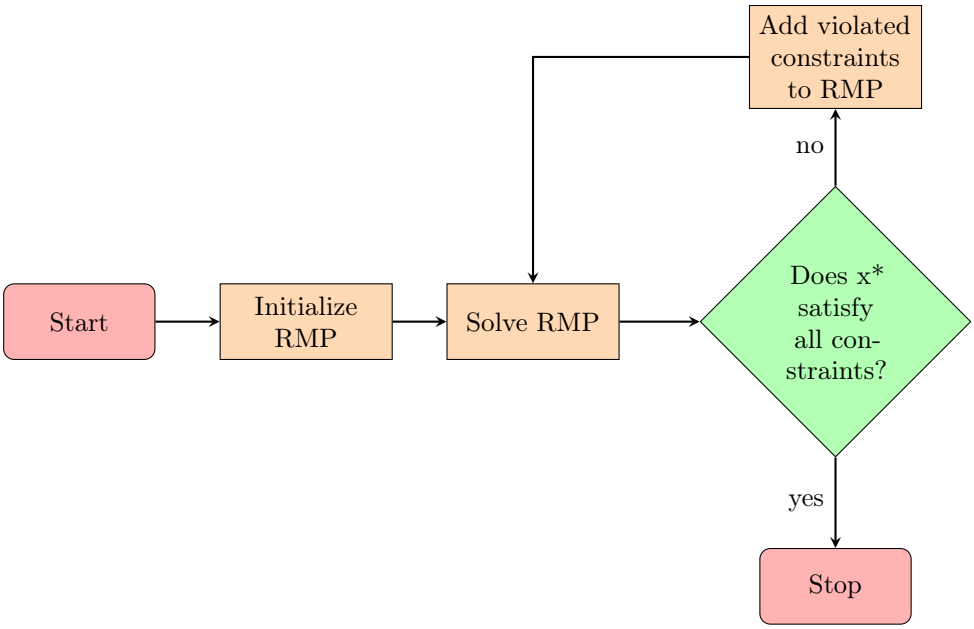


Figure 2.3. *The flow of Row Generation algorithm.*

Chapter 3

Research Problem

In the previous chapter, we presented and analyzed the properties of *Bayesian Framework* and *Congestion Games*, along with *Linear Programming* mathematical tools. Here, our research problem is introduced.

Our aim is to study the signaling scheme in the Bayesian persuasion framework applied to a specific class of Congestion Games, the *Bayesian Network Congestion Games* (BNCG). In particular, we are interested in studying the computation of optimal *ex ante* persuasive signaling scheme in symmetric BNCG with affine cost functions, in polynomial time.

First, an introduction of the formal definitions and notations of Bayesian Network Congestion Game is proposed. Then, we present the networks setting assumptions in which we work. Finally, the problem of signaling scheme in BNCGs is exposed along with an example.

3.1 Bayesian Network Congestion Games

We start by introducing a special class of Congestion Games. Network Congestion Games (NCGs) are a model to study multi-agent strategic interactions, in which the resources correspond to simple paths in a graph, *e.g.* routing options from a source to a destination.
Intuitively, each player chooses a set of transitions, forming a simple path from a source state to a target state and the cost of a transition increases with the number of players using it. The players selfishly seek to minimize their costs in crossing the network.
Network Congestion Games are very popular, due to their important practical applications, as in the transportation or communication networks.

In real-world problems, however, the state of the network is typically uncertain and unknown to the players (*e.g.* drivers may not be aware of road works and accidents in a road network). To represent uncertainty in the network, the *Bayesian Network Congestion Games* (BNCGs) model is introduced.

The BNCGs model is based on the Bayesian persuasion framework proposed by Kamenica and Gentzkow [6]. Here, a sender, for example a third-party entity, is informed about the realized state of the network and he is able to commit to a signaling scheme, which is a map from network states to action recommendations (*i.e.* routes suggestions) for the players. The distinctive feature in a BNCG is that the route costs are affected by a random network state.

The formal definition of Bayesian Network Congestion Game is given below.

Definition 3.1. *A Bayesian Network Congestion Game is defined as a tuple* $(N, G, \Theta, \mu, \{c_{e,\theta}\}_{e\in E,\theta\in\theta}, \{(s_p, t_p)\}_{p\in N})$*, where:*

- $N := \{1, ..., n\}$ *denotes the set of players;*
- $G := (V, E)$ *is the directed graph underlying the game. In particular,* V *denotes the set of nodes and* $e = (v, v') \in E$ *represents a directed edge from* v *to* v'*;*
- Θ *is the finite set of states of nature;*
- μ *encodes the prior belief that the players have over the states of nature. In particular* $\mu \in int(\Delta_\Theta)$ *is a fully-supported probability distribution over the set* Θ*, with* μ_θ *denoting the prior probability that state of nature is* $\theta \in \Theta$*;*
- $\{c_{e,\theta}\}_{e\in E,\theta\in\Theta}$ *are the edge costs, which each* $c_{e,\theta} : \mathbb{N} \times \Theta \to \mathbb{R}_+$ *defining the cost of edge* $e \in E$ *as a function of the number of players passing through* e *and the state of nature* θ *drawn from the set* Θ*;*
- $\{(s_p, t_p)\}_{p\in N}$ *with* $s_p, t_p \in V$*, denote the source-destination pairs for all the players.*

Further mathematical notation is introduced to describe more accurately Bayesian Network Congestion Games. Formally, A_p defines the set of actions available to player $p \in N$, *i.e.* the set of all directed paths from

s_p to t_p in graph G. Accordingly, $a_p \in A_p$ indicates a player p's path and we write $e \in a_p$ if the path contains the specific edge e.

In BNCGs, an action profile $a \in A$, where $A := \times_{p \in N} A_p$, is a tuple of s_p-t_p directed paths $a_p \in A_p$, one per player $p \in N$. We define the edge congestion ${f_e}^a$ as the number of players selecting a path passing through the same edge e in action profile a. Formally, ${f_e}^a := |\{p \in N \mid e \in a_p\}$. Hence, the expression $c_e({f_e}^a)$ denotes the cost of edge e in action profile a. Finally, the costs experienced by a player $p \in N$ in an action profile $a \in A$ in a state of nature $\theta \in \Theta$ is computed as $c_{p,\theta} := \sum_{e \in a_p} c_{e,\theta}({f_e}^a)$. We can easily see that the cost experienced by one player also depends on the number of other players that use the same edge in the same action profile, so the *no externality* assumption, as defined in Section 2.3.2 is not longer valid in BNCGs. The action taken by a player affects the final outcome of the other players.

3.2 Assumptions on BNCGs

Since the *no externality* assumption is no longer valid in BNCGs, in this section we present crucial properties to deal with the problem of computing optimal signaling schemes. We focus our attention to *symmetric* BNCGs with *affine* cost functions.

Symmetric BNCGs: it implies that the source s_p and the destination t_p in the graph G are the same for all the players. As a consequence, all the players have the same set of actions (*i.e.* paths).

Affine costs: it means that for all $e \in E$ and $\theta \in \Theta$, there exist constants $\alpha_{e,\theta}$, $\beta_{e,\theta} \in \mathbb{R}_+$ such that the edge cost function can be expressed as $c_{e,\theta}({f_e}^a) := \alpha_{e,\theta} {f_e}^a + \beta_{e,\theta}$. It has been proved by Vasserman, Feldman, and Hassidim [22] that this assumption is reasonable in many applications and the problem is trivially NP-Hard when generic costs are allowed.

The symmetric assumption is crucial since it has been proved by Castiglioni, Celli, Marchesi and Gatti [4] that computing an optimal *ex ante* persuasive signaling scheme in *asymmetric* BNCGs is NP-Hard. We focus also on the notion of *ex ante persuasiveness*, as defined by Celli, Gatti and Coniglio [5]. This setting requires that receivers are encouraged to follow the sender's recommendations by only observing the signaling scheme.
The formal definition of *ex ante* persuasive signaling scheme is given in the following page.

Definition 3.2. *A signaling scheme $\phi : \Theta \to \Delta_A$ is ex ante persuasive if, for each $p \in N$ and $a_p \in A_p$, it holds:*

$$\sum_{\theta \in \Theta} \mu_\theta \sum_{a' = (a'_p, a_{-p}) \in A} \phi_{\theta,a'} \big(c_{p,\theta}(a_p, a_{-p}) - c_{p,\theta}(a') \big) \geq 0$$

The notion of *ex ante* persuasiveness has been introduced to computationally approach problems. In fact, the classical notion of *ex interim* persuasiveness, which allows the receivers to deviate after observing the sender's signal, leads to computationally intractable problems most of the time. The *ex-ante* assumption, as argued by Kamenica and Gentzkow [6], is not unrealistic and in our setting we suppose that each receiver decides to either follow the signal or act based on his prior belief about the network state.

3.3 Signaling in BNCGs

Deriving from the Bayesian Persuasion framework proposed by Kamenica and Gentzkow [6], in BNCGs we have a informed sender, which has access to the realized state of nature, trying to influence the behaviour of the multiple receivers. The sender can commit to a signaling scheme $\phi : \Theta \to \Delta_A$ that maps the realized state of nature to a signal for each player. By exploiting *private* communication channels, it is possible for the sender to send different signals to each player, *i.e.* suggest different path routes to the receivers. We are particularly interested in the possibilities of an informer to reduce social cost by distributing information to the receivers, who update their beliefs rationally. The optimal signaling scheme has this role and its computation in polynomial time is the main problem we deal in the present work.
In BNCGs, the probability of recommending an action profile $a \in A$ given the state of nature $\theta \in \Theta$ is denoted by $\phi_{\theta,a}$. Being a probability distribution over action profiles, it holds that $\sum_{a \in A} \phi_{\theta,a} = 1$ for each $\theta \in \Theta$. After observing the state of nature $\theta \in \Theta$, the sender draws an action profile $a \in A$ according to $\phi_{\theta,a}$ and recommends a_p to each player $p \in N$.
Formally, a sender's *optimal ex ante* persuasive signaling scheme ϕ^* is a signaling scheme minimizing the *expected social cost* of the solution, *i.e.*:

$$\phi^* \in \arg\min_{\phi} \sum_{\theta \in \Theta} \mu_\theta \sum_{a \in A} \phi_{\theta,a} \sum_{p \in N} c_{p,\theta}(a).$$

The following example illustrates the interaction flow between the sender and the receivers in a simple Bayesian Network Congestion Game.

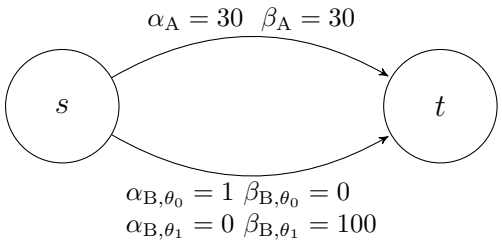
Example 3.1. (Castiglioni, Celli, Marchesi, Gatti [4]).
Figure 3.1 (Up) describes a simple BNCG modeling road network between the JFK International Airport (node s), and Manhattan (node t). It is late at night and three lone researchers have to reach the AAAI venue. They are following navigation instructions from the same application, whose provider (the sender) has access to the current state of the roads (node A *and node* B*, respectively) Roads costs (i.e. travel times) are depicted in Figure 3.1 (Up). In normal conditions (state θ_0), road B is extremely fast ($\alpha_{\mathrm{B}} = 1$ and $\beta_{\mathrm{B}} = 0$). However, it requires frequent road works for maintenance (state θ_1), which increase the travel time. Moreover, it holds $\mu_{\theta_0} = \mu_{\theta_1} = 1/2$. The interaction between the sender and three receivers goes as follow:*

(i) the sender commits to a signaling scheme ϕ;

(ii) the receivers observe ϕ and decide whether to adhere to the navigation system or not;

(iii) the sender observes the realized state of nature and exploits his knowledge to compute recommendations.

Figure 3.1 (Bottom) describes and ex ante persuasive signaling scheme. In this case, when the state of nature is θ_1, one of the receivers is randomly selected to take road B*, even if it is undergoing maintenance. In expectation, following sender's recommendations is strictly better than congesting road* A*.*



	Signals			
	(B, B, B)	(A, A, B)	(A, B, A)	(B, A, A)
θ_0	1	0	0	0
θ_1	0	1/3	1/3	1/3

Figure 3.1. *Example of BNCG.*

Figure 3.1: *Up:* BNCG for Example 3.1. *Bottom:* An *ex ante* persuasive signaling scheme for the case with $n = 3$. The table shows only those $a \in A$ such that $\phi_{\theta,a} > 0$ for some state of nature $\theta \in \Theta = \{\theta_0, \theta_1\}$. The

table shows only action profile $a \in A$ such that $\phi_{\theta,a} \geq 0$.

In the next chapter, we analyze the algorithm developed in order to efficiently compute an *ex ante* persuasive signaling scheme in symmetric BNCGs with affine costs.

Chapter 4

Algorithm Overview

This section presents the mathematical and theoretical foundation of our work. We examine in detail each aspect of the algorithm designed by Castiglioni, Celli, Marchesi and Gatti [4] in order to efficiently compute an optimal *ex ante* persuasive signaling schemes in BNCGs, under the assumptions of symmetry and affine costs. This algorithm, which represents the theoretical basis for our contribution in the present work, is analyzed step by step.

Firstly, a formulation of the problem as a Linear Problem (LP) is considered. Then, we propose the description of how the Ellipsoid method has been applied to the corresponding Dual Problem, to retrieve a solution in polynomial time. The purpose-designed polynomial time separation oracle to solve the separation problem is presented. The mathematical formulation of the problem is the basis of its implementation, which is the subject of the next chapter.

4.1 LP formulation

Our aim is to compute in polynomial time an optimal *ex ante* persuasive signaling scheme in symmetric BNCGs with affine cost functions. The most straightforward approach is to define a Linear Programming (LP) formulation of the problem. For the formulation, we rely on the notation we exposed in the previous chapter. The Problem (4.1), which we refer to as *Primal Problem* in our work, is given in the next page.
For the clearness of LP formulation, notation $I_{\{e \notin a_p\}}$ denotes the Indicator, a function defined such that it holds $I_{\{e \notin a_p\}} = 1$ if $e \notin a_p$, and $I_{\{e \notin a_p\}} = 0$ otherwise. We present each aspect of the demostration to prove the possibility of computing a solution to the problem in polynomial time.

Lemma 1. *Given a symmetric BNCG, an optimal ex ante persuasive signaling scheme ϕ can be found by the following LP:*

LP - Primal Problem

$$\min_{\phi \geq 0, x} \quad \sum_{\theta \in \Theta} \mu_\theta \sum_{a \in A} \phi_{\theta,a} \sum_{p \in N} c_{p,\theta}(a) \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{\theta \in \Theta} \mu_\theta \sum_{a \in A} c_{p,\theta}(a) \phi_{\theta,a} \leq x_{p,s} \qquad \forall p \in N \tag{4.1b}$$

$$x_{p,s} \leq \sum_{\theta \in \Theta} \mu_\theta \sum_{a \in A} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) \phi_{\theta,a} + x_{p,v'} \qquad \forall p \in N, \forall e = (v, v') \in E \tag{4.1c}$$

$$x_{p,t} = 0 \qquad \forall p \in N \tag{4.1d}$$

$$\sum_{a \in A} \phi_{\theta,a} = 1 \qquad \forall \theta \in \Theta \tag{4.1e}$$

Proof: Objective (4.1a) is equivalent to minimizing the social cost while Constraints (4.1e) imply that ϕ is well formed. The set of Constraints (4.1b) implements *ex ante* persuasiveness for every player $p \in N$. In particular, the left-hand side expression represents player p's expected cost, while variable $x_{p,s}$ is the cost of her best deviation, a cost minimizing path given μ and ϕ. This is ensured by Constraints (4.1c) and (4.1d). The former guarantees that $x_{p,v}$ is the minimum cost of a path from a node $v \in V \setminus \{t\}$ to target t. This is shown by noticing that, given $x_{p,t} = 0$ such costs can be inductively defined as:

$$\min_{v' \in V:\ e=(v,v') \in E} \left\{ \sum_{\theta \in \Theta} \mu_\theta \sum_{a \in A} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) \phi_{\theta,a} + x_{p,v'} \right\},$$

where ${f_e}^a + I_{\{e \notin a_p\}}$ stands for the fact that the congestion of edge e must be incremented by one player if the player p does not select a path containing e in the action profile a.

Since the set $|A|$ of action profiles is exponential in the size of the game, the Primal Problem admits polynomially many constraints and

an exponential number of variables. This makes the problem intractable as it is proposed. Therefore, the next step in the description of the process is to provide the dual of Problem (4.1). We can easily derive the *Dual Problem* (4.3) using the principles of LP duality, by introducing the variables y_p (for $p \in N$), $y_{p,e}$ (for $p \in N$ and $e \in E$), $y_{p,t}$ (for $p \in N$), and y_θ (for $\theta \in \Theta$) respectively for Constraints (4.1b), (4.1c), (4.1d) and (4.1e).

Lemma 2. *The corresponding Dual Problem of Problem* (4.1) *is given by the following LP:*

LP - Dual Problem

$$\max_{y} \quad \sum_{\theta \in \Theta} y_\theta \tag{4.3a}$$

$$\text{s.t.} \quad \mu_\theta \left(\sum_{p \in N} c_{p,\theta}(a) y_p - \sum_{p \in N} \sum_{e \in E} c_{e,\theta} (f_e{}^a + I_{\{e \notin a_p\}}) \right) y_{p,e} + y_\theta \leq \mu_\theta \sum_{p \in N} c_{p,\theta}(a) \qquad \forall \theta \in \Theta, \forall a \in A \tag{4.3b}$$

$$\sum_{v' \in V : e=(v,v') \in E} y_{p,e} - \sum_{v' \in V : e=(v',v) \in E} y_{p,e} = 0 \qquad \forall p \in N, \forall v \in V \setminus \{s,t\} \tag{4.3c}$$

$$\sum_{v \in V : e=(s,v) \in E} y_{p,e} - y_p = 0 \qquad \forall p \in N \tag{4.3d}$$

$$y_{p,t} - \sum_{v \in V : e=(v,t) \in E} y_{p,e} = 0 \qquad \forall p \in N \tag{4.3e}$$

$$y_p \leq 0 \qquad \forall p \in N \tag{4.3f}$$

$$y_{p,e} \leq 0 \qquad \forall p \in N, \forall e \in E \tag{4.3g}$$

As it follows from the LP duality, Dual Problem (4.3) admits polynomially many variables and exponentially many constraints. To find an optimal solution in polynomial time for the Problem (4.3) is necessary to exploit the Ellipsoid algorithm. The procedure requires an appropriate polynomial-separation oracle to solve the separation problem. Since the exponential number of constraints of Problem (4.3), a polynomial-time separation oracle was not available and it has been specifically designed

by Castiglioni, Celli, Marchesi and Gatti [4]. In particular, it has been proved that, exploiting the symmetry and affine costs assumptions in BNCGs, Problem (4.3) always admits an *optimal player-symmetric solution*. A player symmetric solution is a vector y such that, for each pair of players $p, q \in N$, it holds that $y_p = y_q, y_{p,e} = y_{q,e}$ for all $e \in E$, and $y_{p,t} = y_{q,t}$. We can therefore restrict our attention to player-symmetric vectors in the definition of a separation oracle for Dual Problem.

Lemma 3. *Dual Problem* (4.3) *always admits an optimal player symmetric solution.*

Proof: Given any optimal solution y to Problem (4.3), we can always recover, in polynomial time, a player-symmetric optimal solution $\tilde{y}$. Specifically, for every player $p \in N$, let $\tilde{y}_p = \frac{\sum_{p \in N} y_p}{n}$, $\tilde{y}_{p,e} = \frac{\sum_{p \in N} y_{p,e}}{n}$ for all $e \in E$, and $\tilde{y}_{p,t} = \frac{\sum_{p \in N} y_{p,t}}{n}$, while $\tilde{y}_\theta = y_\theta$ for every $\theta \in \Theta$. Let us remark that $\tilde{y}$ is player symmetric since: *(i)* for every $e \in E$, it holds that $\tilde{y}_{p,e} = \tilde{y}_{q,e}$ for each pair of players $p, q \in N$; and *(ii)* $\tilde{y}_p = \tilde{y}_q$ and $\tilde{y}_{p,t} = \tilde{y}_{q,t}$ for each $p, q \in N$. First, we can notice that y and $\tilde{y}$ provide the same objective value, as $\tilde{y}_\theta = y_\theta$ for every $\theta \in \Theta$. Thus, we only need to prove that $\tilde{y}$ satisfies all the constraints of Problem (4.3). For $a \in A$ and $i \in [n]$, let us denote with $\pi_i(a)$ an action profile $a' \in A$ such that $a'_p = a_{((p+1) \bmod n)}$, *i.e.* a permutation of a in which each player $p \in N$ takes on the role of player $(p+1) \bmod n$. Moreover, let $\pi(a) := \bigcup_{i \in [n]} \pi_i(a)$. Constraints (4.3b) are satisfied by $\tilde{y}$, since, for every $\theta \in \Theta$ and $a \in A$, it holds:

$$
\begin{aligned}
&\mu_\theta \left(\sum_{p \in N} c_{p,\theta}(a) \tilde{y}_p - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) \tilde{y}_{p,e} \right) + \tilde{y}_\theta \\
&= \frac{1}{n} \sum_{a' \in \pi(a)} \mu_\theta \left(\sum_{p \in N} c_{p,\theta}(a') y_p - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}({f_e}^{a'} + I_{\{e \notin a'_p\}}) y_{p,e} \right) + y_\theta \\
&\leq \frac{1}{n} \sum_{a' \in \pi(a)} \mu_\theta \sum_{p \in N} c_{p,\theta}(a') = \mu_\theta \sum_{p \in N} c_{p,\theta}(a).
\end{aligned}
$$

Similar arguments show that $\tilde{y}$ satisfies all the other constraints, concluding the proof.

4.2 Separation oracle

In this section, we present and analyze the *separation-oracle* designed to compute solutions for the Dual Problem (4.3) in polynomial time.

Formally, a polynomial-time separation oracle for Dual Problem (4.3) is a procedure that, given a vector y of dual variables, it establishes if y is feasible or not. If variable y is not feasible, then the oracle outputs an hyperplane separating y from the feasible region.

Considering the Dual Problem (4.3), we notice that any polynomial-time separation oracle is able to check explicitly if the vector y of dual variable satisfies the sets of Constraint (4.3c), (4.3d), (4.3e), (4.3f) and (4.3g). In fact, the number of constraints of these sets is linear in the number of the players N in the game, and, for set (4.3c), in the number of nodes $v \in V \setminus \{s, t\}$. It permits to focus our attention to the separation problem associated with the exponentially many Constraints set (4.3b). Specifically, the separation oracle in which we are interested is the one that generates constraints from set (4.3b), violated by solution y of Dual Problem (4.3). This procedure allows us to generate only those constraints that are actually useful to solve the problem. Exploiting the *row-generation* algorithm is possible to iteratively populate this subset of the only significant constraints to resolve the Dual Problem.
The separation problem restricted to the exponenatially many set Constraints (4.3b), can be formulated as stated in the following lemma.

Lemma 4. *Given a player-symmetric solution y, solving the separation problem for Constraints* (4.3b) *amounts to finding $\theta \in \Theta$ and $a \in A$ that are optimal for the following problem:*

$$\min_{\theta \in \Theta, a \in A} \mu_\theta \left((1 - \bar{y}) \sum_{p \in N} c_{p,\theta}(a) - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}(f_e{}^a + I_{\{e \notin a_p\}}) \bar{y}_e \right) - y_\theta, \tag{4.5}$$

where we set $\bar{y} = y_1$ and $\bar{y}_e = y_{1,e}$ for all $e \in E$.
It has been proved by Castiglioni, Celli, Marchesi and Gatti [4] that the separation problem (4.5) can be equivalently formulated by avoiding the minimization over the exponentially-seized set A. The underlying idea is that, for a fixed $\theta \in \Theta$, it is possible to exploit the symmetry assumption of BNCGs to represent action profiles $a \in A$ as integer vectors $q_e \in [n]$, which represent the edge congestion for all $e \in E$, *i.e.* the number of players choosing edge e. This leads to the following lemma.

Lemma 5. *Problem* (4.5) *can be formulated as* $\min_{\theta \in \Theta} \chi(\theta)$ *where* $\chi(\theta)$ *is the optimal value obtained solving of the following problem:*

$$\min_{q \in \mathbb{Z}_+^{|E|}} \quad (1-\bar{y}) \sum_{e \in E} \alpha_{e,\theta} q_e^2 + \beta_{e,\theta} q_e - \sum_{e \in E} \bar{y}_e \left(n\alpha_{e,\theta} q_e + (n - q_e)\alpha_{e,\theta} + n\beta_{e,\theta} \right) \tag{4.6a}$$

$$\text{s.t.} \quad \sum_{v \in V : e=(s,v) \in E} q_e = n \tag{4.6b}$$

$$\sum_{v \in V : e=(v,t) \in E} q_e = n \tag{4.6c}$$

$$\sum_{v' \in V : e=(v',v) \in E} q_e = \sum_{v' \in V : e=(v,v') \in E} q_e \qquad \forall v \in V \setminus \{s,t\}. \tag{4.6d}$$

Proof: First, given a state $\theta \in \Theta$, Problem (4.5) reduces to computing $\chi(\theta) := \min_{a \in A}(1-\bar{y}) \sum_{p \in N} c_{p,\theta}(a) - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}})\bar{y}_e$, where the function to be minimized only depends on the number of players selecting each edge $e \in E$ in a, rather than the identity of the players who are choosing e, since they are symmetric. Letting $q_e \in [n]$ be the congestion level of edge $e \in E$ and using the affine costs $c_{e,\theta} = \alpha_{e,\theta} q_e + \beta_{e,\theta}$, it holds $\sum_{p \in N} c_{p,\theta}(a) = \sum_{e \in E} \alpha_{e,\theta} q_e^2 + \beta_{e,\theta} q_e$, and, for every $e \in E$, $\sum_{p \in N} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) = n\alpha_{e,\theta} q_e + (n - q_e)\alpha_{e,\theta} + n\beta_{e,\theta}$. This gives Objective (4.6a). Moreover, Constraints (4.6b), (4.6c) and (4.6d) ensure that q is well defined.

4.3 Min-cost flow problem

The next step is to compute an optimal integer solution for Problem (4.6). This is necessary in order to find a violated constraint for a given solution y and recover an action profile a from q. Reducing Problem (4.6) to an instance of *integer min-cost flow problem* allows us to find an optimal solution in polynomial time. The intuition is that we can consider a modified version of the original graph $G = (V, E)$ in which each original edge $e \in E$ is replaced with n parallel edges between the same pair of nodes, with unit capacity and increasing unit costs. This is possible since Objective (4.6a) is a convex function of q, which is guaranteed by the assumption of affine costs.

Lemma 6. *An optimal integer solution to Problem* (4.6) *can be found in polynomial time by solving a suitably defined instance of min-cost flow problem.*

Proof: First, notice that Objective (4.6a) is a sum edge costs, in which the cost of each edge $e \in E$ is a convex function of the edge congestion q_e, as the only quadratic term in the expression is $(1 - \bar{y})\alpha_{e,\theta} q_e^2$, where the multiplying coefficient is always positive, given $\bar{y} \leq 0$ and $\alpha_{e,\theta} \geq 0$. This allow us to formulate Problem (4.6) as an instance of integer min-cost flow problem. We build a new graph where each $e \in E$ is replaced with n parallel edges, denoted by e_i for $i \in [n]$. We need also to introduce some extra notation. Let's define:

$$g(e, i) := (1 - \bar{y})(\alpha_{e,\theta} i^2 + \beta_e i) - \bar{y}_e(n\alpha_{e,\theta} i + (n-1)\alpha_{e,\theta} + n\beta_{e,\theta}) \qquad \forall e \in E, i \in [n].$$

Each new edge e_i introduced in the modified version of the graph has unit capacity and a per-unit cost equal to:

$$\delta(e_i) := g(e, i) - g(e, i-1).$$

Finding a solution for the integer min-cost flow problem in the modified graph is equivalent to minimizing Objective (4.6a). Since the original edge costs are convex, it holds $\delta(e_i) \geq \delta(e_j)$ for all $j < i \in [n]$. Thus, an edge e_i is used (*i.e.* it carries a unit of flow) only if all the edges e_j for $j < i \in [n]$, are already used. Exploiting this idea, we can recover an integer vector q from a solution to the min-cost flow problem in polynomial time by solving its LP relaxation. This leads us to the main result proposed by Castiglioni, Celli, Marchesi and Gatti [4] which guarantees to find an optimal solution in polynomial time to our problem.

Theorem 1. *Given a symmetric BNCG, an optimal ex-ante persuasive signaling scheme can be computed in polynomial time.*

Proof: The algorithm applies the ellipsoid algorithm to Problem (4.3). At each iteration, we require that the vector of dual variables y given to the separation oracle be player-symmetric, which can be easily obtained by applying the symmetrization technique introduced in the proof of Lemma 3. The separation oracle needs to solve an instance of integer min-cost flow problem for every $\theta \in \Theta$. An integer solution is required in order to be able to identify a violated constraint. Finally, the polynomially many violated constraints generated by the ellipsoid algorithm can be used to compute an optimal ϕ.

In the next chapter, we present the implementation of the described algorithm.

Chapter 5

Algorithm Implementation

In the previous chapter we analyzed from a theoretical point of view the problem of computing an optimal *ex-ante* persuasive signaling scheme in polynomial time for symmetric BNCGs, under the assumption of affine costs.

The aim of the present chapter is to show how we implemented the algorithm to pratically solve the problem and output a solution. The algorithm has been implemented in the Python programming language. The Linear Problems have been solved by Gurobi optimization solver [9], interfaced with our code through the Python module `gurobipy` [10].

In the first section, we discuss how we implemented a generic instance of a BNCG. Then, we present the method we used to write and solve the Linear programming problems in our project. Finally, we describe in detail each step of the algorithm and its implementation.

5.1 Graph Instance Generation

We start by focusing on the main elements of the problem. A Jupyter Notebook script named *TestGenerator* was developed in Python language in order to generate a random directed graph *G = (V, E)* as a collections of nodes along with the identified pairs of edges. The generated graph is then used as an network instance in a BNCG.

To generate a random graph, we relied as a starting point on the Python package called `NetworkX` [15]. NetworkX is a powerful tool for the creation, manipulation and study of complex networks in Python language, that provides network structures and methods which we can work on. Using this package, a graph G in our code is implemented as

an instance of the NetworkX class *DiGraph*, a base class representing directed networks, which stores nodes and edges with optional data. On the other hand, NetworkX has some limits when we want to generate a graph for a Bayesian Network Congestion Game, in particular a graph in which there are a source and a target nodes, and there are only paths connecting these nodes. The way to proceed is therefore to combine the tools NetworkX offers us with functions designed on purpose. Specifically, NetworkX provides the function *random_k_out_graph*(), a directed graph generator which takes as input the number n of nodes of the returned graph, the out-degree k of each node and a parameter *alpha* representing the initial weight of each node. Then, the function outputs a random directed network, generated accordingly to the algorithm specified in the documentation [15] and by the input parameters.

The resulting graph does not yet meet the criteria for use in a BNCG, since it is just a random directed graph, without any specific source and destination pair nodes. The graph is then passed to a function called *build_graph*(), which takes as input also an integer number indicating the length of the minimum path desired in the final *graph*. In this way all the paths from the source and destination have a number of edges greater or equal than the length indicated. The function *build_graph*() has the task of identifying a pair of two nodes in the input graph, such that the minimum path linking them has a length greater or equal than the one specified as input. If no such path exists between any pair of nodes in the input graph, a new graph with new parameters must be generated. If two such nodes exist, then the subgraph consisting of all nodes and edges on a path between them is returned. This two nodes are label as the *source* s and the *destination* t in the network that is used as a base to implement an instance of Bayesian Network Congestion Game. Finally, we provide the possibility to save the graph instance in a .pickle file.

5.2 Problem Instance Generation

This section analyses the implementation of a generic instance of Bayesian Network Congestion Game. We have implemented in Python some functions that take care of creating all the aspects of a BNCG. We start by creating or loading a graph. For the first option we can use the Jupyter Notebook script *TestGenerator*, described in the previous section, that generates a random directed graph with paths from a source s to a target t. For the second option, we use files .pickle, a binary serialization format used to store data in our project.

We present here how we defined in Python the elements of a BNCG, that are used by the algorithm to compute an optimal signaling scheme.

- *State of nature* Θ: it is simply an integer value denoting the number of states of nature in the game;
- *probability distribution* μ: it is a list of float values, one for each $\theta \in \Theta$, which represent probability values provided by the implemented method called *gen_probability_distribution*() from the Dirichlet distribution;
- *players* N: it is simply an integer value denoting the number of players participating in the game;
- *affine values* α, β: they are random integer values generated between 1 and 5, including external values, for each edge $e \in E$ and for each $\theta \in \Theta$. They are stored in Python dictionaries, keyed by edge e and state θ.

We remind that the costs of an edge $e \in E$ in a BNCG with affine cost functions, is given by the result of $c_{e,\theta}({f_e}^a) := \alpha_{e,\theta}{f_e}^a + \beta_{e,\theta}$. As we see in the next section, the value ${f_e}^a$, which indicates the congestion of edge e in action profile a, is dynamically provided during the execution of the algorithm by the resolution of the Dual Problem. The same reasoning applies to cost experienced by a player $p \in N$ in an action profile $a \in A$ in a state of nature θ, which is computed as $c_{p,\theta} := \sum_{e \in a_p} c_{e,\theta}({f_e}^a)$. This happens because, of course, at the time of generation of a BNCG, the edges are no populated by the players. We simply initialise in our code an empty dictionary *edge_cong*() keyed by the edge e and state of nature θ for the edges congestion. Same for the dictionary *edge_costs*(), related to the costs. We describe in the next section how these values are retrieved.

All the data produced will be used by the solver and are saved in a .pickle file, so that it is possible to create a benchmark to evaluate the efficiency of the implemented algorithm.

5.3 Generic LP Implementation

This section describes how to implement a generic LP that will be solved by the Gurobi optimization solver. The Gurobi Python interface `gurobipy` provides us all the tools to create and solve one or more linear programming problems in our work using Python.

The Figure 5.1 shows the typical structure to follow and the provided methods to build a model instance in `gurobipy` environment.

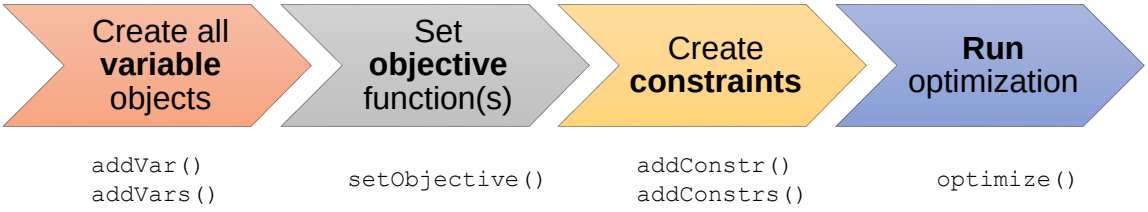


Figure 5.1. *Basic structure in building a model with Guropi Python API.*

As we see from Figure 5.1, `gurobipy` module allows us to consider and implement the separated elements in a model, making it easy to write and read the Linear Problem. The implementation of a model in Gurobi is quite clear straightforward, and all the aspect of the implementation are here analyzed.
The following elements are the main parts in creating a model:

- *variables*: these are the decision variables of the model. Methods *addVar()* and *addVars()* are used to add one or multiple variables to our model;
- *objective function*: the mathematical function of the model that the solver will try to optimize. The function *setObjective()* is used to express the objective function of the model and specify the optimization sense (`GRB.MINIMIZE` for minimization, `GRB.MAXIMIZE` for maximization);
- *constraints:* the set of functional equalities or inequalities that the solver must consider in its optimisation procedure. To add one or multiple constraints to our model, we use *addConstr()* and *addConstrs()*;
- *optimization:* the function *optimize()* runs the optimization of the model.

By following this scheme, we are able to write and resolve a LP in `gurobipy` setting.

5.4 Algorithm Implementation

Let us focusing on the specific implementation of the algorithm, the main part of our work. We based the resolution of the linear problems on Gurobi [9], a powerful and well-known commercial optimization solver

for, among others, linear programming. To adapt the Gurobi interface to our Python code, we relied on Python module `gurobipy`.

We start by introducing the specific structure of the Dual Problem as a LP in `gurobipy`. Then, it is presented the implementation of the separation oracle with the row-generation algorithm applied to our project. Finally, we show how to retrieve, using the data previously computed, an optimal *ex ante* signaling scheme in polynomial time by resolving the Primal Problem.

5.4.1 Implementing the Dual Problem

Let us now describe how the algorithm for computing an optimal *ex-ante* persuasive signaling scheme in polynomial time for a BNCG was implemented.
The solver was entirely implemented using Jupyter Notebook, a web application that allows us to create and share documents that contain equation, data, visualizations and more. The Jupyter Notebook environment supports different programming languages, including Python and it can produce a single file for the solver with the extension .py.
For the initialization, the solver reads the data of the Bayesian Network Congestion Game instance, generated as described in previous section.
The implementation and the resolution of the Dual Problem is described by the following steps:

1. the solver reads the data from the Bayesian Network Congestion Game file;

2. a partial LP model of the Dual Problem is generated. In particular set of Constraints (4.3b) is empty;

3. purpose generated constraints are added to set (4.3b) by considering a generic action profile a. Edge costs $c_{e,\theta}$ and paths costs $c_{p,\theta}$ are computed;

4. the optimization of Dual Problem is run. A symmetric solution $\bar{y}$ is retrieved;

5. a modified version of the graph $G = (V, E)$ is built to compute new edge congestions, using the symmetric solution $\bar{y}$ by solving an instance of min-cost flow problem for every $\theta \in \Theta$. This is the task of the separation oracle;

6. the separation problem is solved. If solution is ≥ 0 go to Step 7. Otherwise add to set (4.3b) the Constraints associated with θ and

a that are solution for the separation problem. Then, go to Step 4.

7. The row-generation algorithm is finished and we provided a set of Constraints (4.3b) to solve Dual Problem.

All the steps are analyzed in details in the present chapter.
The first notable step was to write the Linear Program (4.3), concerning the Dual Problem. Since it has been shown that the Dual Problem always admits an optimal player-symmetric solution, as described in Section 4, it was possible to reduce the number of decision variables of the LP. In fact, we don't need to define for the Dual Problem all the variables y_p for $p \in N$, $y_{p,e}$ for $p \in N$ and $e \in E$, variables $y_{p,t}$ for $p \in N$, but we can respectively define variables y, y_e for $e \in E$ and y_t. This significantly reduces the number of variables of the Dual Problem to work with.

Writing the LP for the Dual Program using `gurobipy` is then straightforward, using the steps defined in the section above and considering only the useful variables. The crucial aspect in the implementation is related to the set of Constraints (4.3b), reported by the following expression:

$$\mu_\theta \left(\sum_{p \in N} c_{p,\theta}(a) y_p - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) y_{p,e} \right) + y_\theta \leq \mu_\theta \sum_{p \in N} c_{p,\theta}(a) \qquad \forall \theta \in \Theta, \forall a \in A$$

which produces an exponential number of constraints. Since it is inefficient to explicitly check if all the constraints from set (4.3b) are satisfied for a given solution y of Dual Problem, we therefore apply the *row-generation* algorithm. So, the set of Constraints (4.3b) is initialised as *empty* and it is populated during the algorithm running process, with only the constraints we actually need to solve the LP. However, initialising the set (4.3b) as completely empty has a negative effect since a first execution of the so determined reduced master problem associated to the Dual Problem would declare the model *unbounded*, given the Gurobi status code `UNBOUNDED`. It happens since there is no constraint that limits the values that variables y_θ can assume, whose sum we need to maximise in the objective function.

To solve the complication of the empty set, we designed for each $\theta \in \Theta$ one simple action profile a on which all the players choose the same path from the source s to the target t. In order to do that, we randomly select a path from s to t from those available. We specify that in our project a path is implemented as a list of nodes in which no node appears more than once (*i.e* there are no loops), and each adjacent pair of nodes in the list is adjacent in the graph. An example of path in our code is given by the following list: $[s, 1, 7, 10, t]$. Then, a designed method $build\_action\_profile()$, given as input a state of nature θ, computes an action profile a as a list of paths used by players in that state of nature, one path per player. Having now an action profile and so a congestion ${f_e}^a$ for every edge $e \in E$ for this action profile a, is possible to compute the edge costs $c_{e,\theta}({f_e}^a)$ and hence the path costs $c_{p,\theta}$. This is also the task of $build\_action\_profile()$ method. With this fixed generated action profile, we can add a constraint to the empty set (4.3b), one for each $\theta \in \Theta$. Now that all the elements are loaded in memory, a timer to record the amount of time of execution of algorithm is started. The first Gurobi optimization of the Dual Problem is run.

The values returned by the optimization of the Dual Problem by the Gurobi solver are variables y_e for $e \in E$ which is implemented as a Python dictionary keyed by edge e, a single variable $\bar{y}$ and variables y_θ for $\theta \in \Theta$. In the next sections we describe how the separation problem 4.5 is implemented using the previous solution of Dual Problem. The separation oracle to resolve the problem is presented.

5.4.2 Min-cost flow Problem

In the present section we explain how the *separation problem*, restricted to the exponential set of Constraints (4.3b), for the Dual Problem has been implemented and resolved by the separation oracle. We remember that mathematical formulation of the separation problem is the following:

$$\min_{\theta \in \Theta, a \in A} \mu_\theta \left((1 - \bar{y}) \sum_{p \in N} c_{p,\theta}(a) - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) \bar{y}_e \right) - y_\theta \tag{5.2}$$

In order to solve the separation Problem (5.2) in polynomial time, we solve instead a suitably defined instance of *min-cost flow problem* for each

$\theta \in \Theta$. To achieve this purpose, we build a modified version of our original graph G, in which each edge $e \in E$ is replaced with n parallel edges, with n number of players in the BNCG, with unit capacity and increasing unit costs. The unit cost implies that each edge e_i can carry a unit of flow in the min-cost flow problem. To accomplish this task, we designed a function called *build_parallel_graph*() which simply takes as parameters the original *graph* and the number of players n and output a new graph, named *newGraph*, as an instance of the NetworkX class *MultiDiGraph*, which allows in a graph the presence of multiple edges between the same pair of nodes, built accordingly to the previous description. For each $\theta \in \Theta$ and new edge e in *newGraph*, a simple function called *calculate_g*() takes as input an edge e and a player i and calculates $g(e,i) := (1-\bar{y})(\alpha_{e,\theta} i^2 + \beta_e i) - \bar{y}_e(n\alpha_{e,\theta} i + (n-1)\alpha_{e,\theta} + n\beta_{e,\theta})$. This function is used to compute also the per unit cost $\delta(e_i) := g(e,i) - g(e,i-1)$ for each new edge e_i in *newGraph*.

We have now to solve an instance of *integer min-cost flow problem* on the network *newGraph* for every $\theta \in \theta$, considering as edge costs the quantities $\delta(e_i)$. This is the task of the separation oracle. To implement the separation oracle, we rely on the function *min_cost_flow*() provided by the Python module NetworkX, which takes as input our instance of *newGraph* and outputs a dictionary keyed by the edge to identify the minimum cost flow satisfying all the demands in *newGraph*. These values represent the edges congestions in *newGraph* and we can easily recover the congestions in our original *graph*, since the edge congestion of an edge e in *graph* can be computed as the sum of flow of the edges between the same pair of nodes in *newGraph*, *i.e.* $q_e = \sum_{i \in N} flow(e_i)$ for $e \in E$. The separation oracle is used to compute edge congestion in original graph for each $\theta \in \Theta$.

5.4.3 Adding the new constraint

At the end of the previous step, using the separation oracle we have recovered as a solution of the min-cost flow problem a dictionary of edge congestion q for every state of nature θ. By using again the method *build_action_profile*(), we are able to retrieve from a dictionary of edge congestion q, the corresponding action profile a and compute path costs c_p for every $\theta \in \Theta$. We can implement the next step, the minimization over the state of nature θ and its associated action profile a to provide a solution for the separation Problem (5.2).

The minimization step simply consists in computing all the values of

expression (5.2) for each pair of the considered values of θ and a and then select the minimum. These are the values of $\theta \in \Theta$ and $a \in A$ which are optimal for (5.2) and depending on the value assumed by the minimum, two scenarios are possible:

- *minimum* ≥ 0: we don't need to add new constraints to the LP model of the Dual Problem. We found a subset of Constraints (4.3b) to be able to find an optimal solution.
- *minimum* < 0: the constraint, associated with the pair θ and a which have given the minimum value, is violated. We have to add this new constraint to the LP model of the Dual Problem and continue with the algorithm. We can simply use the function *addConstr*() provided by `gurobipy` to add a new constraint in set (4.3b).

The violated constraint is added to set (4.3b) of Dual Problem:

$$\mu_\theta \left(\sum_{p \in N} c_{p,\theta}(a) y_p - \sum_{p \in N} \sum_{e \in E} c_{e,\theta}({f_e}^a + I_{\{e \notin a_p\}}) y_{p,e} \right) + y_\theta \leq \mu_\theta \sum_{p \in N} c_{p,\theta}(a) \tag{5.3}$$

in which the action profile a and state of nature θ specified in expression (5.3) are the values associated to the minimum. We save in two lists the action profile a and the paths costs c_p, which are used in the next step of the algorithm. A new optimization of the Dual Problem with `gurobipy` is run, in order to repeat the process of computing a new violated constraint or verify that we reached the required number of constraints. As we said, if the minimization over θ and a returns a values greater than zero, the row-generation algorithm stops, since we have a satisfactory number of constraints to solve Dual Problem.

5.4.4 Primal Problem

We reach this step when the row-generation algorithm is finished and therefore it has provided a subset of Constraints (4.3b) sufficient to optimally solve the Dual Problem. We built during the row-generation algorithm an action set A as a list containing every action profile a which is optimal for each θ for the separation Problem (5.2). Exploiting the *symmetric* assumption of BNCGs, we can recover from each action profile $a \in A$ and $i \in [n]$ a new action set A_new in which every action profile $a' \in A\_new$ such that $a'_p = a_{((p+1) \bmod n)}$ is a permutation of the original a in which each player $p \in N$ takes the role of player $(p+1) \bmod n$.

To visualize more clearly the permutation, we provide an example in Table 5.1. Considering an initial action profile $\boldsymbol{a} = \{a_1, a_2, a_3, a_4, a_5\}$ which represents a list the actions, *i.e.* path from source s to target t, used by a set of 5 players. Specifically, each a in position p_k of the array denotes a path used by the player k. The index i in a_i denotes the particular path, so if $i \neq j$ then $a_i \neq a_j$.

Table 5.1. *Permutation of an action profile.*

	p_1	p_2	p_3	p_4	p_5
$\boldsymbol{a}$	a_1	a_2	a_3	a_4	a_5

	p_1	p_2	p_3	p_4	p_5
$\boldsymbol{a_1}$	a_5	a_1	a_2	a_3	a_4
$\boldsymbol{a_2}$	a_4	a_5	a_1	a_2	a_3
$\boldsymbol{a_3}$	a_3	a_4	a_5	a_1	a_2
$\boldsymbol{a_4}$	a_2	a_3	a_4	a_5	a_1

Table 5.1: *Left*: the original action profile. *Right*: the action profiles obtained through permutation.

We can now write the LP related to the Primal Problem, to retrieve the optimal signaling schemes as a solution of our problem. The Linear Program is simply implemented using the tools provided by Python module `gurobipy`, as we explained in Section 5.3. We have of course to consider, for each $\theta \in \Theta$ as set A of action profiles, the set generated by the permutation of each action profile a obtained from the running of the row generation algorithm. Running the LP just written, provides us a distribution over the optimal signaling scheme for each $\theta \in \Theta$. As we clearly see, as the number of possible action profile has been drastically reduced, the LP can calculate an optimal *ex-ante* persuasive signaling scheme in polynomial time.

5.5 Solver Outputs

The output of the algorithm is composed of a list of values, saved in a .pickle file. More precisely, in addition to the computational time used to find a solution, we save for each $\theta \in \Theta$ a list of the action profiles a which have a probability in the distribution greater than zero, and the associated probability value.

In our experiments proposed in the next chapter, we consider for our analyses only the computational time taken by the algorithm to compute a solution for the problem.

Chapter 6

Experimental Results

In this chapter, we present the experimental results of the project. We generated and tested different problem instances, with the intention of analysing the performances of our algorithm as individual parameters change.

We implemented a random Bayesian Network Congestion Game generator in Jupyter Notebook to provide us instances to work on. The algorithm has been implemented in Python programming language. Each LP has been solved by Gurobi [9] optimization solver, invoked using Python library `gurobipy` [10].

Specifically, we solve 1600 instances of Bayesian Network Congestion Games, by varying parameters such as number of nodes in the graph, the states of nature and the number of players.

6.1 Experimental setting

In this section, we present the experimental problem setting. We indicate the hardware and software tools specifications used for the experiments. The dataset specifications on which we run the algorithm are presented.

6.1.1 Hardware and Software

We have performed all the experiments using a UNIX machine with 32 cores working at 2.3 GHz and equipped with a RAM of 125 GB. Each LP is solved using Gurobi [9] optimization solver, invoked by the Python library `gurobipy` which provide an interface to our code. The algorithm has been implemented in Python programming language.

6.1.2 Dataset

We have implemented a random Bayesian Network Congestion Game generator in Python that allow us to create and save instances to test.

To evaluate the implemented algorithm, we run experiments on graphs from 4 groups of node numbers, specifically graphs with 30, 60, 90 and 120 nodes. For each number of nodes we generated 10 random graph instances with the same number of nodes. The number of players varies between 10 and 100 at intervals of 10, $N = \{10, 20, ..., 90, 100\}$, while the number of states of nature varies between 10 and 40 at intervals of 10, so $\Theta = \{10, 20, 30, 40\}$. We generated and resolved in total 1600 Bayesian Network Congestion Games. We set a time limit of 3600 seconds for each game instance. If the limit is reached during the execution, the algorithm returns the value `TIME LIMIT`, that indicates that no solution has been found within the given time.

Figure 6.1 shows the quantities related to the network instances considered in the experiments, specifically the number of nodes |V|, the average number of edges |E| and the minimum path length from a source and target.

	Network instances		
Experiment	**|V|**	**avg |E|**	**min l**
First	30	76	3
Second	60	165	5
Third	90	257	7
Fourth	120	348	9

Figure 6.1. *Network dataset instances.*

6.2 Test Results

The aim of this section is to report the performances of the implemented algorithm, through the results obtained in the tests. Each experiment corresponds to a groups of graph instances with a fixed number of nodes, on which we build and run BNCGs. Each result reported is the average of the values obtained by running 10 different instances. It is clear that, fixing the number of nodes in the graphs, the execution time for resolving a BNCG instance strictly depends on the *number of players* and the *number of states of nature* involved.

6.2.1 First Experiment

In the first experiment we begin our discussion by considering small network instances with 30 nodes and a minimum path length of 3 edges between the source and the target nodes. Figure 6.2 documents the computational times taken by the algorithm, reporting the number of players involved in the game as values of x-axis.
Since the number of nodes and the average of edges in the considered graphs are quite small, we can expect in this experiment a slight impact on the computing time.

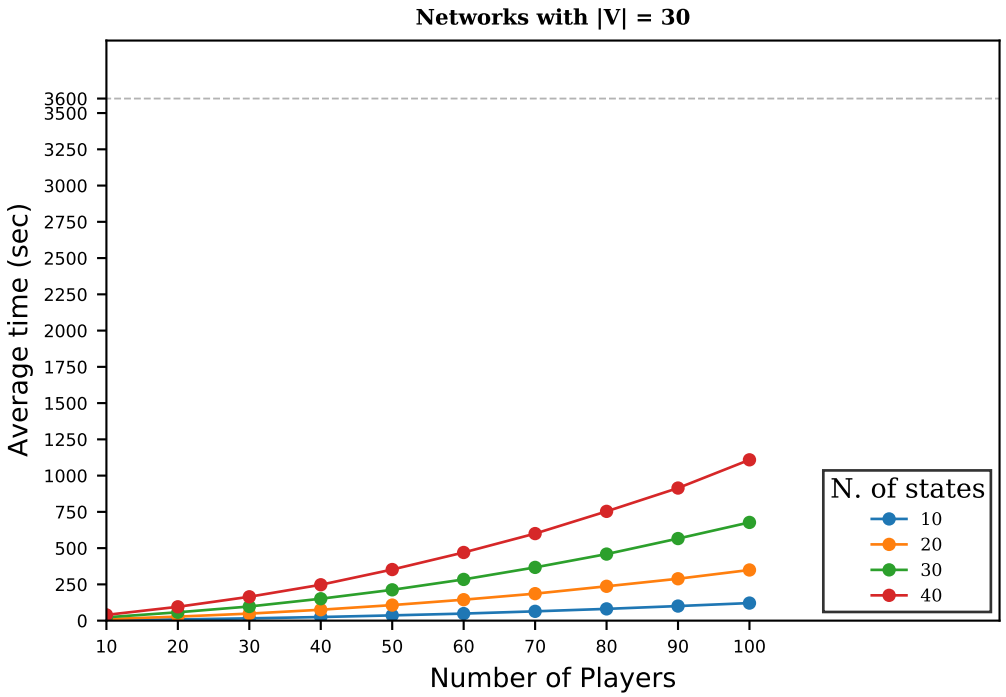


Figure 6.2. *Computing times for Networks with* V = 30 *and varying the number of states.*

The graphical limit in Figure 6.2 is that each point represented in the graph is the average of 10 different experiments on graphs and therefore it is not possible to understand the dispersion of the resulting values for a single game instance. To analyze in the detail this issue, we report in Figure 6.3 the box plots related to the computing time of all game instances in which 100 players are involved in a network with 30 nodes.

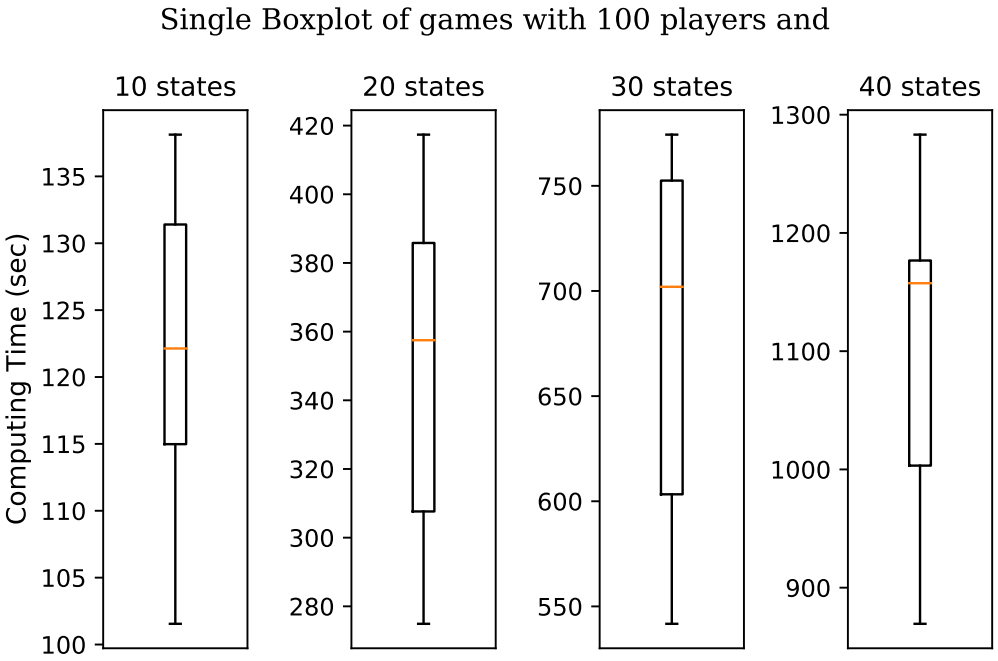


Figure 6.3. *Single box plots of games instances with 100 players in networks with* $|V| = 30$.

We can easily note from Figure 6.3 that when we have a small number of states of nature as 10, the computing times of different instances are very similar. This is quite clear since we are considering a graph with a small number of nodes. Nevertheless, even with a small network with 30 nodes, in the game instances with 40 states of nature, the difference of the computing time can be of 47,6% between different tests. The specific statistical indices in terms of seconds, are shown in Figure 6.4.

	Number of states			
	10	20	30	40
Min	101.54	274.91	541.72	869.44
Max	138.12	417.37	774.35	1283.13
Range	36.56	142.46	232.63	413.69
Mean	121.13	349.33	677.32	1108.97
Variance	140.37	2064.09	6539.34	16621.35

Figure 6.4. *Statistical indices for game instances with 100 players in networks with* $|V| = 30$.

6.2.2 Second Experiment

In the second experiment, we consider network instances with 60 nodes and a minimum path length of 5 edge between the source and target nodes. Figure 6.5 presents the results of the experiments, reported by fixing for each curve the state of nature and by varying on the x-axis the number of players involved in each game.

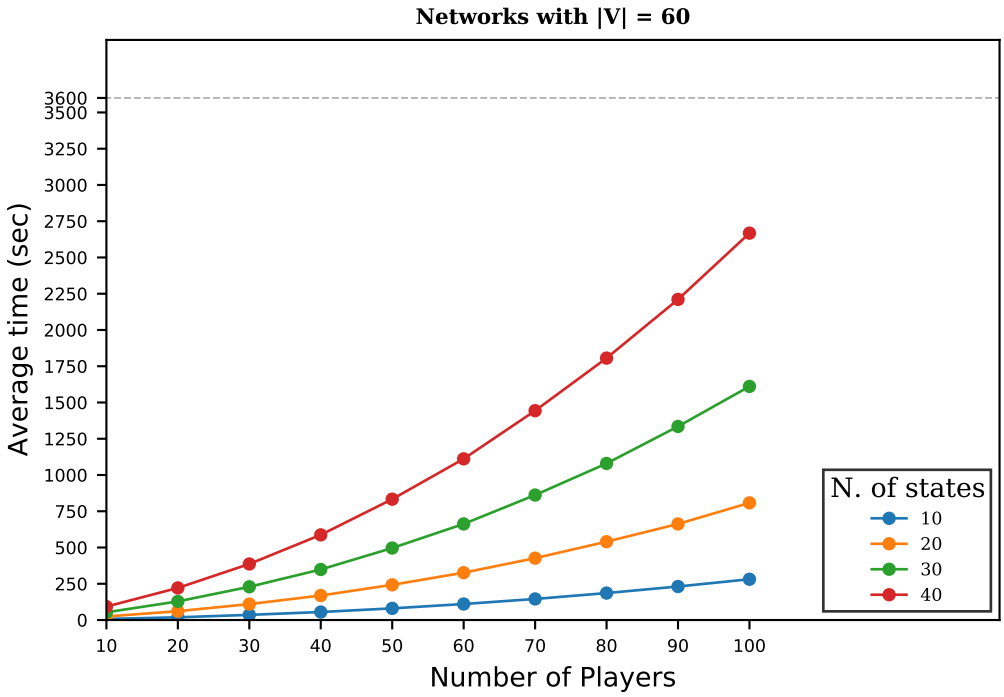


Figure 6.5. *Computing times for Networks with* |V| = 60 *and varying the number of states.*

We can see from Figure 6.5 that, by increasing the number of states of nature, the execution time increases more massively. Nevertheless, even in the case with the highest number of players and states of nature, respectively 100 and 40, in this experiments we never reached in any instance our time limit of 3600 seconds. As in the previous experiment, we report in Figure 6.6 the box plots related to the computing time of all game instances in which 100 players are involved in a network with 60 nodes.

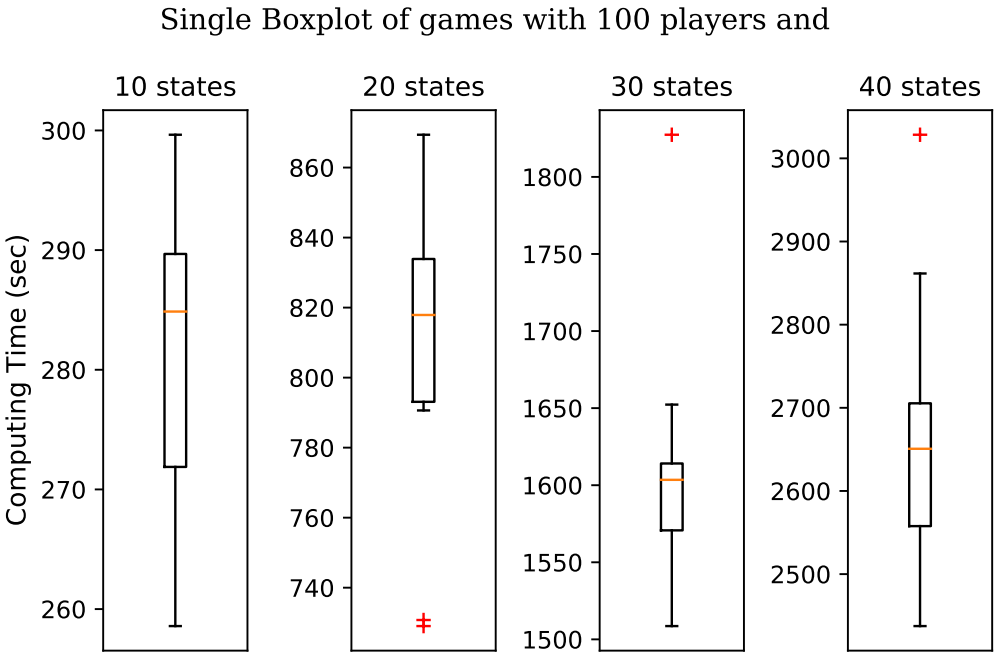


Figure 6.6. *Single box plots of games instances with 100 players in networks with* $|V| = 60$.

In Figure 6.6, even if the number of nodes in the networks is still quite small, we now observe outliers along with small range of values. It indicates that in particular game instances the computational time was distant from the rest of the data. This is clearly observable in boxplots related to 30 and 40 states of nature. Figure 6.7 shows the statistical indices for the second experiment, in terms of seconds of computing time.

	Number of states			
	10	20	30	40
Min	258.58	729.03	1508.61	2437.51
Max	299.64	869.37	1827.35	3028.39
Range	41.06	140.34	318.74	590.88
Mean	281.22	807.75	1610.80	2667.71
Variance	160.76	2009.82	6822.06	27479.09

Figure 6.7. *Statistical indices for game instances with 100 players in networks with* $|V| = 60$.

As it was to be expected, the variance and the range of computing time increases with the number of states of nature.

6.2.3 Third Experiment

As a third experiment, we consider network instances with 90 nodes and a minimum path length of 7 edges between the source and target nodes. Figure 6.8 reports the computational times for this experiment, fixing for each line the number of states of nature involved in the games.

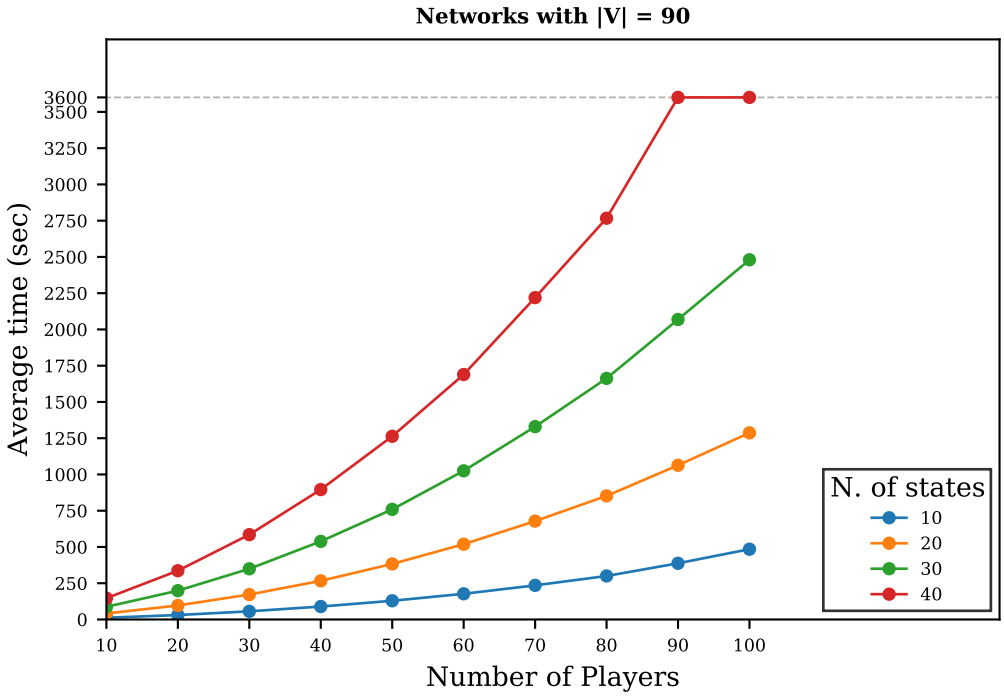


Figure 6.8. *Computing times for Networks with* |V| = 90 *and varying the number of states.*

We can notice that in this experiment the time limit of 3600 seconds has been reached in BNCGs instances with 100 players and 40 states of nature, specifically in the 100% of the cases. This is quite clear since in game instances with 40 states of nature and 90 players, the game instance immediately preceding, the time limit has been reached in 20% of the cases and the average computational time of the remaining solved games in that parameters setting was 3314 seconds.
Since in these game instances the computation has been stopped once reached the time limit, we report in Figure 6.9 the statistical indices of the last available tests for game instances with 100 players, involving respectively 10, 20 and 30 states of nature.

	Number of states		
	10	20	30
Min	451.99	1139.13	2251.11
Max	543.61	1472.27	2862.60
Range	91.62	333.14	611.49
Mean	484.60	1286.87	2480.32
Variance	754.24	7938.32	36148.05

Figure 6.9. *Statistical indices for game instances with 100 players in networks with* $|V| = 90$.

Additionally, we report in Figure 6.10 the tests for 40 states and 80 players, the last parameters setting before reaching the time limit.

	Number of states
	40
Min	2520.84
Max	3151.25
Range	630.41
Mean	2766.93
Variance	40370.57

Figure 6.10. *Statistical indices for game instances with 80 players and 40 states in networks with* $|V| = 90$.

Comparing the statistical indices, we observe that with 30 and 40 states of nature the computational time explodes with respect to the previous cases, and variance is 36148.05 seconds and 40370.57 seconds, with respectively 100 and 80 players involved.

6.2.4 Fourth Experiment

In the fourth experiment we start by graphs instances with 120 nodes and a minimum path length of 9 edges. In this test, we reached the time limit of 3600 seconds for different BNCG instances. Specifically, we reached the time limit in games instances with 30 players and 100 states of nature in every test. On the other hand, the time limit has been reached in the 10% of the tests with 30 states of nature and 90 players. For games with 40 states of nature, the time limit has been reached with 80, 90 and 100 players in all the cases.

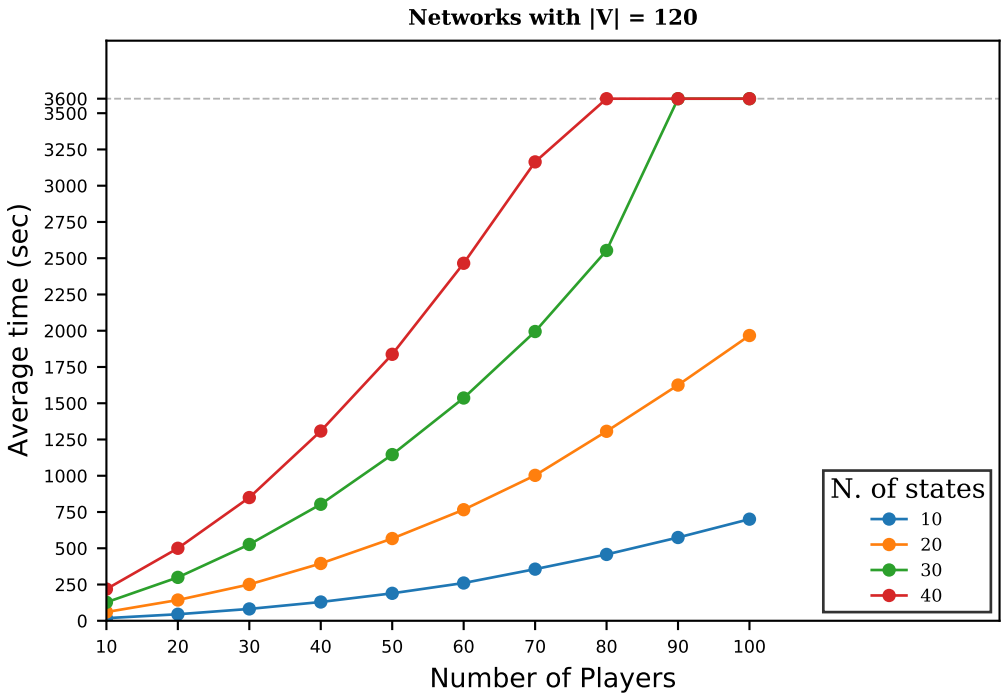


Figure 6.11. *Computing times for Networks with* |V| = 120 *and varying the number of states.*

As done in previous experiments, we provide in Figure 6.12 the statistical indices for games instances with 100 players. Because of the time limit, the attention is restricted to game with 10 and 20 states of nature.

	Number of states	
	10	20
Min	645.24	1799.57
Max	774.31	2331.98
Range	129.07	532.41
Mean	700.64	1966.99
Variance	1284.24	20822.56

Figure 6.12. *Statistical indices for game instances with 100 players and with 10 and 20 states, in networks with* |V| = 120.

Additionally, Figure 6.13 and Figure 6.14 respectively document the statistical indices for the last game instances solved with 30 and 40 states of nature in networks with 120 nodes.

	Number of states
	30
Min	2380.92
Max	2960.08
Range	579.16
Mean	2553.467
Variance	21253.21

Figure 6.13. *Statistical indices for game instances with 80 players and 30 states in networks with* |V| = 120.

	Number of states
	40
Min	2983.27
Max	3313.31
Range	330.03
Mean	3164.17
Variance	11582.74

Figure 6.14. *Statistical indices for game instances with 70 players and 40 states in networks with* |V| = 120.

6.3 Comparison between Experiments

The algorithm has performed well in networks with 30 and 60 nodes, and this is clear since they are very small graphs and the time limit has never been exceeded. For networks with 90 and 120 nodes, even if they consist in a small increase in the amount of nodes, the time limit was reached several times as described in previous sections. In Figure 6.15 we report the limits of the algorithm in the sense of the last game that the algorithm is able to solve within the time limit, along with the associated maximum computing time.

N. of nodes	**N. of states**	**N. of players**	**Computing time (sec)**
90	40	80	3151
120	30	80	2960
120	40	70	3313

Figure 6.15. *Limits of the algorithm.*

To show in detail the performance of the algorithm in the border cases, Figure 6.16 reports the percentage of the solved games in the setting in which the time limit of 3600 seconds has been reached in at least one instance of that game.

N. of nodes	**N. of states**	**N. of players**	**% of solved games**
90	40	90	80 %
90	40	100	0 %
120	30	90	90 %
120	30	100	0 %
120	40	80	0 %
120	40	90	0 %
120	40	100	0 %

Figure 6.16. *Percentage of solved games within the time limit.*

Finally, to sum up the performance of the algorithm in the games involving 100 players, Figure 6.17 shows the computational times reporting the number of nodes in the networks as values of x-axis. With a small number of states of nature, the average computational time remains at low values, even in networks with 120 nodes. Instead, immediately from 20 states of nature, we notice a sudden increase of the computational time from networks with 90 nodes.

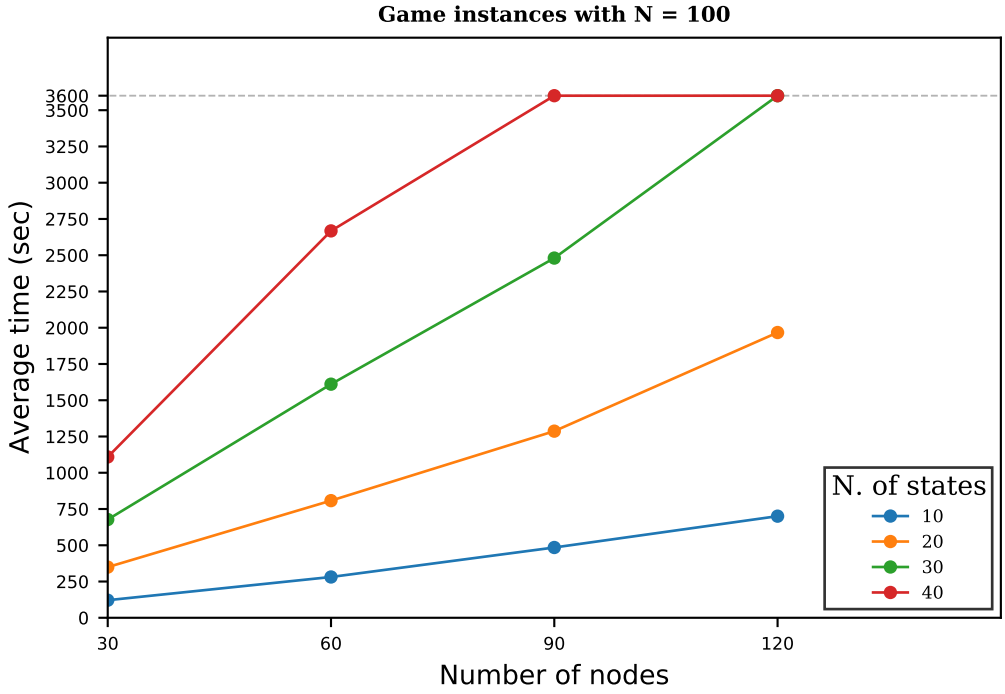


Figure 6.17. *Computing times for game instances with* N = 100 *and varying the number of nodes in network.*

6.4 Regression

In this section, we analyse some of the experiments in order to establish a mathematical dependence between the running time to provide a solution and the number of states of nature and players involved in a game.

We consider graph instances with 90 nodes and BNCGs with 90 players. Executing regression with Python, we conclude that the curve for that BNGCs, considered by varying the number of players, can be approximated with a polynomial function with coefficients equal to 0.196, 5.112 and 20.413, considered to highest power to lower.
Figure 6.18 shows the polynomial function obtained.

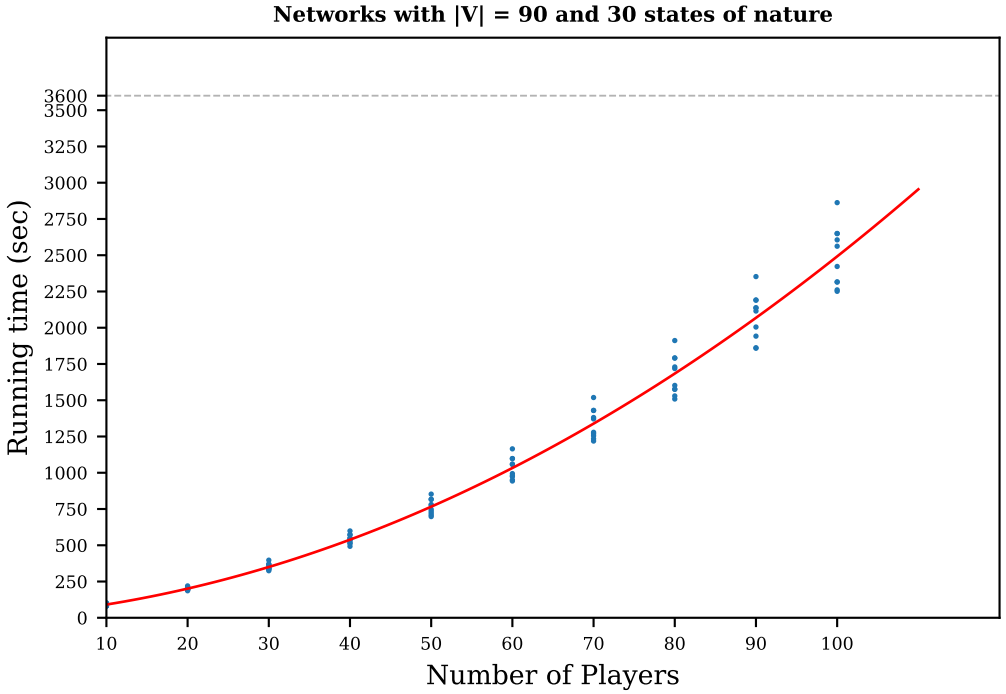


Figure 6.18. *Regression for game instances with* $|\mathrm{V}| = 90$, $|\theta| = 30$ *and varying the number of players in the game.*

On the other hand, Figure 6.19 depicts the function which optimally approximates the curve obtained fixing the number of players. As expected, the function is polynomial with coefficients equal 1.568, 21.163 and 17.435, considered to highest power to lower.

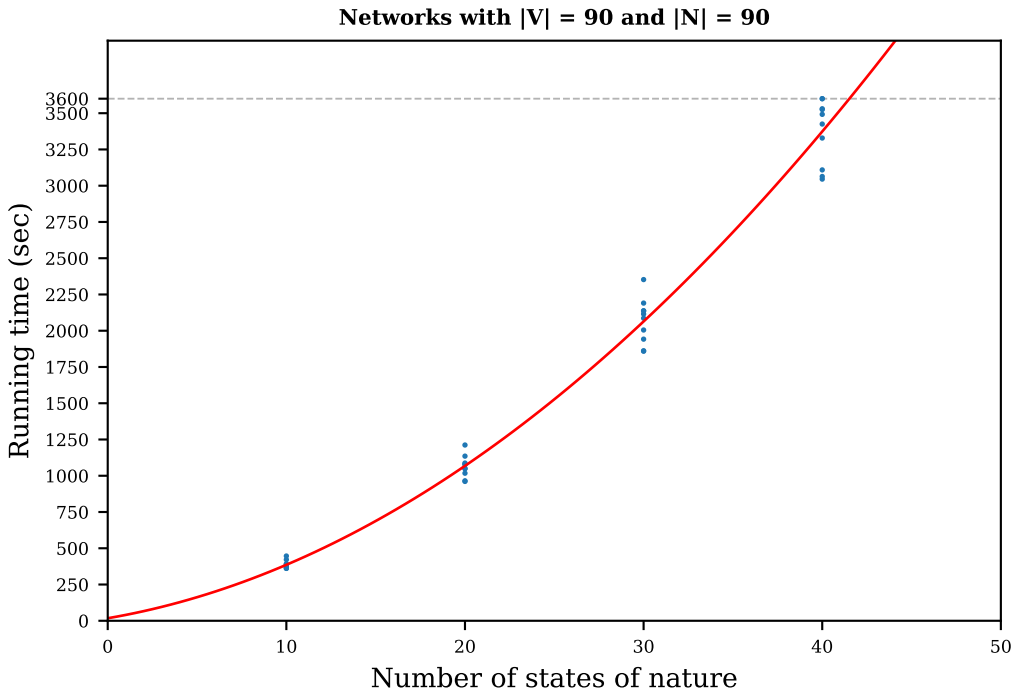


Figure 6.19. *Regression for game instances with* |V| = 90, |N| = 90 *and varying the number of states of nature.*

Chapter 7

Conclusions and Open Questions

7.1 Conclusions

In the present work, we study the problem of computing an optimal signaling scheme in a Bayesian Network Congestion Game (BNCG). We develop our analyses under the crucial assumption of *symmetric* BNCG, which assumes that all the players involved in the game shared the same source and target pair in the network. We consider also the *affine* property in the expression of the edge costs in the network. Moreover, we focused on the notion of *ex ante* persuasive signaling scheme, which requires that receivers are encouraged to follow the sender's recommendations by only observing the signaling scheme. In this setting, the problem of computing an optimal signaling scheme has been proved by Castiglioni, Celli, Marchesi and Gatti [4] to be computable in polynomial time, by exploiting the Ellipsoid method and designing a suitable separation oracle. Our aim is to investigate each aspect of the algorithm designed in order to solve the problem, proposing an implementation for each step. The crucial point is that the ellipsoid algorithm is not usable in practice. Therefore, the question is if, in practice, it is still possible to have an efficient algorithm. We implemented a solver by using Python, a programming language that allowed us to interact with the many API available to solve the problem considered. Specifically, we decide to solve the Linear Problems with Gurobi optimization solver, interfaced with our code through the Python module `gurobipy`. Computational experiments have been executed on networks of various sizes, considering different values for the parameters of number of nodes, number of players and state of nature in the game instances.

7.2 Open Questions

In the present work, we discussed and study the problem of computing an optimal *ex ante* persuasive signaling scheme in Bayesian Network Congestion Games, relying on the properties of *symmetry* and *affine costs*. This has led us to obtain and discuss in this work computational results by applying the algorithm proposed to solve the problem.

Our contributions could be extended relaxing or modifying some assumptions, in particular it would be interesting to study the possible uncertainty of the sender about receivers' payoff. In the future another interesting direction would be the design of practical algorithms for real-world network signaling problems, by considering a real setting and all related problems. Interesting new directions also include the study of settings in which the sender is uncertain on the type of the receivers resorting to online learning tools, such as [30] and [31].

Bibliography

[1] Walter, D. 2016. Computational Complexity Theory. In *The Stanford Encyclopedia of Philosophy*. https://plato.stanford.edu/archives/win2016/entries/computational-complexity/.

[2] Gatti, N. 2017. Theory Lecture 16: Potential games. In *Lecture notes. Economics and Computation.*

[3] Bertrand, N.; Markey, N.; Sadhukhan, S.; and Sankur, O. 2009. Dynamic network congestion games. *arXiv preprint arXiv:2009.13632.*

[4] Castiglioni, M.; Celli, A.; Marchesi, A.; and Gatti, N. 2020. Signaling in Bayesian Network Congestion Games: the Subtle Power of Symmetry. *arXiv preprint arXiv:abs/2002.05190.*

[5] Celli, A.; Coniglio, S.; and Gatti, N. 2020. Private Bayesian Persuasion with Sequential Games. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 1886-1893.

[6] Kamenica, E.; and Gentzkow, M. 2011. Bayesian persuasion. *American Economic Review*, 101(6):2590-2615.

[7] Galli, L. 2014. Algorithms for Integer Programming. *Computers Operations Research*, 1:1-13.

[8] Garey, M.; and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness.* WH Freeman and Company.

[9] Gurobi optimization. 2021. http://www.gurobi.com.

[10] gurobipy, the Gurobi Python Interface. 2021 https://www.gurobi.com/documentation/9.1/quickstart_mac/cs_grbpy_the_gurobi_python.html.

[11] Hogan, S. 2020. A gentle introduction to computational complexity theory, and a little bit more. *Available at* https://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/Hogan.pdf.

[12] Arieli, I.; and Babichenko, Y. 2016. Private Bayesian persuasion. *Available at SSRN 2721307.*

[13] Khachiyan, L. G. 1979. A Polynomial Algorithm in Linear Programming. In *Soviet Math. Dokl*, pages 191-194.

[14] Nemirovskii, A. S.; Yudin, D. B. 1977. Optimization methods adaptive t «significant» dimension of the problem. *Avtomat. i Telemekh.* no 4, 75–87; *Autom. Remote Control*, 38:4 (1977), 513–524.

[15] NetworkX: Network Analysis in Python. 2020. `https://networkx.org/`.

[16] Rosenthal, R.W. 1973. A class of games possessing pure-strategy Nash equilibria. In *International Journal of Game Theory*, pages 65-67.

[17] Osborne, M.; and Rubinstein, A. 1994. *A Course in Game Theory.* The MIT Press.

[18] Rebennack S. 2008. Ellipsoid Method. In *Floudas C., Pardalos P. (eds) Encyclopedia of Optimization.* Springer, pages 890–899.

[19] Dughmi S.; and Haifeng Xu. 2016. Algorithmic Bayesian Persuasion. In *Proceedings of the Fortyeighth Annual ACM Symposium on Theory of Computing*, STOC'16, pages 412-425.

[20] Dughmi, S.; 2017. Algorithmic information structure design: a survey. *ACM SIGecom Exchanges*, 15:2-24.

[21] Dughmi, S.; and Haifeng Xu. 2017. Algorithmic persuasion with no externalities. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 351-368.

[22] Vasserman, S.; Feldman, M; and Hassidim, A. 2015. Implementing the wisdom of waze. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence,* pages 660-666.

[23] Wang, Y. 2015. Bayesian persuasion with multiple receivers. *Available at SSRN 2625399.*

[24] Lübbecke, M. 2011. Column Generation. In *Cochran JJ, Cox LA, Keskinocak P, Kharoufeh JP, Smith JC (eds) Wiley encyclopedia of operations research and management science. American Cancer Society, Atlanta.* `https://doi.org/10.1002/9780470400531.eorms01580`.

[25] Bhuiyan, B. 2018. An Overview of Game Theory and Some Applications. In *Philosophy and Progress*. 59(1-2), 111-128.

[26] Kakade, S.; Kalai, A. T.; and Ligett, L. 2007. Playing games with approximation algorithms. In *Proceedings of the 39th annual ACM symposium on Theory of Computing* (STOC'07), pages 546–555.

[27] Castiglioni, M.; Celli, A.; Gatti, N. Persuading Voters: It's Easy to Whisper, It's Hard to Speak Loud. AAAI 2020: 1870-1877.

[28] Castiglioni, M.; Gatti, N. Persuading Voters in District-Based Elections. AAAI 2021.

[29] Castiglioni, M.; Celli, A.; Gatti, N. Public Bayesian Persuasion: Being Almost Optimal and Almost Persuasive. CoRR abs/2002.05156 (2020)

[30] Castiglioni, M.; Celli, A.; Marchesi, A.; Gatti, N. Online Bayesian Persuasion. NeurIPS 2020

[31] You Zu, Krishnamurthy Iyer, Haifeng Xu: Learning to Persuade on the Fly: Robustness Against Ignorance. CoRR abs/2102.10156 (2021)