UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FÉLIX CARVALHO RODRIGUES

Smoothed Analysis in Nash Equilibria and the Price of Anarchy

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof[a]. Dr[a]. Luciana Salete Buriol
Advisor

Prof. Dr. Marcus Ritt
Coadvisor

Porto Alegre, December 2011

CIP – CATALOGING-IN-PUBLICATION

Rodrigues, Félix Carvalho

Smoothed Analysis in Nash Equilibria and the Price of Anarchy / Félix Carvalho Rodrigues. – Porto Alegre: PPGC da UFRGS, 2011.

87 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2011. Advisor: Luciana Salete Buriol; Coadvisor: Marcus Ritt.

1. Algorithmic game theory. 2. Smoothed analysis. 3. Lemke-Howson algorithm. 4. Bimatrix games. 5. Frank-Wolfe algorithm. 6. Network games. 7. Traffic assignment problem. 8. Price of anarchy. I. Buriol, Luciana Salete. II. Ritt, Marcus. III. Título.

“A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.”

— DOUGLAS ADAMS

(THE HITCHHIKER’S GUIDE TO THE GALAXY)

ACKNOWLEDGMENTS

My most profound gratitude goes to Prof. Guido Schäfer, who has made this thesis possible and whom without it would not be possible for me to go into the game theory field and leave with a sane mind. For all his ideas, corrections, patience and specially guidance, which I carry not only for this thesis but for my life.

I thank my advisors, Prof. Luciana Buriol and Prof. Marcus Ritt, both which had the patience and willingness to explore a different field and provided invaluable support through the whole time I was writing this thesis.

I thank everybody at CWI (Centrum Wiskunde & Informatica) in Amsterdam, specially Bart de Keijzer, Tony Huynh and again Guido Schäfer for making me feel at home while so far away. I am specially grateful to Bart and his family, for letting me stay in their home and offering such warm hospitality the whole period I was there.

I would also like to thank the "European South American Network for Combinatorial Optimization under Uncertainty" project, programme acronym/reference FP7-People/ 247574, for the financial support while I was at Amsterdam, which made this thesis possible.

I thank Kao Félix and Letícia Nunes for the friendship and the needed moments of normal life while doing this Master's program.

Finally, I would like to give a special thank you to my father Osvaldir, my mother Helena and my sister Priscila. Without their support I would never be allowed to spend so much time on the academic world and on this thesis, with so little worry on other aspects of life.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

FW	Frank-Wolfe algorithm
LH	Lemke-Howson algorithm
NE	Nash Equilibrium
PoA	Price of Anarchy
PoS	Price of Stability
TAP	Traffic Assignment Problem
SPoA	Smoothed Price of Anarchy

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This thesis analyzes problems in game theory with respect to perturbation. It uses smoothed analysis to accomplish such task and focuses on two kind of games, bimatrix games and the traffic assignment problem.

The Lemke-Howson algorithm is a widely used algorithm to compute a Nash equilibrium of a bimatrix game. This problem is PPAD-complete (Polynomial Parity Arguments on Directed graphs), and there exists an instance which takes exponential time (with any starting pivot.) It has been proven that even with a smoothed analysis it is still exponential. However, no experimental study has been done to verify and evaluate in practice how the algorithm behaves in such cases. This thesis shows in detail how the current known worst-case instances are generated and shows that the performance of the algorithm on these instances, when perturbed, differs from the expected behavior proven in theory.

The Traffic Assignment Problem models a situation in a road network where users want to travel from an origin to a destination. It can be modeled as a game using game theory, with a Nash equilibrium happening when users behave selfishly and an optimal social welfare being the best possible flow from a global perspective.

We provide a new measure, which we call the Smoothed Price of Anarchy, based on the smoothed analysis of algorithms in order to analyze the effects of perturbation on the Price of Anarchy. Using this measure, we analyze the effects that perturbation has on the Price of Anarchy for real and theoretical instances for the Traffic Assignment Problem. We demonstrate that the Smoothed Price of Anarchy remains in the same order as the original Price of Anarchy for polynomial latency functions. Finally, we study benchmark instances in relation to perturbation.

Keywords: Algorithmic game theory, smoothed analysis, Lemke-Howson algorithm, bimatrix games, Frank-Wolfe algorithm, network games, traffic assignment problem, price of anarchy.

Análise suavisada em equilíbrios Nash e no Preço da Anarquia

RESUMO

São analisados nesta dissertação problemas em teoria dos jogos, com enfoque no efeito que perturbações acarretam em jogos. A análise suavizada (*smoothed analysis*) é utilizada para tal análise, e dois tipos de jogos são o foco principal desta dissertação, jogos bimatrizes e o problema de atribuição de tráfego (*Traffic Assignment Problem*.)

O algoritmo de Lemke-Howson é um método utilizado amplamente para computar um equilíbrio Nash de jogos bimatrizes. Esse problema é PPAD-completo (*Polynomial Parity Arguments on Directed graphs*), e existem instâncias em que um tempo exponencial é necessário para terminar o algoritmo. Mesmo utilizando análise suavizada, esse problema permanece exponencial. Entretanto, nenhum estudo experimental foi realizado para demonstrar na prática como o algoritmo se comporta em casos com perturbação. Esta dissertação demonstra como as instâncias de pior caso conhecidas atualmente podem ser geradas e mostra que a performance do algoritmo nestas instâncias, quando perturbações são aplicadas, difere do comportamento esperado provado pela teoria.

O Problema de Atribuição de Tráfego modela situações em uma rede viária onde usuários necessitam viajar de um nodo origem a um nodo destino. Esse problema pode ser modelado como um jogo, usando teoria dos jogos, onde um equilíbrio Nash acontece quando os usuários se comportam de forma egoísta. O custo total ótimo corresponde ao melhor fluxo de um ponto de vista global.

Nesta dissertação, uma nova medida de perturbação é apresentada, o Preço da Anarquia Suavizado (*Smoothed Price of Anarchy*), baseada na análise suavizada de algoritmos, com o fim de analisar os efeitos da perturbação no Preço da Anarquia. Usando esta medida, são estudados os efeitos que perturbações têm no Preço da Anarquia para instâncias reais e teóricas para o Problema de Atribuição de Tráfego. É demonstrado que o Preço da Anarquia Suavizado se mantém na mesma ordem do Preço da Anarquia sem perturbações para funções de latência polinomiais. Finalmente, são estudadas instâncias de *benchmark* em relação à perturbação.

Palavras-chave: teoria dos jogos, análise suavisada, algoritmo de Lemke-Howson, jogos bimatrizes, algoritmo de Frank-Wolfe, problema da atribuição de tráfego, preço da anarquia.

1 INTRODUCTION

Game theory is a field of applied mathematics that deals with decision making problems for more than one person. It assumes that a person making his decision behaves in a noncooperative way while having knowledge of the decision of other persons participating in the game. In game theory, a game consists of players that choose various strategies with different payoffs depending on the strategies of the other players. Players always choose a strategy that maximizes their payoff, or equivalently minimizes their costs.

A Nash equilibrium is reached when all players select a strategy such that no player can increase its payoff by changing its strategy. A Nash equilibrium is stable in the sense that when equilibrium happens, the game is in a locked state, with no more changes in strategies. There can be multiple Nash equilibria in a game, with different sum of payoffs.

There is a large number of problems in computer science as well as in other sciences where game theory can be used as a tool for mechanism design or for analysis of a number of processes. For example, game theory can be used in theoretical biology to analyze the behavior of populations, using games such as the hawk-dove game.

In the hawk-dove game, also called the game of chicken, two animals want a single resource, be it food, mating or group dominance, and can play one of two strategies: it can fight the other animal, the "hawk" strategy, or it can bluff with a threat, the "dove" strategy. If both players bluff, then neither gets the resource. If one fights and the other bluffs, than the fighter gets the reward, while the bluffer gets injured. If both decide to fight, then both die or get badly injured. This game can be seen in Figure 1.1 in its extensive format, where each possible move is shown.

There are three Nash equilibria in this game. One of the players plays hawk and the other dove, the opposite, or each player chooses between both strategies with a certain probability. In the latter case, the equilibrium is called mixed. The equilibria found in this game can give insight as to how animal population behaves, especially in relation to territory dominance behavior.

The same game can be used to model brinkmanship, both in internal politics or between nations. Taking the debt ceiling debate in the USA as a recent example, the parties have the option of compromising or not on a deal to extend the debt ceiling with a budget accompanying it. If neither party compromises in a deal, the consequences for both would be disastrous. If one gives in the other can push the deal to their ideals and win most of its goals and if both compromise neither gets the totality of its goals.

These examples show the wide spectrum of possible applications of game theory, in particular in predicting behavior of players.

There is also a notion of the best possible social welfare for a game. Social welfare can be defined as the total sum of payoffs of a given game state. This optimal social welfare notion can also be viewed as if all players cooperated selflessly to maximize the

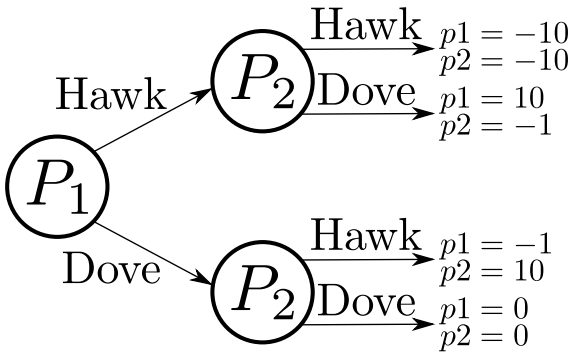

Figure 1.1: A Hawk-Dove Game, where P_1 is one player and P_2 is the other. $p1$ and $p2$ are the payoffs of each chosen strategy for each player.

total payoff. Note that a Nash equilibrium is not necessarily the best possible outcome for maximizing the social welfare. In fact, it can be quite far from it depending on the game.

The most obvious example of a large difference between a Nash equilibrium and the optimal social welfare is the tragedy of commons , usually known in game theory as the CC-PP game, short for commonize costs - privatize profits game.

In the tragedy of commons, we assume that a number of shepherds share a common grassland, i.e. the pasture is open to all herdsmen. Each shepherd tries to maximize its gain by keeping as many sheep as possible on the common grassland. Adding a sheep to its flock means he can sell it later with a certain profit, however the costs of raising the animal is shared by all herdsmen that use the grassland, diluting it among them. Since the costs of the added sheep are a fraction of the profits from it, each herdsmen will keep adding sheep to its flock until the environment collapses, hurting all shepherds.

Using the hawk-dove game previously described as the starting game, what would happen if the situation changes as to give one player an advantage on the fighting strategy over the other? Suppose that two lions are in a dispute for control of a pride. One of the lions steps on a hunting trap and injures one of its legs. If both choose to fight, the healthy lion will win and get control of the pride every time, even if it gets injured. The healthy lion now has a dominant strategy of always fighting, as long as the payoff of the fight-fight strategy remains larger than a bluff-fight strategy, i.e. the value of controlling the pride even when sustaining injury is greater than the minor injuries he would receive if it bluffed. The equilibrium is now unique, with the healthy lion fighting and the injured one bluffing.

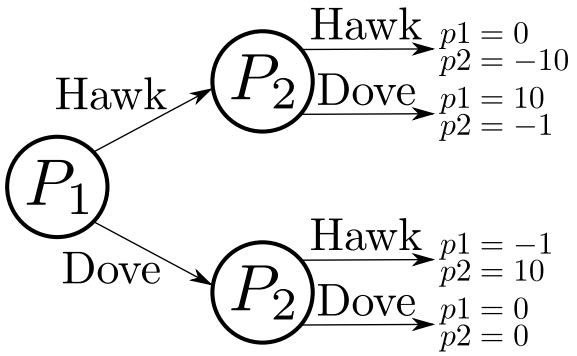

Figure 1.2: A perturbation of the Hawk-Dove Game in Figure 1.1, with payoff 0 for P_1 when the hawk-hawk strategy is chosen.

Games of this kind are called bimatrix games. These games have two players, with strategies usually explicitly given for each player. To find an equilibrium for this game, several numerical methods can be used (CEPPI et al., 2010; SAVANI, 2006). When modeled after real world systems, these games may present small perturbation in its input, which may alter not only the precision of the equilibrium, but how the worst case instances would behave if they suffered these inherent perturbations.

These games are often modeled after real world problems. However, these problems are often subject of perturbations, with the payoffs changing depending on other external variables. These perturbations may change the equilibria that existed in the game. A game that previously had a simple equilibrium to spot may now with perturbations have multiple or mixed equilibria or the opposite.

Another frequently used application of game theory is in analysis of network models. How the players are viewed depends on the specific network problem, but on most problems of this kind they are trying to get from a source to a destination in the network, e.g., they can be packets traveling on a computer network, cars on a traffic network, train companies sharing a railway network, etc. The focus of these problems when using game theory is on finding the equilibrium of the users, in contrast to traditional analysis where the users are not free agents, and finding the system optimal of a network is the goal.

Finding an equilibrium in network games can be computationally hard, and perturbation may affect the efficiency of the algorithms designed to solve these problems. Perturbation in these kinds of games are common, and it can derive from multiple situations, such as cable interferences, bad weather, road accidents and various others. In order to analyze the effect that these perturbations have on games, particularly on the complexity of finding a Nash equilibrium, smoothed analysis can be used.

Smoothed analysis is an alternative to worst-case analysis, as well as to the average-case analysis. In this kind of analysis, the input instances are perturbed by a certain magnitude, and the smoothed complexity takes both the number of instructions as well as the magnitude of perturbation into account. In other words, it measures the worst case complexity of an algorithm assuming that its input is subject to small perturbations.

This idea can also be used in similar fashion to analyze other information, such as the Nash equilibria themselves or the distance from the equilibria to the optimal social welfare.

This thesis analyzes how perturbation alters games using smoothed analysis for two different games: for bimatrix games and for the traffic assignment problem, also known as the selfish routing game.

1.1 Contributions

For bimatrix games, this thesis uses an exact implementation of the Lemke-Howson method for finding a Nash Equilibrium, with rational strategy values. An experimental analysis is done for known worst case instances for bimatrix games, using the concepts of smoothed analysis to slightly perturb the instance.

The concept of a Smoothed Price of Anarchy is defined, based on the smoothed analysis of algorithms. Furthermore, this concept is then applied for the traffic assignment problem.

Lower bounds on the Smoothed Price of Anarchy for both linear and polynomial latency functions are proven, and an experimental analysis of perturbation is done for large benchmark instances currently used for academic purposes.

1.2 Structure of this Thesis

The remainder of this thesis is organized as follows:

- Chapter 2 overviews the basics of game theory. Furthermore, it provides a short explanation of the PPAD complexity class and introduces two important class of games: network games and bimatrix games. It presents a short explanation of the Lemke-Howson algorithm used to find a Nash equilibrium on bimatrix games. Finally, it defines Price of Anarchy (PoA) and Price of Stability (PoS).

- Chapter 3 presents the Traffic Assignment Problem. It provides a description of the Frank-Wolfe algorithm used to solve this problem, and models the problem using game theory. The chapter simplifies the Price of Anarchy definition present in Chapter 2 to the specific case of the Traffic Assignment Problem. At last, it presents two worst case analysis for Pigou instances, which are proved to be the worst possible instances for the PoA in TAP, for both linear and polynomial latencies.

- Chapter 4 introduces smoothed analysis of algorithms. It provides a definition of several perturbation models that can be used in this type of analysis. Furthermore, it defines a novel way to analyze perturbation on the Price of Anarchy, which we name the Smoothed Price of Anarchy.

- Chapter 5 studies the effects of perturbation on bimatrix games. It describes how to generate the known class of games for which the Lemke-Howson algorithm has an exponential number of steps, and analyzes these instances with relation to perturbation. Several experiments are performed on these instances and it is shown that they drop to a polynomial number of steps even for polynomially small perturbations.

- Chapter 6 defines the Smoothed Price of Anarchy in relation to the Traffic Assignment Problem. Furthermore, it provides new lower bounds on the SPoA which is in the same order of magnitude for polynomial latencies as the worst case for the PoA. Finally, experiments are performed on benchmark instances, measuring the effect of perturbation on these instances.

- Chapter 7 summarizes this thesis and lists some possible avenues for future exploration. It provides a possible path to find hard bimatrix instances for perturbation, and lists possible applications of the Smoothed Price of Anarchy to different games.

2 GAME THEORY

Game theory is a branch of mathematics with a vast number of applications in areas such as biology and economics. There are also a number of applications in computer science, for example in network routing and optimization. In general, the goal of game theory is to model situations where the benefit of the choices of an individual depends on the choices of others.

A strategic game can be defined as a set of players, strategies and a payoff associated to each strategy combination.

Definition 2.1. *Let there be n players. Each player i has a set $S_i = \{s_1, ..., s_{m_i}\}$ of* pure *strategies, where $m_i \geq 2$. A vector $x = (x_1, x_2, ..., x_n) \in S$ is a pure strategy profile, where x_i is a pure strategy for player i. The set S with all such strategies is defined as:*

$$S = S_1 \times S_2 \times ... \times S_n.$$

For each player i there is a payoff function $f_i : S \to \mathbb{R}$*, which defines the payoff $f_i(x)$ of player i for a given* strategy profile $x \in S$.

Then, a strategic game G is defined as the tuple consisting of the strategy set S and the payoff function f for n players, i.e.

$$G = (S, f).$$

We use the prisoners' dilemma, a well-known problem in this area, to illustrate these definitions. The game is stated as follows. Two suspects of a crime are held by the police. The police does not have enough evidence to incriminate both, and each prisoner has the choice of either confess or remain silent. If both remain silent, the authorities will not be able to prove charges against them and each will face a prison time of two years for previous felonies. If only one of them confesses, he will face one year in prison and his confession will be used to condemn the other to five years in prison. If they both confess, each will have a sentence of four years.

Table 2.1: Prisoners' dilemma.

P1 \ P2	**Confess**	**Silent**
Confess	$-4, -4$	$-1, -5$
Silent	$-5, -1$	$-2, -2$

Table 2.1 presents the payoff matrix of the prisoners' dilemma game. In this matrix, the value in each cell i, j corresponds to the payoffs of the strategy in which prisoner P1

chooses strategy i and prisoner P2 chooses strategy j. This way of describing a game is known as *normal form*. For contrast, we can see the same game in *extensive form* in Figure 2.1.

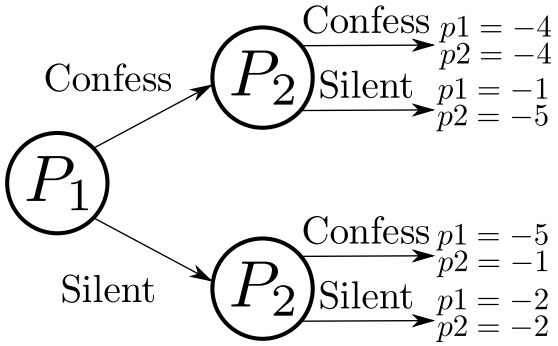

Figure 2.1: The prisoner's dilemma game in extensive form

2.1 Nash Equilibria

A Nash equilibrium is a strategic profile in which each player, taking into consideration other players strategies, does not have any incentive to unilaterally change its strategy.

Formally, let x_{-i} be the vector of strategies played by all players except player i:

$$x_{-i} = (x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n) \in S_{-i}$$

Definition 2.2 (Best Response w.r.t. x_{-i})**.** *A* best response *strategy for a player i is a $x \in S_i$ such that, given a vector of strategies x_{-i} of the other players, satisfies*

$$\max_{\forall x_i \in S_i} f_i(x_i, x_{-i}).$$

Therefore, a strategy profile $x^* = (x_1^*, x_2^*, ..., x_n^*) \in S$ is a Nash equilibrium if, for each player i, x_i^* is a best response to x_{-i}^*. In other words:

Definition 2.3 (Nash Equilibrium)**.** *A game (S, f) is at a Nash equilibrium $x^* = (x_1^*, ..., x_n^*)$ if:*

$$\forall i \leq n, \forall x_i \in S_i,\ x_i \neq x_i^* : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*).$$

In the prisoners' dilemma, clearly the only possible Nash equilibrium is the case where both confess. If a prisoner remains silent, the other would have an advantage when he confesses, reducing his prison time to one year.

In the example, if both prisoners remained in silence the best social welfare possible would be obtained. This exemplifies that a Nash equilibrium does not always correspond to a strategy solution with the optimal social welfare.

Another important fact is that a game can have multiple Nash equilibria, with different payoffs for each equilibrium. A Nash equilibrium is stable, that is, once an equilibrium is reached, no unilateral strategy change of a player is profitable for him.

A Nash equilibrium can either be a pure or mixed equilibrium. A *pure equilibrium* happens when each player uses only one pure strategy at the equilibrium. A game has a *mixed equilibrium* if it is allowed to play a set of strategies with a certain probability distribution at the equilibrium. The *support* of a mixed strategy is the set of all pure strategies which have a nonzero probability associated with them.

Definition 2.4 (Mixed Strategy)**.** *Let $\Delta(X)$ denote the set of probability distributions over X. Given a game $G = (S, f)$ with n players, a* mixed *strategy of player i is a probability distribution $\rho_i \in \Delta(S_i)$.*

Definition 2.5 (Mixed Nash Equilibrium)**.** *Given $S_\Delta = \Delta(S_1) \times ... \times \Delta(S_n)$. Let $g_i : S_\Delta \to \mathbb{R}$ denote the expected payoff of player i, defined as*

$$g_i(\rho_1, ..., \rho_n) = \sum_{s \in S} \rho_1(s_1) \times ... \times \rho_n(s_n) f_i(s) \ .$$

A game (S, f) is at a mixed Nash equilibrium *$y^* = (y_1^*, ..., y_n^*)$ if:*

$$\forall i \leq n, \forall y_i \in \Delta(S_i),\ y_i \neq y_i^* : g_i(y_i^*, y_{-i}^*) \geq g_i(y_i, y_{-i}^*).$$

An example of a game where a mixed equilibrium can be reached is a simplified penalty kick game. In a soccer match, the penalty kicker can shoot the ball either to the right or to the left, and the goalkeeper can defend right or left. If the kicker shoots to the side, then the goalkeeper defends and the soccer game continues, with both players having zero payoff. If the kicker shoots to a different side than the goalkeeper dives, then the kicker scores, with the kicker getting a payoff of 1 while the goalkeeper gets -1.

Table 2.2: Penalty game, player *K* is the kicker and *G* is the goalkeeper.

K \ G	**Right**	**Left**
Right	$0, 0$	$1, -1$
Left	$1, -1$	$0, 0$

In this game, an equilibrium is met when the kicker plays right with a 50% chance (and left with 50% chance too) and the goalkeeper also defends right with a 50% chance (and to the left). Here, the "right" and "left" are from the perspective of the kicker. This is also an example of a *zero-sum* game, where the amount lost by one player is equal to the gain of the adversary.

Finally, the problem of finding a Nash equilibrium can be defined as:

Definition 2.6 (Problem NASH)**.** *Given a game (S, f), find a mixed Nash equilibrium according to Definition 2.5.*

There are other problems related to Nash equilibria, such as listing all possible equilibria, or finding a pure Nash equilibrium. In this thesis we are focusing in the problem described in Definition 2.6, with different kinds of instances.

2.1.1 PPAD Complexity Class

Nash proved (NASH, 1951) that in any finite game there exists at least one mixed Nash equilibrium. The method used to prove this fact is nonconstructive, it uses the Brouwer's fixed point theorem. This theorem says that, given a closed ball D in $\mathbb{R}^n$, and a continuous function $f : D \to D$, there is a fixed point $x \in D$, i.e., a point such that $f(x) = x$, but without providing a polynomial time method to find x.

Therefore, Nash's proof is not useful to develop an efficient algorithm to find an equilibrium. In fact, it can be used to prove that NASH is as hard as the problem of finding a Brouwer fix point. Daskalakis et al. (DASKALAKIS; GOLDBERG; PAPADIMITRIOU, 2006) proved that NASH is PPAD-complete for three or more players and Chen et al. (CHEN; DENG, 2006) extended this proof to two player games.

The PPAD complexity class stands for *Polynomial Parity Argument on Directed Graphs*, and is a subset of the TFNP (Total Function Nondeterministic Polynomial) complexity class, which is the set of the function problems in FNP that are proven to be total.

Definition 2.7 (TFNP)**.** *Given a binary relation $P(a, b)$, it belongs to TFNP if and only if there exists a deterministic polynomial time algorithm which can state if $P(a, b)$ holds for a given a and b, and $P(a, b)$ is total, i.e. for every a, there exists a b such that $P(a, b)$ is true.*

The PPAD class is defined by (PAPADIMITRIOU, 1994) as the class of all problems reducible in polynomial time to the problem called *End-Of-Line*. In addition, a problem X is PPAD-Complete if X is in PPAD and End-Of-Line is reducible in polynomial time to X.

Definition 2.8 (End-Of-Line)**.** *Let $f(v)$ be a function that defines a directed graph G with every vertex having at most one successor and one predecessor, with no isolated vertex, where $f(v)$ returns both successor and predecessor of vertex v, such that $f(v)$ can be computed in polynomial-time on the size of v.*

Given $f(v)$ and a vertex $s \in G$ with no predecessor, find a vertex $v' \neq s$ such that v' has no successor, or no predecessor.

Note that while $f(v)$ needs to be polynomial in terms of the size of v, the graph G that it defines does not. In fact, the graph G can have an exponential size. The graph G will always have a sink node (a node with no successor), since the vertex s is guaranteed to be a source vertex with no predecessor. Therefore we know that an answer exists. However, finding it depends on the size of the graph G, which can be exponentially large. This kind of graph is exemplified in Figure 2.2.

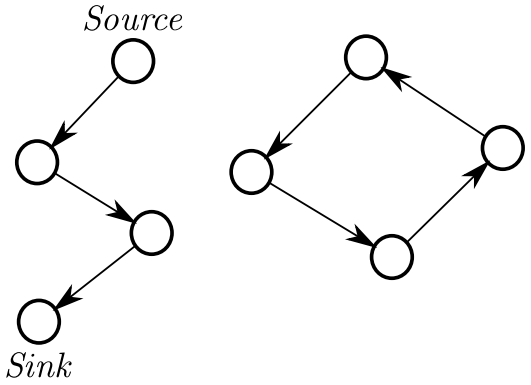

Figure 2.2: Example of a graph in the End-Of-Line problem.

2.1.2 Network Games

Game theory can be used to analyze several traditional problems in networks from a different perspective. Network games model the interaction between parts of systems controlled by different parties which often have different objectives.

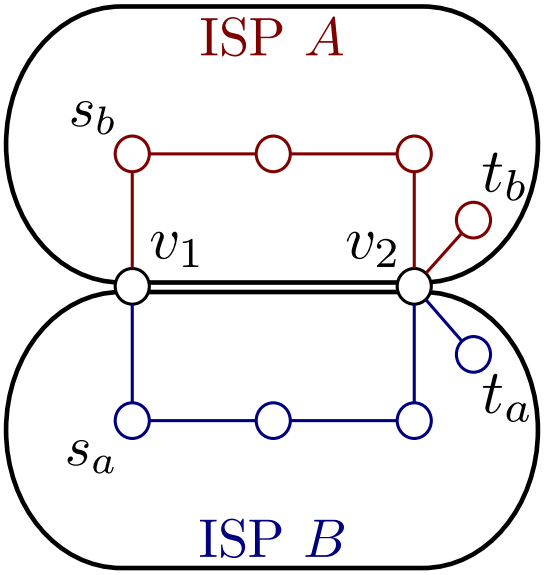

Figure 2.3: Example of a network game.

The most common approach for analyzing these network problems is to consider the users of this network as players in a non-cooperative game. For instance, the players could be packets on the Internet or cars in a traffic network, both of which act in a non-cooperative way trying to maximize their own utility.

In these games, the set of strategies for each player is usually not explicitly given and can be exponential in size, with the network being used to derive them as needed. As a simple example we can take ISPs (Internet Service Providers) that need to send traffic between each other. On the traffic routing between different ISPs, the chosen route of each provider affects the cost of the other providers traffic.

In the example in Figure 2.3, we can see two choke points v_1 and v_2. While routing in the ISP's own network costs 1 per traveled link, routing traffic through the other provider network costs 1 for the other provider. ISP A wants to send data from s_a to t_a, while ISP B wants to send data from s_b to t_b. Clearly the equilibrium is to send all traffic through vertex v_1, even though the optimal solution for the whole network would be for them to use vertex v_2. Note that this particular example can be seen as a prisoner's dilemma kind of game, where choosing v_1 is equal to confessing while choosing to use vertex v_2 being the same as remaining silent.

2.2 Bimatrix Games

In Bimatrix Games, two players have pure strategies given by matrices A and B, with m strategies for the first player and n strategies for the second player. A Nash equilibrium in this kind of game can be defined as a vector of probabilities for each player over the pure strategies given in the players' matrices.

Definition 2.9 (Two-Player NASH)**.** *Consider a game in normal form, with two players M and N, with m pure strategies for player M and n pure strategies for player M. In a given play of the game, if M plays his ith pure strategy, and N plays his jth pure strategy, the payoff to M is $a_{i,j}$ and the payoff to N is $b_{i,j}$.*

Let A and B be $m \times n$ matrices such that the (i, j)-element of A is $a_{i,j}$ and the (i, j)-element of B is $b_{i,j}$. Let x and y be mixed strategies of players M and N, respectively, represented by a column of nonnegative elements such that x_i represents the probability that player M plays his ith pure strategy, and y_j represents the probability that player N plays his jth pure strategy.

Find a mixed strategy for player M and one for N such that it is at a Nash equilibrium.

2.2.1 The Lemke-Howson Algorithm

The Lemke-Howson algorithm (LH) (LEMKE; J. T. HOWSON, 1964) is a well known algorithm to find a Nash equilibrium in a two player game. This algorithm can have an exponential number of steps to reach an equilibrium in the worst case scenario, as proved by Savani and Stengel (SAVANI; STENGEL, 2004). However, little experimental study has been performed on these worst case instances and on how small perturbations alter its performance. It resembles the simplex method to solve linear programs. The core of the algorithm is to perform iterated pivoting.

For a two player game, with a matrix A representing the payoffs of player P1 and a matrix B representing the payoffs of player P2, the Lemke-Howson algorithm operates by keeping a guess of what the supports should be, and, in each iteration, that guess is slightly modified by a pivoting operation.

Another way of seeing the operation is using a representation of the game in the form of two polytopes P and Q. In this representation, for a matrices A and B of order $m \times n$ and $n \times m$, respectively , the polytopes are defined as:

$$P = \left\{x \in \mathbb{R}^m \mid \ x \geq \mathbf{0},\ B^T x \leq \mathbf{1}\right\}, \quad Q = \left\{x \in \mathbb{R}^n \mid \ y \geq \mathbf{0},\ A^T y \leq \mathbf{1}\right\}$$

In this representation, the LH algorithm follows a path (the LH path) of vertex pairs (x, y) of $P \times Q$ starting at a point $(0, 0)$, and ending in a Nash equilibrium. It alternately follows edges of P or Q, while keeping the vertex in the other polytope fixed. This corresponds to the iterated pivoting used to algebraically execute these steps.

To perform the LH algorithm, it makes use of a structure called *tableau* which represents the inequalities in matrices A and B by introducing slack variables to generate equalities.

2.2.1.1 Example

Consider the following game shown in Table 2.3:

Table 2.3: Example Game.

A \ B	3	4	5
1	$3, 4$	$5, 2$	$6, 3$
2	$6, 2$	$1, 4$	$5, 1$

The correspondent polytopes P and Q are the ones shown in Figure 2.4:

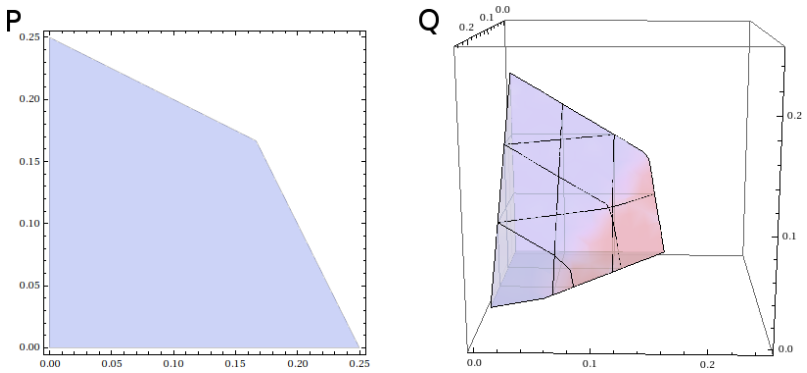

Figure 2.4: Polytopes P and Q representing the game from Table 2.3.

A tableau corresponding to the game should look like the following:

$$\begin{array}{lllll} s_1 & = 1 & & -3x_3 & -5x_4 & -6x_5 \\ s_2 & = 1 & & -6x_3 & -x_4 & -5x_5 \\ s_3 & = 1 & -4x_1 & -2x_2 & & \\ s_4 & = 1 & -2x_1 & -4x_2 & & \\ s_5 & = 1 & -3x_1 & -x_2 & & \end{array}$$

Here $s_1, ..., s_5$ are the slack variables and the $x_1, ..., x_5$ are the decision variables. Initially all slack variables are in the base, and the pivoting process begins by choosing one variable to enter the base (the pivot) and one variable to leave the base. The first pivot can be chosen arbitrarily, while the leaving variable is chosen by finding the lowest *min-ratio* value, which is the lowest value of the current base value divided by the entering variable coefficient negated.

From this point on the variable that was taken from the base will determine the next pivot to enter the base. If variable s_i left the base, then variable x_i should be the next pivot in line.

Using the example, if x_1 was to enter the base, s_3 should leave, since it has the lowest min-ratio. The tableau should look like the following:

$$\begin{array}{lllll} s_1 & = 1 & & -3x_3 & -5x_4 & -6x_5 \\ s_2 & = 1 & & -6x_3 & -x_4 & -5x_5 \\ x_1 & = 1/4 & -1/4s_3 & -1/2x_2 & & \\ s_4 & = 1/2 & 1/2s_3 & -3x_2 & & \\ s_5 & = 1/4 & 3/4s_3 & -1/2x_2 & & \end{array}$$

This process executes until the initially chosen variable or its corresponding slack variable leaves the base. In the example, the algorithm would terminate if either x_1 or s_1 leaves the base. To obtain the probability distribution after the algorithm finishes, the non-basic variables are set to zero and and these values are then normalized to sum one.

This algorithm does not take into consideration the effects of degeneration. In other words, at each iteration of the pivoting one single variable must win the min-ratio test. In general, a tie breaking rule must be applied in these degenerate cases. The lexicographic method can be used to break such ties, as in (STENGEL, 2002).

2.3 Price of Anarchy

The *Price of Anarchy (PoA)* is a measure of the inefficiency of equilibria that was introduced by Koutsoupias and Papadimitriou (KOUTSOUPIAS; PAPADIMITRIOU, 1999). It measures how well players in a game perform when they are at a Nash equilibrium, compared to an optimum outcome that could be achieved if all players cooperated.

In a game with more than one Nash equilibrium, it takes the worst equilibrium and compares it to the strategy with the best possible total sum of payoffs.

Definition 2.10 (Price of Anarchy)**.** *Let G be a strategic game with n players, with a set of strategies S_i for each player i and a cost function $c_i : S \to \mathbb{R}$, where $S = S_1 \times \cdots \times S_n$. Also let $C : S \to \mathbb{R}$ be a social cost function that maps every strategy profile $s \in S$ to some non-negative cost of the game. Given an instance $I = (G, (S_i), (c_i))$, let NE(I) be the set of strategy profiles $s \in S$ that are a Nash equilibrium for I.*

Then, the Price of Anarchy of I is defined as

$$\textit{PoA}(I) = \frac{\max_{s \in \textit{NE}(I)} C(s)}{\min_{s \in S} C(s)} \quad ,$$

and the Price of Anarchy of a class of games $\mathcal{G}$ is defined as

$$\textit{PoA}(\mathcal{G}) = \max_{I \in \mathcal{G}} \textit{PoA}(I)$$

Using the Prisoners' Dilemma seen in Figure 2.1 as an example, the social cost function would be defined as the sum of each played strategy. With that cost, we can easily see that the best social welfare for this game is when both remain silent, with a cost of -4, compared to one player staying silent while the other confesses with -6 and to both confessing with -8, which is the Nash equilibrium. The Price of Anarchy in this case is then PoA $= -8/-4 = 2$.

In the context of network games, the Price of Anarchy is one of the most important measures to analyze, both as a tool to improve such networks or to assess how inefficient a given network can be. Its importance is clear in network problems were there is no direct control over players, leading to users eventually reaching a Nash equilibrium. The equilibrium reached however is not necessarily the worst possible, so another measure relating equilibria and optimal social welfare is also relevant. If instead comparing the worst Nash equilibrium of a game with the system optimum, we use the best equilibrium, we obtain a inefficiency measure called the *Price of Stability (PoS)*.

Definition 2.11 (Price of Stability)**.** *Let G be a strategic game with n players, with a set of strategies S_i for each player i and a cost function $c_i : S \to \mathbb{R}$, where $S = S_1 \times \cdots \times S_n$. Also let $C : S \to \mathbb{R}$ be a social cost function that maps every strategy profile $s \in S$ to some non-negative cost of the game. Given an instance $I = (G, (S_i), (c_i))$, let NE(I) be the set of strategy profiles $s \in S$ that are a Nash equilibrium for I.*

Then, the Price of Stability of I is defined as

$$\textit{PoS}(I) = \frac{\min_{s \in \textit{NE}(I)} C(s)}{\min_{s \in S} C(s)}$$

In this thesis, the Price of Anarchy will be the only inefficiency measure used, since for the Traffic Assignment Problem analyzed in details in Chapter 3, all equilibria have the same cost, making both terms interchangeable for this specific problem.

3 THE TRAFFIC ASSIGNMENT PROBLEM

In the *Traffic Assignment Problem (TAP)*, each user travels in a road network from a source to a destination with an assigned traversal cost to each edge of the network. How the users behave leads to two optimization problems. In the first problem users obey Wardrop's first principle of equilibrium (WARDROP, 1952), which states that each user seeks to minimize his travel time non-cooperatively. For the second one, users behave following Wardrop's second principle of equilibrium, where they cooperatively choose each route in order to guarantee that the average travel time of all users is minimum.

The road network is represented by a directed multigraph $G = (V, E)$, where V is the set of vertices and E is the set of arcs. The users are modeled by a set of commodities K with each commodity $i \in K$ having an associated vertex pair $(s_i, t_i) \in V \times V$. Users that have the same origin-destination pair (s_i, t_i) are said to belong to the same commodity i. For each commodity $i \in K$ we are given a demand d_i which specifies the total flow (corresponding to the users of commodity i) that has to be sent from s_i to t_i.

The set of paths from s_i to t_i is denoted as $\mathcal{P}_i$. Let $\mathcal{P} = \cup_{i \in K} \mathcal{P}_i$. A flow f specifies for each path $P \in \mathcal{P}$ a non-negative flow value that is sent along P, i.e., f is a function $f : \mathcal{P} \to \mathbb{R}^+$. The flow on arc $e \in E$ is defined as $f_e = \sum_{P:e \in P} f_P$, where $P \in \mathcal{P}$. A flow f is *feasible* if it satisfies the demand for every commodity, i.e., $\sum_{P \in \mathcal{P}_i} f_P = d_i$ for every $i \in K$.

For each arc $e \in E$ we are given a latency function $l_e : \mathbb{R}^+ \to \mathbb{R}^+$ which maps the flow f_e of an edge e to the traversal time $l_e(f_e)$. The latency of a path $P \in \mathcal{P}$ is defined as the sum of the edge latencies in the path, i.e., $l_P = \sum_{e \in P} l_e(f_e)$. Subsequently, we use (G, d, l) to refer to an instance of the Traffic Assignment Problem.

We assume that all latency functions are nonnegative, differentiable and nondecreasing. For real-world instances, the most common type of latency functions originates form the U.S. Bureau of Public Roads (Bureau of Public Roads, 1964), which can be expressed as:

$$l_e(f_e) = t_e \left(1 + \alpha \left(\frac{f_e}{c_e} \right)^{\beta} \right). \tag{3.1}$$

Here t_e is the free-flow travel time of edge e, i.e., the time it takes to travel through road e if there is no congestion. The constant c_e stands for the capacity of edge e and α and β are tuning parameters, usually set to 0.15 and 4, respectively (all variables are greater than zero).

In order to evaluate the total travel time of the network, we define a cost function $c(f) = \sum_{e \in E} l_e(f_e) f_e$. The *system optimum* refers to a feasible flow that minimizes this cost function.

Definition 3.1 (System Optimum)**.** *A system optimum is a flow f^* that minimizes the following program:*

$$\begin{aligned} \underset{f}{\textit{minimize}} \quad & c(f) = \sum_{e \in E} l_e(f_e) f_e \\ \textit{subject to} \quad & \sum_{P \in \mathcal{P}_i} f_P = d_i \quad \forall i \in K \\ & \sum_{P \in \mathcal{P}: e \in P} f_P = f_e \quad \forall e \in E \\ & f_P \geq 0 \quad \forall P \in \mathcal{P}. \end{aligned} \tag{3.2}$$

A feasible flow f is a *Wardrop flow* (or *user equilibrium*) if the flow of every commodity i travels along a minimum latency path available. That is, for every commodity i all flow-carrying paths have the same latency and all other paths have no smaller latency. More formally:

Definition 3.2 (Wardrop flow)**.** *A flow f is a Wardrop flow if*

$$\forall i \in K, \ \forall P_1, P_2 \in \mathcal{P}_i, \ \ f_{P_1} > 0 : \quad l_{P_1}(f) \leq l_{P_2}(f). \tag{3.3}$$

The problem of computing a Wardrop flow can be described by the following program:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \sum_{e \in E} \int_0^{f_e} l_e(x)\, dx \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}_i} f_P = d_i \qquad \forall i \in K \\ & \sum_{P \in \mathcal{P}: e \in P} f_P = f_e \qquad \forall e \in E \\ & f_P \geq 0 \qquad \forall P \in \mathcal{P}. \end{aligned} \tag{3.4}$$

Note that, while there can be multiple user equilibria, the cost $c(f)$ of a Wardrop flow f is unique (see (ROUGHGARDEN, 2003)).

An optimal flow corresponds to a Wardrop flow with respect to marginal cost functions. In order for this equivalence to hold we further need to assume that all latency functions are *standard* (ROUGHGARDEN, 2003), i.e., $x \cdot l(x)$ is convex. The *marginal cost function* of edge e is defined as $l_e^*(x) = l_e(x) + x\frac{d}{dx}(l_e(x))$. Now, a feasible flow f^* is an optimal flow for (G, d, l) if and only if it is a Wardrop flow for the instance (G, d, l^*) (see (ROUGHGARDEN, 2003) for details).

In order to have a intuition as to why this is true, consider the Karush–Kuhn–Tucker (KKT) optimality conditions (KUHN; TUCKER, 1951). Suppose we are given an optimization problem of minimizing the expression $\sum_{e \in E} h_e(f_e)$, subject to the same restrictions as in 3.2. For all $e \in E$, let the function h_e be continuously differentiable and convex. Then f is an optimal solution for this optimization problem if and only if for every $i \leq n$,

$$\forall P_1, P_2 \in \mathcal{P}, f_{P_1} > 0 : \sum_{e \in P_1} h'_e(f_e) \leq \sum_{e \in P_2} h'_e(f_e). \tag{3.5}$$

Using these conditions, we can determine functions h_e such that the KKT optimality conditions seen in Equation 3.5 reduce to the Wardrop flow conditions in Equation 3.3. Using this, together with the characterization given to Wardrop flows in Equation 3.4 and the system optimum flow in Equation 3.2, it is possible to state that a flow f is an optimal flow with respect to (G, d, l) if and only if f is a Wardrop flow with respect to (G, d, l^*), where for each $e \in E$, $l_e^*(x) = (x \cdot l_e(x))'$.

3.1 The Frank-Wolfe Algorithm

Computing a Wardrop flow as well as an optimal flow can be done by using the Frank-Wolfe Algorithm (FRANK; WOLFE, 1956). The Frank-Wolf Algorithm can solve problems where the objective function is convex. However, the constraints of the problem must be linear for the algorithm to work.

The algorithm starts by finding a feasible solution to the linear constraints of the problem. Then, in each iteration, it finds a descent direction and a distance to descend, thereby reducing the objective function value. The algorithm stops when no improvement can be made to the objective function value.

Let $z(x)$ be a convex objective function with $a \cdot x \geq b$ as linear constraints, for the program:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & z(x) \\ \text{subject to} \quad & a \cdot x \geq b \,. \end{aligned} \tag{3.6}$$

The Frank-Wolfe algorithm works as follows:

1. Set a basic feasible solution x_0 to start the algorithm with counter $k \leftarrow 0$.

2. Find a feasible direction of descent p_k.

   In order to find p_k, one must solve the problem obtained by replacing the function z with the first-order Taylor expansion around x_k, i.e., $t_k(y) = z(x_k) + \nabla z(x_k)(y - x_k)$. Since $z(x_k)$ is constant in this iteration and y is to be minimized, the linear programming problem to be solved is:

   $$\begin{aligned} \underset{y}{\text{minimize}} \quad & \nabla z(x_k) y \\ \text{subject to} \quad & a \cdot y \geq b \,. \end{aligned} \tag{3.7}$$

   Let y_k be the optimal solution for this problem. Then $p_k = y_k - x_k$, and since both x_k and y_k have the same constraints, this is a feasible direction.

3. Find a step length α_k that minimizes the new iteration point, i.e.:

   $$\underset{0 \leq \alpha_k \leq 1}{\text{minimize}} \quad z(x_k + \alpha_k p_k) \tag{3.8}$$

   Note that α_k must be in the interval $[0, 1]$ in order for the solution to remain feasible.

4. Terminate the algorithm if a stopping criteria is fulfilled, otherwise let $x_k \leftarrow x_k + \alpha_k p_k$ and $k \leftarrow k + 1$, then repeat from step 2.

   For an exact computation of $z(x^*)$, where x^* is the optimal solution, the algorithm should be stopped when $t_k(y_k) = 0$.

While the Frank-Wolfe algorithm does converge to an exact solution, its rate of convergence after a threshold that depends on the instance being computed is very slow.

There are several methods of accelerating the convergence to a degree, with several published works such as (ARRACHE; OUAFI, 2008), (FUKUSHIMA, 1984) and (FLORIAN, 1977). These improvements focus on several different attributes, such as improving the search directions or modify the line search to take longer steps. However, in all these modified Frank-Wolfe algorithms there is still a threshold where the speed of the convergence is too slow for most practical uses.

If a relative amount of error is allowed, a more practical stopping criterion can be adopted. A criterion commonly adopted is to stop the algorithm if

$$\frac{z(x_k) - t_k(y_k)}{|z_k(y_k)|} \leq \varepsilon \ , \tag{3.9}$$

where ε is the relative objective error allowed.

3.2 Traffic Assignment Problem in Game Theory

The Traffic Assignment Problem can also be defined as a game. Let the users be defined as players, with each path that a user can take being a different strategy. The payoffs that the players try to maximize can be seen as the negated latency costs of the road network. A Wardrop flow f then can be said to be at a *Nash equilibrium*, while a flow f^* at system optimum is the optimal social welfare of the game.

3.2.1 Price of Anarchy in the Traffic Assignment Problem

In the context of the Traffic Assignment Problem (TAP), the definition in 2.10 simplifies to the following:

Definition 3.3 (Price of Anarchy on the TAP)**.** *Let $I = (G, d, l)$ be an instance of TAP. The Price of Anarchy of I is*

$$\mathit{PoA}(I) = \frac{c(f)}{c(f^*)},$$

where f and f^* are a Wardrop flow and an optimal flow of I, respectively. (Recall that the cost of a Wardrop flow is unique.)

The Price of Anarchy of TAP is defined as

$$\mathit{PoA} = \max_I \mathit{PoA}(I),$$

where the maximum is taken over all possible input instances.

The Price of Anarchy depends on which types of latency functions we allow our instances to have. Roughgarden and Tardos (ROUGHGARDEN; TARDOS, 2002) proved that for linear latencies the Price of Anarchy is $\frac{4}{3}$. Furthermore, Roughgarden proved that the Price of Anarchy is independent of network topology (ROUGHGARDEN, 2003). Besides other results, these studies reveal that the Price of Anarchy for polynomial latency functions is admitted on very simple single-commodity instances consisting of two parallel arcs, also known as *Pigou instances*.

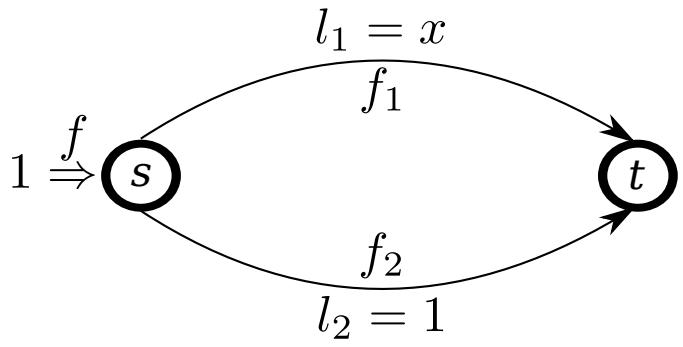

Figure 3.1: Pigou instance with linear latency functions.

3.2.1.1 Linear case

Consider the instance $I = (G, d, l)$ depicted in Figure 3.1. With one unit of flow to be sent from s to t, there are two paths that can be chosen: either take the path e_2 with a constant cost of $l_2(x) = 1$ or use e_1 with a linear latency of $l_1(x) = x$.

The user equilibrium happens when both paths have the same cost, therefore the flow f_2 must be one. The cost $c(f)$ of the flow is consequently also one. In order to compute an optimal flow, we exploit the equivalence that an optimal flow is a Wardrop flow with respect to marginal cost functions $l_1^*(x) = 2x$ and $l_2^*(x) = 1$. Again both paths must have the same cost, so $f_2^* = 1/2$ and $f_1^* = 1/2$.

The cost of the total flow f^* is thus $c(f^*) = f_1^*(1) + f_2^*(f_2^*) = \frac{1}{2}1 + \frac{1}{2}\frac{1}{2} = \frac{3}{4}$. The Price of Anarchy is accordingly

$$\text{PoA}(I) = \frac{c(f)}{c(f^*)} = \frac{4}{3}. \tag{3.10}$$

The same reasoning can be applied to the case where polynomial latency functions are allowed, as shown next.

3.2.1.2 Polynomial case

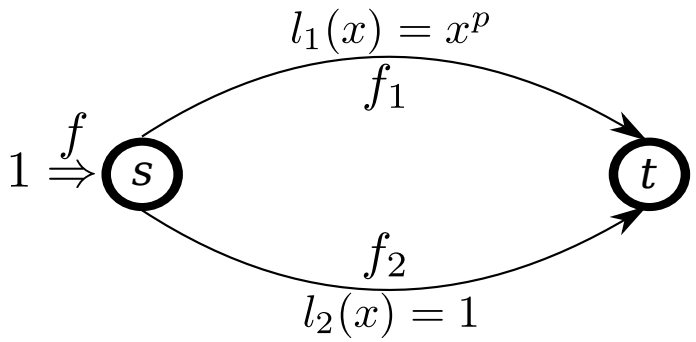

Figure 3.2: Pigou instance with polynomial latency functions.

Consider the instance $I = (G, d, l)$ depicted in Figure 3.2. There is one unit of flow that has to be sent from s to t. The respective latency functions of the upper edge e_1 and the lower edge e_2 are $l_1(x) = x^p$ and $l_2(x) = 1$.

The Wardrop flow f sends the entire flow on e_1, i.e., $f_1 = 1$ and $f_2 = 0$, and has a cost $c(f) = 1$. In order to compute an optimal flow, we again exploit the fact that a flow at the

system optimum is the same as a flow at the user equilibrium with respect to the marginal cost functions $l_1^*(x) = (p+1)x^p$ and $l_2^*(x) = 1$. Equalizing these latency functions, we obtain that $f_1^* = (p+1)^{-1/p}$ and $f_2^* = 1 - (p+1)^{-1/p}$. The cost of this flow is

$$c(f^*) = (p+1)^{-1/p}\left((p+1)^{-1/p}\right)^p + 1 - (p+1)^{-1/p} = \frac{(p+1)(p+1)^{1/p} - p}{(p+1)(p+1)^{1/p}}.$$

The Price of Anarchy of this instance I is therefore

$$\text{PoA}(I) = \frac{c(f)}{c(f^*)} = \frac{(p+1)\sqrt[p]{p+1}}{(p+1)\sqrt[p]{p+1} - p}. \tag{3.11}$$

This PoA is actually the worst possible, and is $\Theta(\frac{p}{\ln p})$. Roughgarden (ROUGHGARDEN, 2003) showed that the Price of Anarchy of multi-commodity instances with polynomial latency functions of degree at most p is at most $\text{PoA}(I)$ as stated in (3.11). It is also shown that the $\text{PoA}(I) = \frac{4}{3}$ shown in (3.10) is the worst possible ratio for multi-commodity instances with linear latency functions.

4 SMOOTHED ANALYSIS

Understanding the behavior of algorithms is critical in algorithm design. The most common approach to the analysis of algorithms is the worst-case analysis. In this kind of complexity analysis, the performance of an algorithm is analyzed by picking the worst possible instance for a given size parameter n, as defined below.

Definition 4.1 (Worst-Case Complexity)**.** *Let $T_A(x)$ be the running time of algorithm A for instance x. Let X_n be set of all inputs of A with size of n.*

Then, the worst-case complexity of algorithm A for size n is:

$$W(A, n) = \max_{x \in X_n} [T_A(x)].$$

This kind of analysis has the strong advantage that it does not need an input model to be used, a worst case instance suffices. It also provides a strong upper bound on how badly an algorithm can perform.

The worst case analysis can and has been used extensively to prove good performance of some algorithms. However, there is plenty of algorithms, such as the simplex algorithm (DANTZIG, 1963; MURTY, 1983) and algorithms for the knapsack problem (MARTELLO; TOTH, 1990), which have poor worst case complexity but are still used in several real-world scenarios. These algorithms have exponential worst case complexities, but perform rather well for "practical" scenarios.

Average case analysis was introduced to better understand these algorithms. It analyzes running time over a distribution of inputs, and depends entirely on this distribution.

Definition 4.2 (Average-Case Complexity)**.** *Let $T_A(x)$ be the running time of algorithm A for instance x. Let X_n be set of all inputs of A with size less or equal to n.*

Then, given a probability distribution μ_n on X_n, the average-case complexity of algorithm A for size n is

$$AVG(A, n) = \mathop{\mathbf{E}}_{x \sim \mu_n} [T_A(x)] \ ,$$

where $x \sim \mu_n$ represents that x is a random instance chosen accordingly to distribution μ_n.

A good average case complexity may be a hint that an algorithm performs well in practice, but this evidence is highly dependent on the input distribution used in the analysis. A bound on the average case complexity on one distribution says nothing on the behavior of the algorithm with respect to other distributions. Furthermore, defining a distribution that corresponds to those found on practice can be a difficult or impossible task

to accomplish. It is often the case that the analysis is performed on simpler distributions than found in real cases, in which case it may give a view of the algorithm that is too optimistic.

While the worst-case analysis is a powerful tool to demonstrate that an algorithm can be used with performance guarantees in real circumstances, it may not provide the best argument against using an algorithm, since worst case instances may not appear in some of real world scenarios. The average-case analysis can also not be ideal to analyze some of these algorithms, since finding a distribution model that fits well with practical scenarios is often a hard task. In many cases this means that the distribution used is oversimplified and in many cases "too forgiving" on the algorithm being analyzed.

Moreover, even if worst case instances occur in practice, they might often have small perturbations associated with it. This perturbation may be, for example, due to noise in data gathering from physical instruments, or due to the fact that an algorithm can suffer from small rounding errors due to floating point calculations. One might wonder whether these worst case instances remain exponential if we are allowed to slightly perturb them at random. The smoothed analysis contemplates this kind of algorithm analysis.

The smoothed analysis of algorithms was defined by Spielman and Teng in (SPIELMAN; TENG, 2004). It is a relatively new approach to the analysis of algorithms in contrast to the worst case and average case analysis. One of the main motivations for this new kind of analysis is to understand the simplex algorithm, which has exponential worst case behavior and still is an efficient algorithm in practice.

The smoothed complexity of an algorithm is the maximum over inputs of the expected running time of the algorithm being analyzed under slight perturbations of that input. This also means that smoothed complexity is measured not only in the size n of the input but also in the magnitude σ of the perturbation.

Definition 4.3 (Smoothed Complexity)**.** *Let $T_A(x)$ denote the running time of algorithm A for instance x. Let X_n be the set of instances of A with size n.*

For an instance x and a magnitude parameter σ, let $\mu_\sigma(x)$ denote the set of instances that can be obtained from x by applying random perturbations of magnitude σ. Then, the smoothed complexity of algorithm A under perturbations of magnitude σ is:

$$S(A, n, \sigma) = \max_{x \in X_n} \left(\mathop{\mathbf{E}}_{\bar{x} \in \mu_\sigma(x)} [T_A(\bar{x})] \right).$$

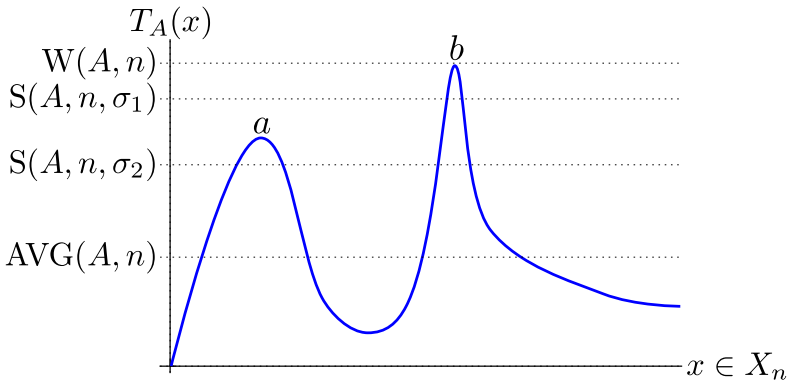

Figure 4.1: Illustration of different complexity analysis of the same algorithm A. The magnitude σ_1 is lower than σ_2.

One of the most important aspects of smoothed analysis is to analyze NP-Complete problems, which have no known algorithms with polynomial worst case complexity. To this end, we can say that an algorithm has polynomial smoothed complexity if it is polynomially bounded in both n and $\frac{1}{\sigma}$.

Definition 4.4 (Polynomial Smoothed Complexity)**.** *Let $T_A(x)$ denote the running time of algorithm A for instance x. Let X_n and $\mu_\sigma(x)$ be the same as in Definition 4.3.*

Then, algorithm A has polynomial smoothed complexity if there exist constants n_0, σ_0, c, k_1, k_2 such that for all $n \geq n_0$ and $0 \leq \sigma \leq \sigma_0$,

$$\max_{x \in X_n} \left(\mathop{\mathbf{E}}_{\bar{x} \in \mu_\sigma(x)} [T_A(\bar{x})] \right) \leq c\, n^{k_1} \sigma^{-k_2} \, .$$

If the perturbation magnitude σ is set to zero, then the smoothed complexity is the same as the worst case complexity, since there is no perturbation. As σ approaches infinity, the analysis becomes more and more similar to the average case analysis, since the perturbation is so large that the specific instance analyzed becomes irrelevant. An algorithm with a good smoothed complexity and bad worst case complexity will perform well on the neighborhood of all instances, even if for a specific worst case instance it behaves badly. In other words, it is a strong evidence of good performance in practice, since even if worst case instances exist, they are very susceptible to small perturbations.

In Figure 4.1, we can see an illustration of how these various complexity measures relate. In the horizontal axis the set of all instances to A with size n are showed, for a fixed n, while in the vertical axis the running time of those instances is presented. Note that $\sigma_1 < \sigma_2$, and in this example it means that while the instance which gives the upper bound for $\mathrm{S}(A, n, \sigma_1)$ is b, for $\mathrm{S}(A, n, \sigma_2)$ it is a.

4.1 Perturbation Models

There can be many different kinds of perturbations on the smoothed analysis. It allows the type of perturbation to be specified depending on the particular algorithm being analyzed. For continuous problems, two perturbation models are important to note: the uniform perturbation model and the Gaussian model.

In the uniform perturbation model, the perturbed instances are chosen from a uniformly random range that is defined by the perturbation magnitude σ and the original instance. These perturbations can be either additive or relative. The same reasoning applies to the Gaussian perturbation model, only now the perturbations are derived from a Gaussian distribution.

For instances in the $\mathbb{R}^n$, the above ideas can be formally defined as follows.

Definition 4.5 (Uniform Perturbations)**.** *Let $x = (x_1, ..., x_n) \in \mathbb{R}^n$ be an instance of algorithm A.*

A uniform perturbation of magnitude σ of x is a random vector $\bar{x} = (\bar{x}_1, ..., \bar{x}_n)$, where $\bar{x}_i = x_i + \varepsilon_i$, and ε_i is taken independently uniformly at random from the range $[-\sigma, \sigma]$.

Definition 4.6 (Relative Uniform Perturbations)**.** *Let $x = (x_1, ..., x_n) \in \mathbb{R}^n$ be an instance of algorithm A.*

A uniform perturbation of magnitude σ of x is a random vector $\bar{x} = (\bar{x}_1, ..., \bar{x}_n)$, where $\bar{x}_i = x_i(1 + \varepsilon_i)$, and ε_i is taken independently uniformly at random from the range $[-\sigma, \sigma]$.

An alternative definition can be made for these perturbation models, where the range where the deviation is taken is never negative, i.e., from 0 to σ. For Gaussian perturbations, similar definitions are stated below.

Definition 4.7 (Gaussian Perturbations)**.** *Let $x = (x_1, ..., x_n) \in \mathbb{R}^n$ be an instance of algorithm A.*

A Gaussian perturbation of magnitude σ of x is a random vector $\bar{x} = (\bar{x}_1, ..., \bar{x}_n)$, where $\bar{x}_i = x_i + \varepsilon_i$, and ε_i is a Gaussian random variable with standard deviation σ.

Definition 4.8 (Relative Gaussian Perturbations)**.** *Let $x = (x_1, ..., x_n) \in \mathbb{R}^n$ be an instance of algorithm A.*

A Gaussian perturbation of magnitude σ of x is a random vector $\bar{x} = (\bar{x}_1, ..., \bar{x}_n)$, where $\bar{x}_i = x_i(1 + \varepsilon_i)$, and ε_i is a Gaussian random variable with standard deviation σ.

4.2 Smoothed Price of Anarchy

While the original idea of the smoothed analysis was to measure the time complexity of algorithms, it is suited to analyze several other quality measures, like space or the number of cache-misses of an algorithm. A particular quality measure that has not been explored to the present moment is to compare the Price of Anarchy ratio used in game theory.

We extend the smoothed analysis idea for time complexity analysis to analyze the Price of Anarchy measure. The idea is to randomly perturb a given input instance and to analyze the expected Price of Anarchy on the perturbed instances. We first define the Smoothed Price of Anarchy of an instance I.

Definition 4.9 (Smoothed Price of Anarchy of I)**.** *Suppose that we are given a class of games $\mathcal{G}$. Given an instance $I = (G, (S_i), (c_i)) \in \mathcal{G}$, we randomly perturb I by adding some random noise to the input data.*

Let $\bar{I}$ be an instance that can be obtained from I by random perturbations and let σ be a parameter for the magnitude of perturbation. The Smoothed Price of Anarchy (SPoA) *of I is then defined as*

$$\mathit{SPoA}(I, \sigma) = \frac{\max_{s \in NE(\bar{I})} \mathbf{E}[\bar{C}(s)]}{\min_{s \in S} \mathbf{E}[\bar{C}(s)]},$$

where the expectation is taken over all instances $\bar{I}$ that are obtainable from I by random perturbations of magnitude σ. Here $\bar{C}$ refers to the cost of the perturbed instance.

Note that there might be several ways to perturb the input instance, as seen in Section 4.1. How this should be done depends on the underlying application. Finally, the Smoothed Price of Anarchy of a game $\mathcal{G}$ can be defined.

Definition 4.10 (Smoothed Price of Anarchy)**.** *Using the same assumptions from Definition 4.9, the Smoothed Price of Anarchy of $\mathcal{G}$ is then*

$$\mathit{SPoA}(\mathcal{G}, \sigma) = \max_{I \in \mathcal{G}} \mathit{SPoA}(I, \sigma) \ .$$

5 SMOOTHED COMPLEXITY OF BIMATRIX GAMES

Bimatrix games are usually presented in explicit normal form, with the payoffs often taken as measurements from real world scenarios. Therefore, it is useful to study how perturbations affect the behavior of algorithms used to find equilibria in these games.

As seen in Section 2.2, the Lemke-Howson Algorithm is used to find an equilibrium in bimatrix games. It works in practice by shifting variables around a tableau, and is very susceptible to numerical errors, which makes using smoothed complexity for analysis of this particular algorithm very important. The smoothed complexity of the Lemke-Howson algorithm can be defined as below.

Definition 5.1 (Smoothed Complexity of the Lemke-Howson algorithm)**.** *Let $T(A, B)$ denote the running time of the LH algorithm on the bimatrix game defined by (A, B) and let $\bar{A}$, $\bar{B}$ denote perturbations of magnitude σ of A and B, respectively. Then, the smoothed complexity of the LH algorithm under perturbations of magnitude σ is:*

$$S_{LH}(n, \sigma) = \max_{A,B \in \mathbb{R}^{n \times n}} \mathbf{E}_{\bar{A},\bar{B}} \left[T(\bar{A}, \bar{B}) \right] \quad ,$$

Note that the type of perturbation is not described. For perturbations on bimatrix games, additive perturbations can be common. As seen in Chapter 4, the two of the most common ones are uniform and Gaussian perturbations. In the Gaussian model, the new values $\bar{a}_{i,j}, \bar{b}_{i,j}$ are obtained from $a_{i,j}, b_{i,j}$ by adding independent random variables distributed as Gaussians with mean 0 and standard deviation σ. In the uniform model, the new values are randomly and uniformly chosen from $[a_{i,j}-\sigma, a_{i,j}+\sigma]$ and $[b_{i,j}-\sigma, b_{i,j}+\sigma]$.

In (CHEN; DENG, 2006), it is proved that the smoothed complexity of LH algorithm is not polynomial, both for uniform perturbations as for Gaussian perturbations. The authors prove that there is no algorithm that finds an ϵ-approximate Nash equilibrium in polynomial time. It then proceeds to prove that this is equivalent to saying that the problem of finding a Nash equilibrium of a bimatrix game is not in smoothed polynomial time under σ-perturbations.

The way the proof is constructed does not provide a constructible instance, making it harder to assess how likely is this instance to happen in practice. Hence, in our experiments we generated the instances described by Savani and Stengel (SAVANI; STENGEL, 2004), which are the only ones known to be exponential for the LH algorithm.

In this chapter we describe how the only known worst case instances are generated, and then show the experiments we conducted on them. We analyze these instances with perturbations with different magnitudes and show that these instances are not the same as the one predicted by the results of(CHEN; DENG, 2006).

5.1 Known Worst-Case Instances

In (SAVANI; STENGEL, 2004), Savani and Stengel introduced instances for which the LH algorithm takes an exponential number of pivoting iterations to find a Nash equilibrium. Exponentially long paths have been discovered for the LCP (linear complementary problem) in (MURTY, 1978) and (FATHI, 1979). However, even though a Nash equilibrium can be a solution to some LCPs, the problems covered in these papers are not bimatrix games. Therefore, these instances are, to the best of our knowledge, the only known instances where a bimatrix game has an exponential LH path.

These worst case instances are based on dual cyclic polytopes (ZIEGLER, 2001; GRUNBAUM, 2003) with dimension d and $2d$ facets. The inequalities that define this kind of polyhedra are known in arbitrary dimensions (GRUNBAUM, 2003), which make these polytopes a good start for creating scalable worst case instances. These polytopes can also be used to construct games with an exponential number of equilibria (STENGEL, 1999).

Given two players P_1 and P_2, with corresponding payoff matrices A and B with size $d \times d$, a worst case instance is constructed based on the polytopes described by these matrices. In order to construct these worst case instances and record them in the two matrices A and B, we first construct the polytope P, which can be seen as the payoff matrix B of the player P_2. Using this matrix, we can then derive the payoff matrix A of the player P_1.

In order to obtain this dual cyclic polytope P, we first obtain a cyclic polytope P' with $2d$ vertices, where each vertex $\mu(t_i)$ is obtained from the *moment curve* $\mu : x \longmapsto (x, x^2, ..., x^d)$, for $1 \leq i \leq 2d$. It is known that a cyclic polytope can be formed by the vertices of the moment curve as long as $t_1 < t_2 < ... < t_{2d}$, any set of $t_i \in \mathbb{R}$ is a possible choice. For further details see (GRUNBAUM, 2003).

This polytope is then translated in order to ensure that it has the origin in its interior. A simple way of doing that is by subtracting the arithmetic mean μ_{mean} of the points for each vertex, thus resulting in the following polytope:

$$P'' = \left\{ z \in \mathbb{R}^d | c_i z \leq 1, \;\; 1 \leq i \leq 2d \right\}, \qquad c_i = \mu_i - \mu_{mean}$$

These inequalities can be divided into two $d \times d$ matrices C and D, with C containing $c_1, ..., c_d$ and D containing $c_{d+1}, ..., c_{2d}$. The polytope P is defined in the following way:

$$P = \left\{ x \in \mathbb{R}^d | x \leq \mathbf{0}, \; -DC^{-1}x \leq r \right\}, \qquad r = \mathbf{1} - DC^{-1}\mathbf{1}$$

We must now re-normalize the inequalities so that the right-hand side is one. This is done in order to make this polytope also a set of strategies for one of the players. By multiplying $-DC^{-1}$ by a diagonal matrix S such that $s_{i,i} = \frac{1}{r_i}$ and $s_{i,j} = 0$ if $i \neq j$, we accomplish this goal. The normalized polytope can be defined as:

$$P = \left\{ x \in \mathbb{R}^d | x \leq \mathbf{0}, \; -SDC^{-1}x \leq \mathbf{1} \right\}.$$

Finally, the payoff matrix B is defined by $B = (-SDC^{-1})^T$. The symmetric case where matrix $A = B^T$ does not work in this case, since it leaves a short path as the solution. Hence, we must obtain the payoff matrix A by copying the matrix B with some permutations in place. The function λ defined below helps to define these permutations:

$$\lambda(k) = \begin{cases} k, & k = 1 \ \ or \ \ k = d, \\ k + (-1)^k, & 2 \leq k \leq d - 1, \\ k - (-1)^k, & d + 1 \leq k \leq 2d. \end{cases}$$

The matrix A and its entries $a(i, j)$ then can be obtained from the entries $b(i, j)$ of B by the following relation:

$$a(\lambda(i),\ \lambda(j + d) - d) = b(j, i) \qquad 1 \leq i, j \leq d.$$

This ensures that this bimatrix game has exactly one equilibrium and that the LH algorithm will take an exponentially long path to find it.

5.2 Results

5.2.1 An Exact Implementation of the Lemke-Howson Algorithm

For the LH algorithm, two implementations were tested, one from Codenotti et al. (CODENOTTI; ROSSI; PAGAN, 2008) and one from the GAMBIT (MCKELVEY; MCLENNAN; TUROCY, 2010) project. The implementations are open source (GNU GPLv2) and freely available.

We developed a solution based on the implementation of Codenotti et al. The software is written in C/C++ and originally used a floating point representation of the game's payoffs. This could easily lead to incremental errors and numerical instability in the solver. We extended the software to deal with rational numbers using the GMP (GNU Multiple Precision Arithmetic Library) (GRANLUND et al., 2011). Even though our solver cannot handle irrational numbers, note that any irrational number on the input would not be treated as such by using the old representation as well. Using this representation, all the results are exact, even though the running time can be slower depending on how big the operands are.

The solver with these modifications proved to be more stable and faster than GAMBIT's implementation. Our implementation is available at (RODRIGUES, 2011).

The generator of the instances used in our experiments was developed using the SAGE software system (STEIN et al., 2010). SAGE is a collection of various open source softwares intended to join numerous packages into a single interface based on Python. It consists of nearly 100 open-source packages in an unified interface. It has many tools in areas such as combinatorics, graph theory and symbolic linear algebra.

5.2.2 Experimental Results

For our experiments, the tests were performed varying the number of strategies n from 2 up to 20. The starting pivot used was also a variable ranging from 1 to $2n$.

Table 5.1 reproduces the results of (SAVANI; STENGEL, 2004). The columns represent the number of pure strategies available for each player, i.e. the size of the matrices, while the rows inform which variable is chosen to enter the base on the first step of the LH algorithm (refer to Section 2.2 for more details on the Lemke-Howson algorithm). The measure used is the number of times the algorithm performs a pivoting operation. This measure is denoted as LH *path lengths*. Note that while the path lengths are directly related to the execution times for these instances, this may not hold for all instances, since our implementation uses an exact representation of rational numbers. Since the perturbations used are so small, this can lead to rationals with big numerators and denominators,

Table 5.1: Path lengths for worst case instances.

Pivot Number	2	4	6	8	10	12	14
1	**4**	**20**	**88**	**376**	**1596**	**6764**	**28656**
2	**4**	8	24	92	380	1600	6768
3	4	8	24	92	380	1600	6768
4	4	**20**	24	40	108	396	1616
5		10	24	40	108	396	1616
6		10	**88**	92	108	176	464
7		10	36	92	108	176	464
8		10	36	**376**	380	396	464
9			16	146	380	396	464
10			16	146	**1596**	1600	1616
11			36	42	612	1600	1616
12			36	42	612	**6764**	6768
13				42	152	2586	6768
14				42	152	2586	**28656**
15				146	68	618	10948
16				146	68	618	10948
17					152	178	2592
18					152	178	2592
19					612	178	644
20					612	178	644
21						618	288
22						618	288
23						2586	644
24						2586	644
25							2592
26							2592
27							10948
28							10948

forcing our implementation to use big integer representations to deal with these complications. This can lead to a substantive decrease in performance.

For the perturbations of the generated worst case instance, the uniform perturbation model was used. For the matrix values $a_{i,j}$ and $b_{i,j}$, a new value was uniformly chosen from the intervals $[a_{i,j} - \sigma, a_{i,j} + \sigma]$ and $[b_{i,j} - \sigma, b_{i,j} + \sigma]$.

In Figure 5.1, the parameter σ has been set to 10^i, for $i = 0, -1, ..., -8$. The starting pivot was fixed to be n, since this is the worst case for the algorithm. For each of the worst case instances (with $n \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$) and each σ, 100 uniformly perturbed instances were generated and executed, and their average is the data used on the plot.

In Figure 5.2, the magnitude σ has been set from 10^{-2} down to 10^{-8}, with the starting pivot fixed at n. For each of the tested magnitudes, 100 uniformly perturbed instances were generated and executed.

Both plots indicates that, contrary to what one might expect due to the proof by Chen et al., the running times the worst case instances of Savani and Stengel do indeed drop to

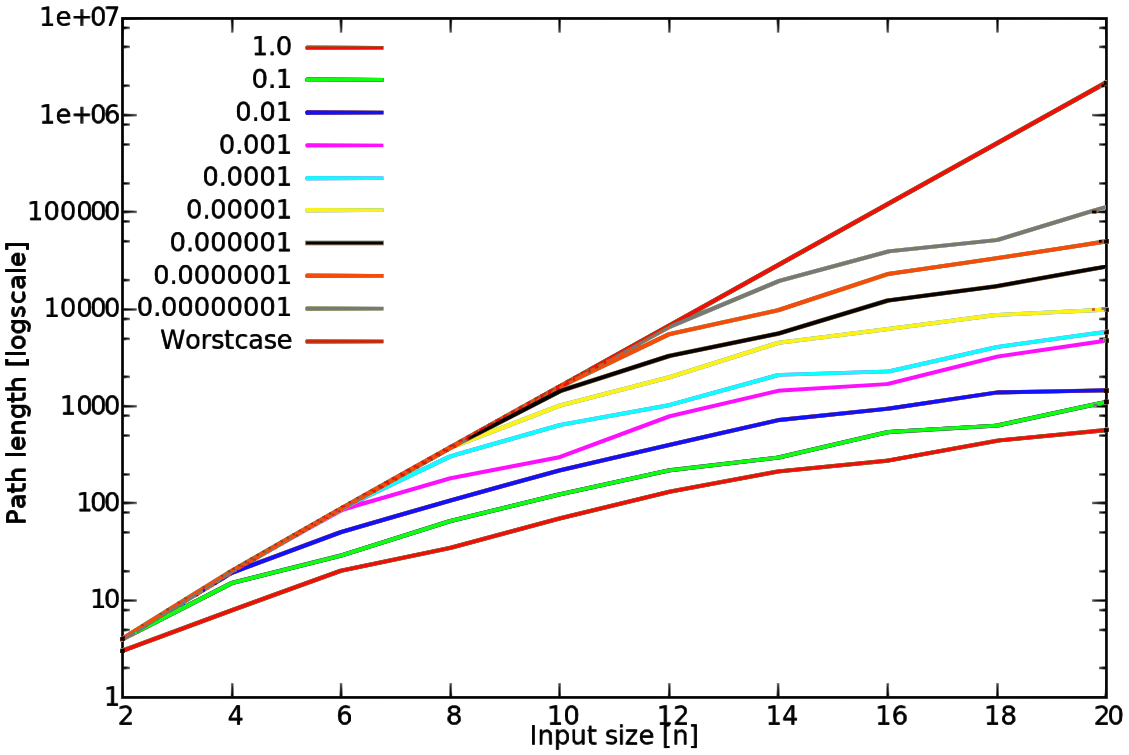

Figure 5.1: Path length for each perturbed instance as its size grows with constant perturbation magnitudes. Note that the list on the legend is in inverse order to the data on the graphic.

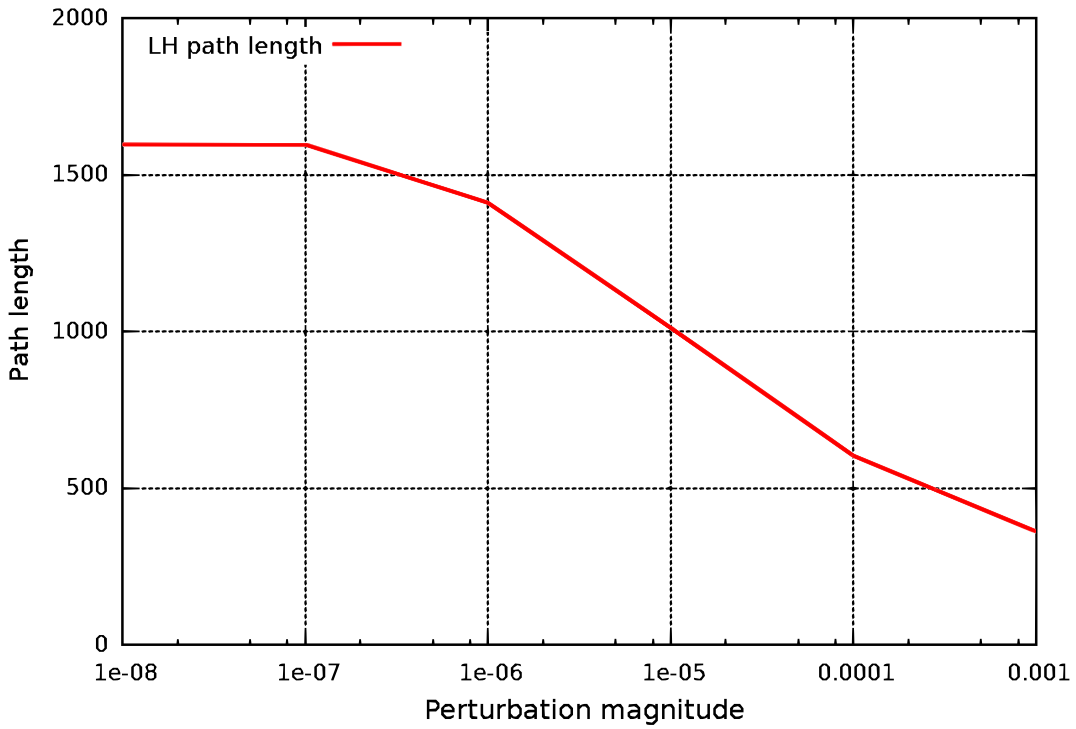

Figure 5.2: Path length for each perturbed instance as its perturbation magnitude grows with the size of the input fixed $n = 10$.

a polynomial running time with very small perturbations. Note that these perturbations are extremely small ones, since the values of the original instances can go up to $(2n)^n$.

Another indication that this may be the case comes from Figure 5.3. In this plot the perturbation was set as a function of the input size. There we can see that even when the σ gets polynomially small ($\sigma = \frac{1}{n^4}$), it does not keep up with the worst case instance. The polynomial decrease in σ ends up behaving in a similar fashion to the fixed sigma, further illustrating our point.

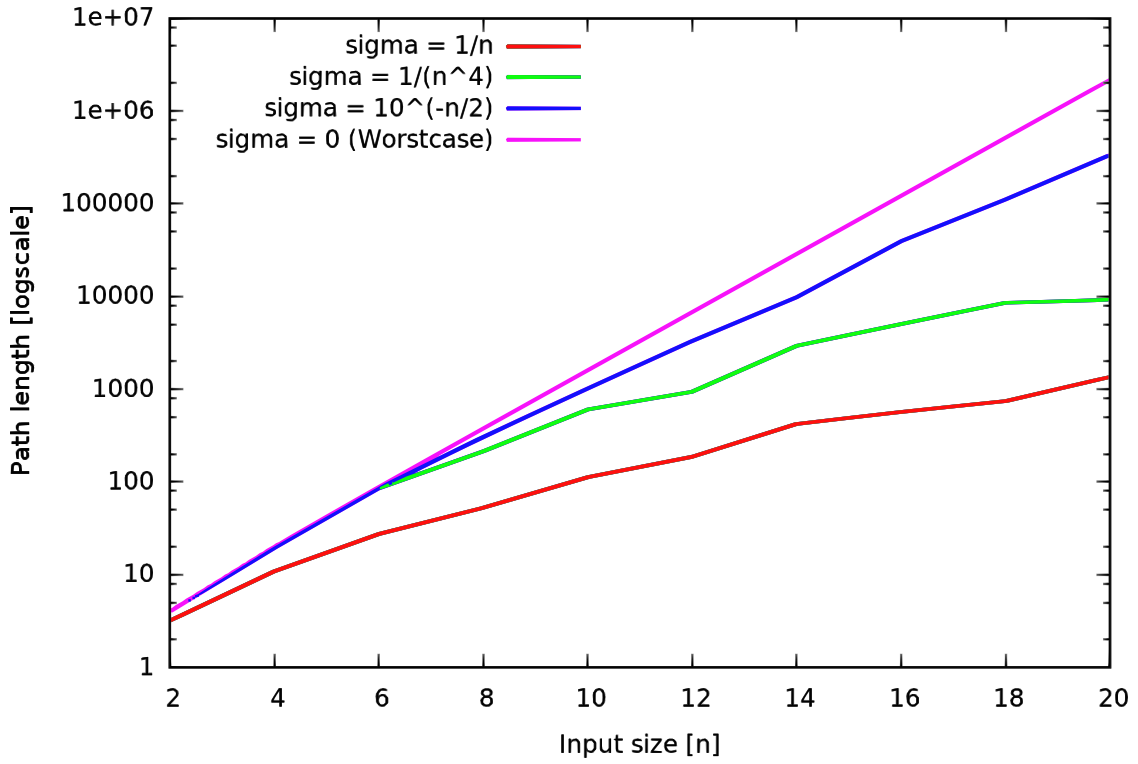

Figure 5.3: Path length for each perturbed instance as its size grows for perturbation magnitudes as functions of n. Note that the list on the legend is in inverse order to the data on the graphic.

These worst case instances having smoothed polynomial running times does not contradict the proof by Chen et al. The proof only shows that there is at least one worst case instance where the smoothed complexity does not drop to polynomial time. Hence, what these experiments seen to indicate is that, while this instance should exist, it has not yet been discovered. Observe that these conclusions are based on a limited set of measurements. In particular, we could only measure small dimensions ($n \leq 20$) but we try to extrapolate an assumption behavior from these data.

The fact that this hard instance is not yet known is a strong indicator that these cases are very rare in real life instances, and that the LH algorithm performs quite well to most known cases, and very well for known worst case instances, assuming that small perturbations are allowed on the input.

Such perturbations are in fact common on real instances, usually caused by a floating point representation that is not precise enough. In fact, had we not made our solver use an exact rational representation, errors caused by the more common floating point representation would have caused changes in the original worst case instance with dimensions as small as 10 in the machines used for the experiment.

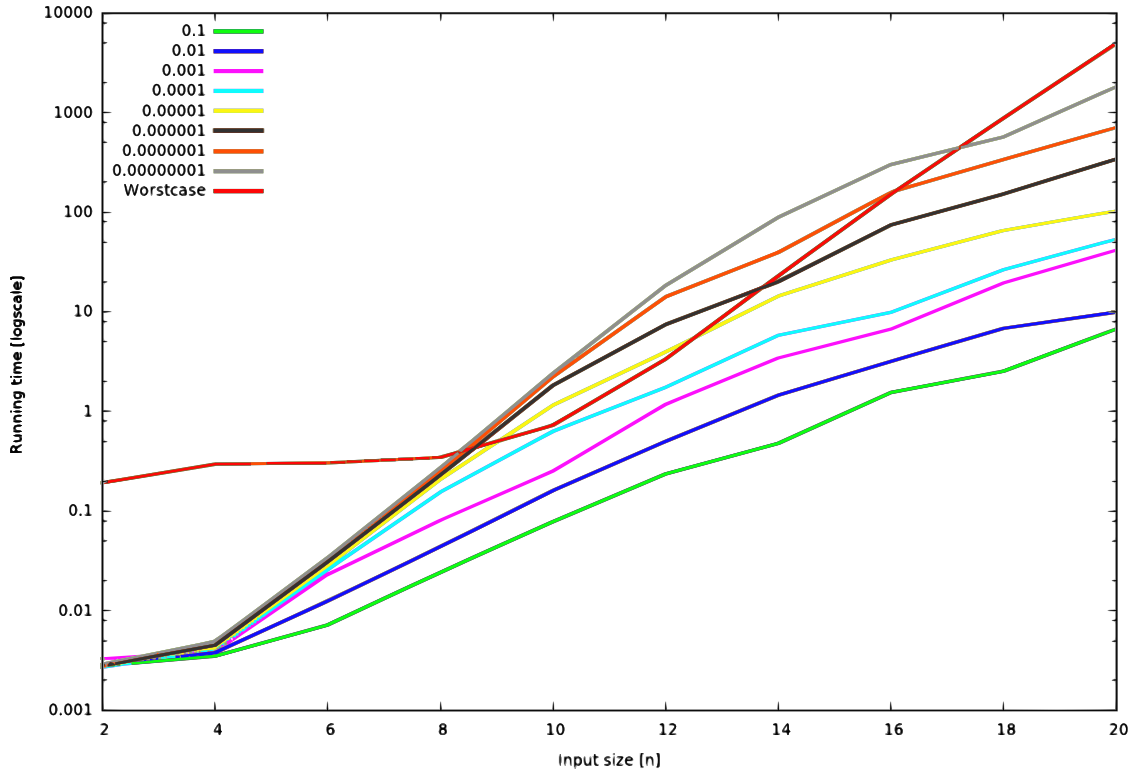

Figure 5.4: Time measured for each perturbed instance as its size grows (in seconds). Note that the list on the legend is in inverse order to the data on the graphic.

In order to draw definite conclusions, it would be preferable to execute instances larger than $n = 20$. However, even though there is a significant decrease in the LH path length, the running time does not drop as much as the path length. In Figure 5.4, we present the running times for the same instances. We can see that the perturbed instances are a lot closer to the worst case instance, even slower in some cases. This is due to the extra precision required to compute the equilibrium in an exact manner with very small σ values. When a perturbation as small as 10^{-8} is added, much of the algorithm execution time is dedicated to perform operations on big integers that would overload if not treated correctly.

In Table 5.2, Table 5.4 and Table 5.6, results are presented for all possible choices of pivots, with the average values presented for the perturbation magnitude σ of 0.01, 0.2 and 1, respectively. Table 5.3, Table 5.5 and Table 5.7 show the minimum and maximum path length for each starting pivot choice and instance size. The maximum dimension showed is 14, resulting in a maximum of 28 possible starting pivots. In these tables it is possible to notice that there is a big variation on the path lengths for the perturbed instances, as shown in the difference between the minimum path lengths and maximum path lengths observed. Often the minimum path length is lower than 50, which demonstrates how much such small perturbations can make the instances easier for the LH algorithm.

Table 5.2: Average path lengths for $\sigma = 1$

Pivot Number	2	4	6	8	10	12	14
1	2.90	7.40	11.04	30.15	76.80	58.40	86.05
2	2.80	7.10	11.52	15.80	22.65	44.50	153.45
3	3.65	6.40	14.85	26.90	78.75	61.30	90.70
4	2.45	8.45	18.42	32.50	35.95	45.00	61.00
5		7.05	14.09	20.50	65.40	54.45	75.20
6		6.30	17.71	29.05	45.85	72.50	122.00
7		6.20	18.09	20.85	54.40	40.60	81.35
8		6.30	7.33	31.65	47.05	81.35	166.50
9			13.19	28.35	40.60	48.20	99.15
10			10.66	15.50	48.30	89.55	198.40
11			15.95	25.35	42.85	55.70	111.60
12			9.90	13.70	35.70	146.85	220.40
13				22.40	42.50	111.70	153.00
14				17.45	21.80	88.50	278.35
15				32.75	50.85	69.45	182.45
16				19.15	28.95	22.05	189.30
17					28.35	61.35	127.40
18					35.80	38.00	33.70
19					50.20	51.90	105.25
20					50.50	47.35	50.30
21						39.20	57.25
22						48.60	65.90
23						84.25	69.20
24						46.00	66.90
25							108.20
26							80.85
27							245.05
28							83.35

Table 5.3: Path with minimum and maximum lengths for $\sigma = 1$

Pivot Number	2	4	6	8	10	12	14
1	2 – 5	4 – 15	5 – 33	5 – 140	5 – 268	20 – 303	20 – 500
2	2 – 4	4 – 15	5 – 27	8 – 43	11 – 48	14 – 113	26 – 566
3	2 – 5	4 – 10	4 – 40	5 – 140	5 – 273	21 – 385	21 – 515
4	2 – 4	4 – 17	5 – 35	9 – 79	11 – 95	21 – 130	21 – 186
5		4 – 14	4 – 37	4 – 83	5 – 174	21 – 235	20 – 538
6		4 – 15	4 – 34	9 – 66	11 – 119	21 – 177	21 – 465
7		4 – 13	5 – 27	4 – 65	5 – 111	21 – 159	21 – 373
8		4 – 14	4 – 13	8 – 73	9 – 144	21 – 225	21 – 783
9			4 – 33	9 – 57	4 – 150	20 – 158	21 – 419
10			7 – 27	8 – 30	8 – 192	21 – 228	21 – 851
11			4 – 28	9 – 52	9 – 133	20 – 340	20 – 538
12			7 – 20	7 – 26	10 – 131	20 – 521	21 – 1390
13				8 – 58	9 – 112	20 – 611	37 – 649
14				7 – 45	7 – 68	25 – 184	20 – 1405
15				11 – 72	9 – 135	21 – 189	21 – 599
16				7 – 60	7 – 104	10 – 42	25 – 816
17					8 – 95	21 – 193	21 – 484
18					7 – 137	21 – 108	10 – 105
19					9 – 173	20 – 135	21 – 283
20					7 – 156	21 – 196	21 – 135
21						16 – 86	21 – 187
22						21 – 258	21 – 220
23						10 – 346	20 – 200
24						21 – 177	21 – 261
25							23 – 408
26							21 – 504
27							10 – 1086
28							21 – 495

Table 5.4: Average path lengths for $\sigma = 0.2$

Pivot Number	2	4	6	8	10	12	14
1	4.00	7.70	19.40	42.00	69.05	128.90	191.80
2	4.00	8.45	11.60	21.15	39.60	100.00	87.90
3	4.00	7.15	15.30	45.75	63.10	83.90	118.05
4	4.00	13.25	22.60	30.25	28.45	42.25	67.25
5		9.90	11.95	32.35	59.60	56.75	95.15
6		6.60	29.65	28.95	62.40	80.70	117.75
7		11.00	23.00	22.95	54.60	79.60	62.65
8		7.00	11.10	49.40	72.25	118.55	182.75
9			17.75	34.25	49.90	67.60	49.30
10			11.80	19.85	84.55	124.95	245.70
11			21.85	32.05	70.95	89.85	73.45
12			12.10	13.60	42.20	135.50	357.65
13				16.45	53.70	86.70	301.25
14				25.25	22.35	111.80	252.20
15				18.65	47.35	68.80	231.00
16				31.05	35.10	24.95	245.15
17					24.60	54.20	193.30
18					56.00	38.05	36.30
19					68.20	57.70	164.00
20					71.15	63.10	32.90
21						72.90	89.50
22						78.00	75.80
23						135.90	76.90
24						82.15	116.30
25							110.15
26							139.75
27							152.25
28							118.85

Table 5.5: Path with minimum and maximum lengths for $\sigma = 0.2$

Pivot Number	2	4	6	8	10	12	14
1	4 – 4	4 – 16	5 – 76	5 – 104	20 – 383	20 – 996	20 – 1027
2	4 – 4	6 – 15	8 – 18	12 – 55	14 – 82	29 – 190	24 – 399
3	4 – 4	4 – 15	5 – 44	5 – 144	21 – 264	20 – 767	20 – 1207
4	4 – 4	6 – 22	9 – 46	9 – 59	15 – 66	21 – 110	21 – 183
5		6 – 16	4 – 24	5 – 93	21 – 240	21 – 281	20 – 745
6		4 – 10	8 – 58	9 – 86	21 – 176	21 – 246	21 – 432
7		8 – 15	9 – 38	4 – 60	20 – 150	21 – 358	21 – 502
8		5 – 8	4 – 29	8 – 136	21 – 186	21 – 497	21 – 932
9			8 – 32	9 – 69	10 – 210	20 – 193	20 – 320
10			7 – 31	10 – 42	20 – 337	21 – 747	21 – 1008
11			9 – 43	9 – 74	20 – 328	24 – 208	20 – 402
12			7 – 32	7 – 40	18 – 85	20 – 773	21 – 1211
13				8 – 32	21 – 156	21 – 452	106 – 880
14				7 – 73	10 – 109	25 – 336	20 – 689
15				9 – 62	20 – 79	21 – 278	21 – 585
16				7 – 110	13 – 84	10 – 71	34 – 698
17					14 – 109	20 – 230	21 – 522
18					25 – 164	21 – 86	10 – 146
19					10 – 184	20 – 212	21 – 430
20					24 – 325	21 – 238	21 – 73
21						16 – 199	21 – 229
22						21 – 349	21 – 283
23						10 – 306	20 – 190
24						21 – 377	21 – 726
25							35 – 408
26							21 – 1361
27							10 – 508
28							21 – 622

Table 5.6: Average path lengths for $\sigma = 0.01$.

Pivot Number	2	4	6	8	10	12	14
1	4.00	16.75	38.00	41.90	159.90	179.90	734.10
2	4.00	9.90	18.90	49.30	73.55	122.85	424.35
3	4.00	7.30	31.60	35.35	142.15	188.70	586.85
4	4.00	19.05	29.15	41.20	53.40	81.10	102.45
5		10.50	20.70	34.15	53.10	94.50	365.95
6		8.80	54.25	98.10	139.50	157.40	172.15
7		9.90	27.60	35.80	51.60	56.80	338.55
8		8.80	25.85	109.70	219.55	254.15	207.65
9			15.25	77.75	90.15	88.40	169.10
10			14.95	56.25	222.35	291.20	338.55
11			27.50	43.35	140.15	167.20	213.85
12			29.80	16.25	118.70	425.20	609.90
13				25.30	94.65	324.30	370.85
14				30.80	28.95	164.05	727.15
15				53.15	49.40	202.00	328.60
16				38.60	36.55	62.90	332.10
17					58.10	98.95	263.15
18					71.75	49.75	175.65
19					123.20	93.70	206.25
20					119.70	107.05	79.35
21						115.70	187.20
22						123.75	165.00
23						166.50	107.50
24						139.95	303.85
25							171.85
26							427.90
27							638.60
28							556.30

Table 5.7: Path with minimum and maximum lengths for $\sigma = 0.01$.

Pivot Number	2	4	6	8	10	12	14
1	4 – 4	8 – 21	20 – 78	20 – 190	20 – 890	21 – 750	21 – 5965
2	4 – 4	8 – 14	10 – 28	26 – 88	24 – 226	24 – 440	27 – 1422
3	4 – 4	6 – 8	20 – 46	21 – 96	21 – 680	21 – 762	21 – 4668
4	4 – 4	18 – 20	22 – 42	21 – 72	21 – 122	38 – 243	40 – 216
5		10 – 12	10 – 30	20 – 75	20 – 225	21 – 601	21 – 1007
6		8 – 10	33 – 91	21 – 186	21 – 310	35 – 472	37 – 582
7		9 – 11	22 – 38	12 – 92	20 – 176	20 – 350	21 – 988
8		8 – 10	10 – 42	20 – 229	21 – 468	35 – 770	37 – 1256
9			12 – 32	21 – 144	34 – 222	20 – 348	21 – 510
10			10 – 26	25 – 114	20 – 502	35 – 1047	35 – 1618
11			10 – 41	20 – 70	21 – 263	82 – 492	20 – 770
12			21 – 38	10 – 37	34 – 262	34 – 1288	35 – 5315
13				15 – 50	21 – 187	35 – 929	47 – 978
14				21 – 56	10 – 64	53 – 558	34 – 5456
15				10 – 145	20 – 89	35 – 770	35 – 1154
16				21 – 106	21 – 80	16 – 160	90 – 1524
17					30 – 124	34 – 241	35 – 1043
18					21 – 254	23 – 150	42 – 328
19					10 – 357	34 – 231	35 – 447
20					21 – 529	23 – 330	23 – 243
21						36 – 434	35 – 383
22						23 – 313	23 – 459
23						37 – 677	34 – 567
24						23 – 505	23 – 1217
25							41 – 606
26							23 – 2594
27							35 – 2444
28							23 – 3380

6 SMOOTHED PRICE OF ANARCHY IN THE TRAFFIC ASSIGNMENT PROBLEM

The Traffic Assignment Problem (FLORIAN; HEARN, 1995; WARDROP, 1952) models applications in which traffic participants (also called *users*) choose routes in a given road network so as to minimize their individual travel times. As seen in Chapter 3, each arc of the network has an associated latency function that expresses the flow-dependent delay that users experience if they travel along that arc. The goal of every user is to choose a path from his origin to his destination such that the total delay to travel along this route is minimized.

A user equilibrium is a flow that happens when no player can unilaterally reduce his travel time by using a different path to its destination. A system optimum on the other hand is an optimal flow which happens when the assignment for all players minimizes the total travel times on the network. Because the delay of each user also depends on the choices made by the others, this problem can also naturally be interpreted as a strategic game in which players (users) compete for resources (roads) and every player acts selfishly in the sense that he attempts to choose a route of minimum delay.

There exist multiple ways to influence user equilibria, usually through the use of tolls (HEARN; RAMANA, 1998), with the intent of making the gap between the user equilibrium and the system optimum minimal. Perturbations in real world scenarios are almost guaranteed to happen. It is then useful to know how much perturbation really affects the equilibrium calculated for an instance. In other words, how much it can alter the difference between the user equilibrium and the system optimum. If it can completely alter this relationship, much of the current knowledge of traffic assignment can't be used without adjusting for perturbation.

In our context, we perturb TAP instances by adding some random noise to the latency functions. Our perturbations thus reflect fluctuations in the travel times of the edges. More specifically, suppose that we are given an instance $I = (G, d, l)$ of TAP. We then define *perturbed latency functions* $\bar{l}$ as follows.

Definition 6.1 (Perturbed Latency Functions)**.** *Given an instance $I = (G, d, l)$, then perturbed latency functions are defined as:*

$$\forall e \in E: \quad \bar{l}_e = (1 + \varepsilon_e) l_e, \quad \varepsilon_e \xleftarrow{i.u.r} [0, \sigma]. \tag{6.1}$$

Note that ε_e is chosen independently uniformly at random out of the range $[0, \sigma]$ for every edge $e \in E$. We can then define the Smoothed Price of Anarchy of the Traffic Assignment Problem as follows.

Definition 6.2 (Smoothed Price of Anarchy of I in TAP)**.** *Let f_I and f_I^* denote a Wardrop flow and an optimal flow, respectively, for a given instance $I = (G, d, l)$. The Smoothed Price of Anarchy of I is then defined as*

$$\mathit{SPoA}(I, \sigma) = \frac{\mathbf{E}[\bar{c}(f_{\bar{I}})]}{\mathbf{E}[\bar{c}(f^*_{\bar{I}})]},$$

where $\bar{c}$ refers to the total cost with respect to the perturbed latency functions, i.e., $\bar{c}(f) = \sum_{e \in E} \bar{l}_e(f_e) f_e$ for a given flow f.

As in Definition 4.9, the expectation is taken over all instances $\bar{I}$ that are obtainable from I by perturbations. The Smoothed Price of Anarchy in with respect to a game can then be defined.

Definition 6.3 (Smoothed Price of Anarchy of TAP)**.** *Let $\mathcal{G}$ be the set of possible instances for the Traffic Assignment Problem. Furthermore, let $I \in \mathcal{G}$ be an instance of TAP.*

The Smoothed Price of Anarchy of $\mathcal{G}$ is then defined as

$$\mathit{SPoA}_{TAP}(\sigma) = \max_{I \in \mathcal{G}} \mathit{SPoA}(I, \sigma)\ .$$

Clearly, other smoothing models are conceivable as well. However, here we have chosen the one above because of its good trade-off between simplicity and relevance. Note that a consequence of our relative perturbation model is that the effect of random perturbations is more severe on edges that are sensitive to variations in traffic rate while it is less severe on edges which are rather insensitive to changes in traffic rate.

Note that for our real-world instances, whose latency functions are of the form indicated in Equation (3.1), the above perturbation is equivalent to substituting the free-flow travel time t_e with $(1 + \varepsilon)t_e$.

6.1 Lower Bounds

In this section we present lower bounds for the Smoothed Price of Anarchy for the Traffic Assignment Problem for both linear and polynomial latency functions. We make use of the fact that the Pigou instances, described in Section 3.2, are worst case instances to the unperturbed Price of Anarchy of TAP. We show that the lower bound for the polynomial latency functions case is actually very close to the worst case PoA for TAP, being bound asymptotically by the same function.

6.1.1 Linear Latencies

For the lower bounds on the Smoothed Price of Anarchy with linear latency functions, we use Pigou instances with linear latency functions. We derive exact bounds on this instance using the perturbation model seen in Definition 6.1.

Theorem 6.1. *The Smoothed Price of Anarchy of the Pigou instance with linear latency functions and perturbation magnitude σ is*

$$\mathit{SPoA}(I, \sigma) = \begin{cases} \dfrac{4\sigma(3+\sigma)}{6\sigma(2+\sigma) - (3+\sigma(3+\sigma))\ln[1+\sigma]} & \text{for } 0 \le \sigma \le 1 \\[2ex] \dfrac{24\sigma^2(3+\sigma)}{5+\sigma(3+25\sigma(3+\sigma)) - 6(1+\sigma)^3\ln[2] + 6\ln[1+\sigma]} & \text{for } \sigma > 1 \end{cases}$$

Proof of Theorem 6.1. Let I be the original Pigou instance with two vertices s and t, a demand of 1 unit of flow and latency functions $l_1(x) = x$ for e_1 and $l_2(x) = 1$ for e_2 as introduced in Section 3.2.1.1. We perturb the latency functions of this instance according to the perturbation model described in Definition 6.1. Figure 6.1 shows the perturbed instance with latency functions

$$l_1(x) = (1 + \varepsilon_1)x \quad \text{and} \quad l_2(x) = 1 + \varepsilon_2,$$

where $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$ are random variables.

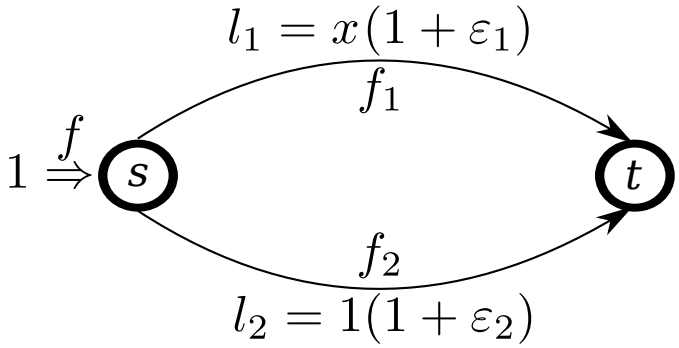

Figure 6.1: Pigou instance with linear latency functions and perturbations $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$.

The flow at a user equilibrium will use the edge e_1 so long as its latency remains lower than e_2. Thus

$$f_1 \leq \frac{1 + \varepsilon_2}{1 + \varepsilon_1}.$$

The flow f_1 will be one if $\varepsilon_2 \geq \varepsilon_1$, since that is our total demand. Therefore, for this case $c(f) = 1 + \varepsilon_1$.

If $\varepsilon_2 \leq \varepsilon_1$, the flow f_1 will be $\frac{1+\varepsilon_2}{1+\varepsilon_1}$, and thus

$$c(f) = \frac{1 + \varepsilon_2}{1 + \varepsilon_1}(1 + \varepsilon_1)\frac{1 + \varepsilon_2}{1 + \varepsilon_1} + \left(1 - \frac{1 + \varepsilon_2}{1 + \varepsilon_1}\right)(1 + \varepsilon_2) = 1 + \varepsilon_2.$$

In order to determine $\mathbf{E}[c(f)]$ for $\varepsilon_1, \varepsilon_2$ chosen uniformly at random from $[0, \sigma]$, we need to solve the double integral combining the two possible total costs divided by the combined probability density function $\frac{1}{\sigma^2}$.

$$\begin{aligned}\mathbf{E}[c(f)] &= \int_0^{\sigma}\int_0^{\varepsilon_2} \frac{1 + \varepsilon_1}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 + \int_0^{\sigma}\int_{\varepsilon_2}^{\sigma} \frac{1 + \varepsilon_2}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 \\ &= \frac{3 + \sigma}{6} + \frac{3 + \sigma}{6} = 1 + \frac{\sigma}{3}\end{aligned}$$

To compute the optimum flow of the system, we exploit the fact that an optimal flow is a Wardrop flow with respect to the marginal cost functions $l_1^*(x) = (p + 1)(\varepsilon_1 + 1)x^p$ and $l_2^*(x) = 1 + \varepsilon_2$. Then

$$f_1^* \leq \frac{1 + \varepsilon_2}{2(1 + \varepsilon_1)} \quad \text{and} \quad f_2^* \geq 1 - \frac{1 + \varepsilon_2}{2(1 + \varepsilon_1)}.$$

Note that both f_1^* and f_2^* must be in the interval $[0, 1]$. If $\sigma \leq 1$, then f_1^* is $\frac{1+\varepsilon_2}{2(1+\varepsilon_1)}$. If that is the case, then

$$c(f^*) = \frac{1+\varepsilon_2}{2(1+\varepsilon_1)}(1+\varepsilon_1)\frac{1+\varepsilon_2}{2(1+\varepsilon_1)} + \left(1 - \frac{1+\varepsilon_2}{2(1+\varepsilon_1)}\right)(1+\varepsilon_2) = (1+\varepsilon_2) - \frac{(1+\varepsilon_2)^2}{4(1+\varepsilon_1)}. \tag{6.2}$$

The expectation over the random choices $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$, with $\sigma \leq 1$, is therefore

$$\begin{aligned}\mathbf{E}[c(f^*)] &= \int_0^\sigma \int_0^\sigma \frac{(1+\varepsilon_2)}{\sigma^2} - \frac{(1+\varepsilon_2)^2}{4\sigma^2(1+\varepsilon_1)}\ d\varepsilon_1 d\varepsilon_2 \\ &= \frac{6\sigma(2+\sigma) - (3+\sigma(3+\sigma))\ln[1+\sigma]}{12\sigma}\end{aligned}$$

Thus, for $\sigma \in [0, 1]$,

$$\text{SPoA}(I, \sigma) = \frac{4\sigma(3+\sigma)}{6\sigma(2+\sigma) - (3+\sigma(3+\sigma))\ln[1+\sigma]}, \quad \text{for } 0 \leq \sigma \leq 1.$$

For $\sigma > 1$, there are two cases: if $\varepsilon_1 \leq \frac{\varepsilon_2 - 1}{2}$, then the flow f_1^* is one, and $c(f^*) = 1 + \varepsilon_1$. Otherwise, if $\varepsilon_1 > \frac{\varepsilon_2 - 1}{2}$, then f_1^* is the same one used in equation 6.2 and thus so is $c(f^*)$.

Taking the expectation over the random choices $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$ for these ranges with different total cost functions, we obtain

$$\begin{aligned}\mathbf{E}[c(f^*)] &= \int_0^1 \int_0^\sigma \frac{(1+\varepsilon_2)}{\sigma^2} - \frac{(1+\varepsilon_2)^2}{4\sigma^2(1+\varepsilon_1)}\ d\varepsilon_1 d\varepsilon_2 + \int_1^\sigma \int_0^{\frac{\varepsilon_2-1}{2}} \frac{(1+\varepsilon_1)}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 \\ &\quad + \int_1^\sigma \int_{\frac{\varepsilon_2-1}{2}}^\sigma \frac{(1+\varepsilon_2)}{\sigma^2} - \frac{(1+\varepsilon_2)^2}{4\sigma^2(1+\varepsilon_1)}\ d\varepsilon_1 d\varepsilon_2 \\ &= \frac{5 + \sigma(3 + 25\sigma(3+\sigma)) - 2(1+\sigma)^3\ln[8] + 6\ln[1+\sigma]}{72\sigma^2}\end{aligned}$$

Thus, the SPoAfor this case is

$$\text{SPoA}(I, \sigma) = \frac{24\sigma^2(3+\sigma)}{5 + \sigma(3 + 25\sigma(3+\sigma)) - 6(1+\sigma)^3\ln[2] + 6\ln[1+\sigma]}, \quad \text{for } \sigma \geq 1.$$

Combining both equations for both ranges of σ, we have that

$$\text{SPoA}(I, \sigma) = \begin{cases} \dfrac{4\sigma(3+\sigma)}{6\sigma(2+\sigma) - (3+\sigma(3+\sigma))\ln[1+\sigma]} & \text{for } 0 \leq \sigma \leq 1 \\ \dfrac{24\sigma^2(3+\sigma)}{5 + \sigma(3 + 25\sigma(3+\sigma)) - 6(1+\sigma)^3\ln[2] + 6\ln[1+\sigma]} & \text{for } \sigma > 1 \end{cases} \tag{6.3}$$

□

Figure 6.2 shows the SPoA bound for Pigou as a function of σ, for $\sigma \leq 1$, while Figure 6.3 illustrates the SPoA bound for much bigger values of σ. While the decrease is fast for big values of σ, it is always larger than 1.15, staying almost in the middle of the best possible ratio of one and the worst possible ratio of $4/3$.

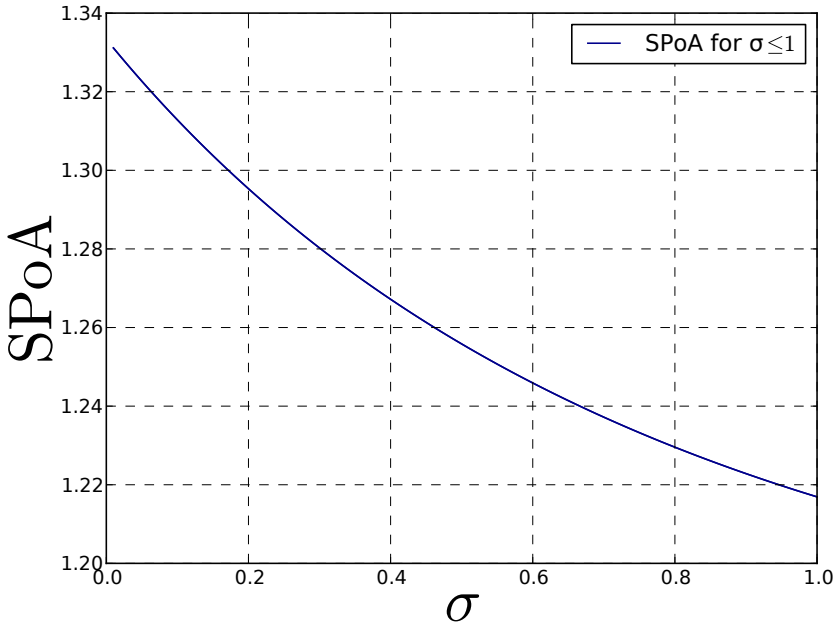

Figure 6.2: Smoothed Price of Anarchy of Pigou instances with linear latency functions for $\sigma \leq 1$.

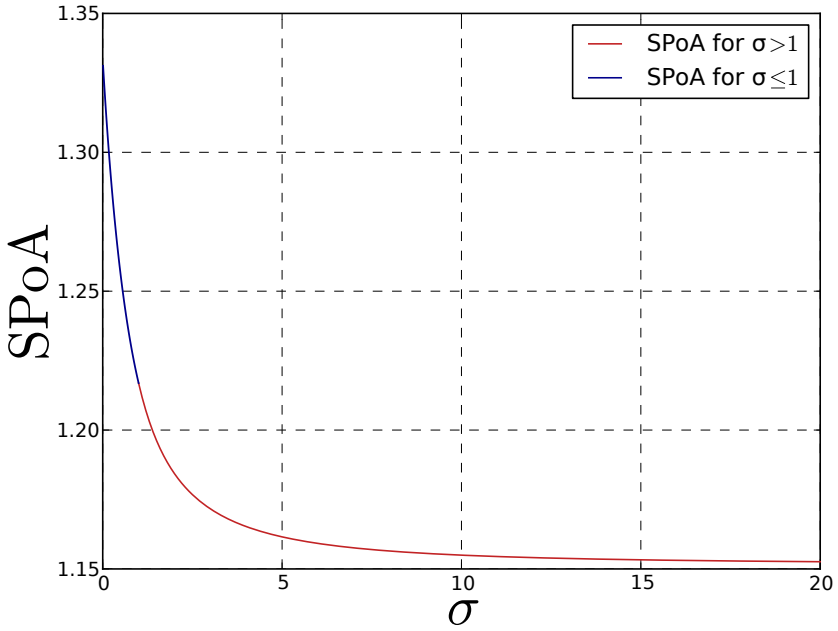

Figure 6.3: Smoothed Price of Anarchy of Pigou instances with linear latency functions for big perturbation magnitudes σ.

Note that a σ of one means that the latencies can double their costs, and greater magnitudes makes the smoothed price of anarchy behave more like the average price of anarchy would. For a σ of less than one, the drop on the PoA ratio is significant, with a SPoA of 1.21692 for $\sigma = 1$.

This method of finding bounds can be extended for other types of latency functions, and is extended for polynomial latency functions in the next section.

6.1.2 Polynomial Latencies

We consider Pigou instances with polynomial latency functions. We will derive exact bounds on the Smoothed Price of Anarchy under our random perturbations for these instances. These bounds also establish a lower bound on the Smoothed Price of Anarchy for general multi-commodity instances. We leave it as an important open problem to derive bounds on the Smoothed Price of Anarchy for multi-commodity instances and polynomial latency functions.

Theorem 6.2. *The Smoothed Price of Anarchy of the Pigou instance with polynomial latency functions of degree p is*

$$SPoA(I,\sigma) = \frac{3+\sigma}{3+\frac{3\sigma}{2}+\frac{3p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)(-1+p^2)\sigma^2}} \tag{6.4}$$

Proof of Theorem 6.2. Let I be the original Pigou instance with two vertices s and t, a demand of 1 unit of flow and latency functions $l_1(x) = x^p$ for e_1 and $l_2(x) = 1$ for e_2 as introduced in Section 3.2.1.2. After perturbing the latency functions of I as described above, we obtain the instance depicted in Figure 6.4 with latency functions

$$l_1(x) = (1+\varepsilon_1)x^p \quad \text{and} \quad l_2(x) = 1+\varepsilon_2,$$

where $\varepsilon_1, \varepsilon_2 \in [0,\sigma]$ are random variables.

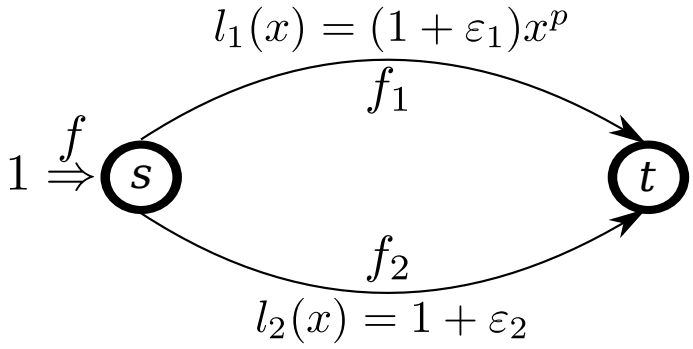

Figure 6.4: Pigou instance with polynomial latency functions and perturbations $\varepsilon_1, \varepsilon_2 \in [0,\sigma]$.

We first determine a Wardrop flow. With the addition of perturbation, edge e_1 is used as long as its latency is lower than the latency of e_2. Therefore

$$f_1 \leq \sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}.$$

Since this can be greater than our maximum flow,

$$f_1 = \min\left(\sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}, 1\right) \quad \text{and} \quad f_2 = 1 - \min\left(\sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}, 1\right).$$

The flow f_1 is going to be 1 as long as $\varepsilon_2 \geq \varepsilon_1$. If this is the case, then $c(f) = 1 + \varepsilon_1$. If $\varepsilon_2 \leq \varepsilon_1$, then

$$c(f) = \sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}(1+\varepsilon_1)\left(\sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}\right)^p + \left(1 - \sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}\right)(1+\varepsilon_2) = 1+\varepsilon_2.$$

In order to determine $\mathbf{E}[c(f)]$ for $\varepsilon_1, \varepsilon_2$ chosen uniformly at random from $[0, \sigma]$, we need to solve the following double integral. (Note that the combined probability density function is $\frac{1}{\sigma^2}$).

$$\begin{aligned}\mathbf{E}[c(f)] &= \int_0^\sigma \int_0^{\varepsilon_2} \frac{1+\varepsilon_1}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 + \int_0^\sigma \int_{\varepsilon_2}^\sigma \frac{1+\varepsilon_2}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 \\ &= \frac{3+\sigma}{6} + \frac{3+\sigma}{6} = 1 + \frac{\sigma}{3}\end{aligned}$$

In order to compute the system optimum flow, we exploit the fact that an optimal flow is a Wardrop flow with respect to the marginal cost functions $l_1^*(x) = (p+1)(\varepsilon_1+1)x^p$ and $l_2^*(x) = 1+\varepsilon_2$. Then

$$f_1^* = \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}} \quad \text{and} \quad f_2^* = 1 - \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}.$$

Note that σ must be at most p for the optimum flow f_1^* to remain below the maximum flow. The cost of f^* is

$$\begin{aligned}c(f^*) &= \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}(1+\varepsilon_1)\left(\sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}\right)^p + \left(1 - \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}\right)(1+\varepsilon_2) \\ &= 1+\varepsilon_2 - p(1+p)^{-1-\frac{1}{p}}(1+\varepsilon_1)^{-\frac{1}{p}}(1+\varepsilon_2)^{1+\frac{1}{p}}.\end{aligned}$$

Taking the expectation over the random choices $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$, we obtain

$$\begin{aligned}\mathbf{E}[c(f^*)] &= \int_0^\sigma \int_0^\sigma \frac{1+\varepsilon_2 - p(1+p)^{-1-\frac{1}{p}}(1+\varepsilon_1)^{-\frac{1}{p}}(1+\varepsilon_2)^{1+\frac{1}{p}}}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 \\ &= 1 + \frac{\sigma}{2} + \frac{p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)\left(-1+p^2\right)\sigma^2}\end{aligned}$$

Thus

$$\mathrm{SPoA}(I, \sigma) = \frac{3+\sigma}{3 + \frac{3\sigma}{2} + \frac{3p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)(-1+p^2)\sigma^2}} \tag{6.5}$$

□

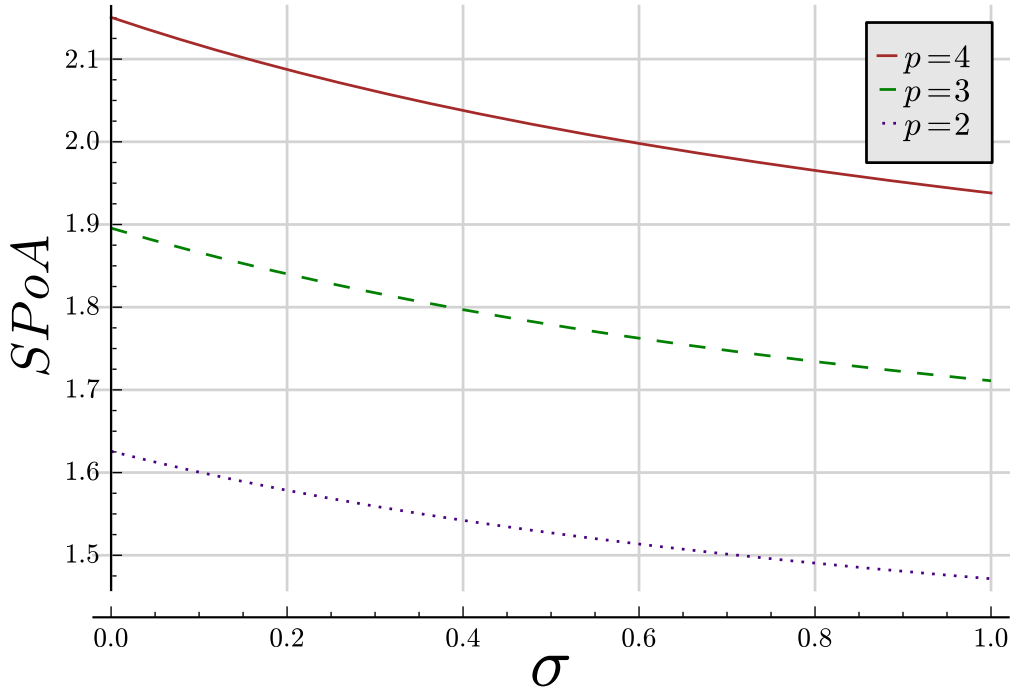

Figure 6.5: Smoothed Price of Anarchy of Pigou instances with polynomial latency functions as a function of σ for $p = 2, 3, 4$.

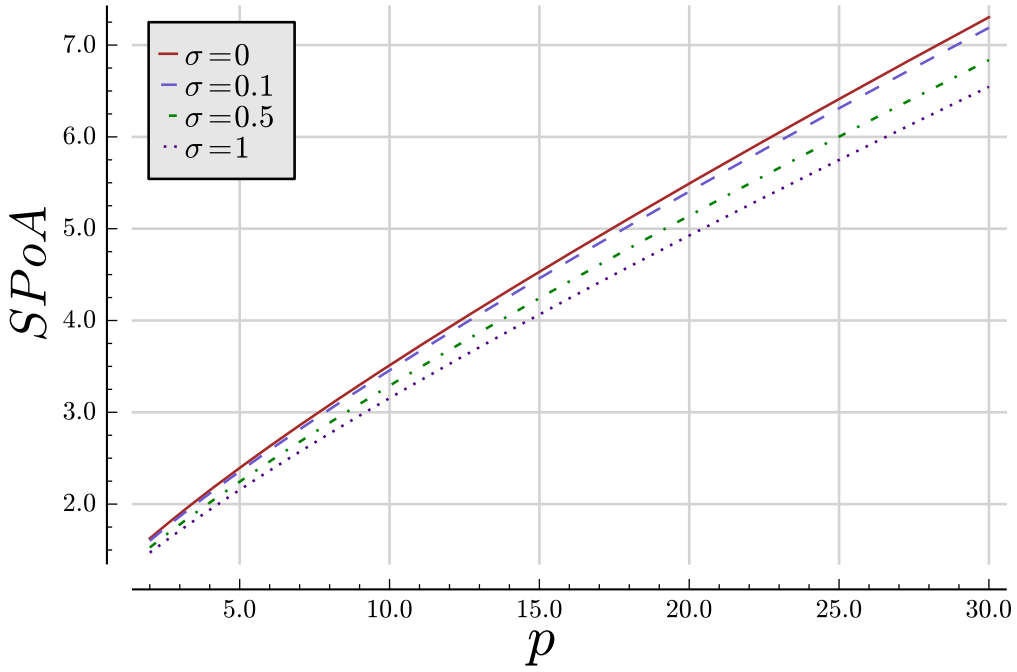

Figure 6.6: Smoothed Price of Anarchy of Pigou instances with polynomial latency functions as a function of p for $\sigma = 0, 0.1, 0.5$ and 1.

Figure 6.5 illustrates the SPoA bound for Pigou instances for $p = 2, 3, 4$ as a function of σ, while Figure 6.6 shows the SPoA bound as a function of p, with fixed $\sigma = 0, 0.1, 0.5$ and 1.

Recall that the Pigou instance is the worst-case instance for the Price of Anarchy. Clearly, the Smoothed Price of Anarchy either stays the same or improves (i.e., decreases). As our bound shows, it improves but the decrease is rather low. Even for perturbations of the magnitude $\sigma = 1$, the decrease is about 10% only. Note that in this case we may double the latency functions. With increasing degree, this decrease becomes more significant. If we restrict σ to be less than or equal to $\frac{1}{p}$, which can be seen in Figure 6.7, then the Smoothed Price of Anarchy asymptotically remains $\Theta(\frac{p}{\ln p})$ as in the deterministic case.

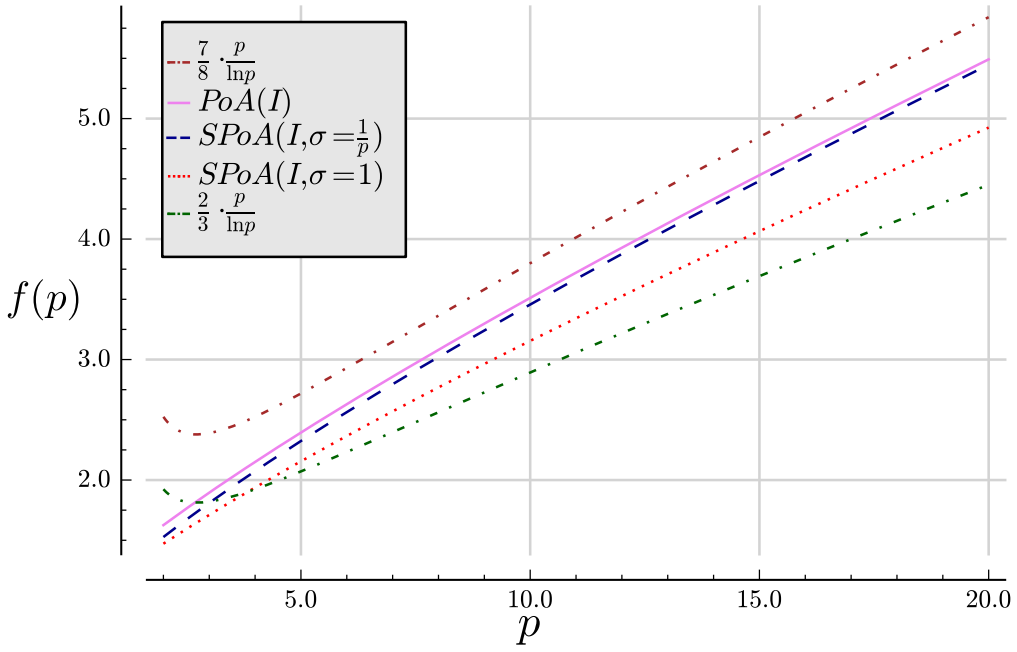

Figure 6.7: Smoothed Price of Anarchy of Pigou instances shown to remain $\Theta(\frac{p}{\ln p})$

6.2 Experimental Results in Benchmark Instances

6.2.1 Experimental Setup

In order to evaluate if real world instances behave in a different manner in relation to the worst case instances for the Price of Anarchy, we tested a few benchmark instances freely available for academic research from the Transportation Network Test Problems (BAR-GERA, 2011).

In these instances, the latencies follow the U.S. Bureau of Public Roads definition, shown in (3.1), with $\alpha = 0.15$ and $\beta = 4$. We chose a few instances to compare and perturb, with both big and small instances evaluated. The instances details can be seen in Table 6.1.

To find the user equilibrium and system optimum, we used the Frank-Wolfe algorithm (FRANK; WOLFE, 1956). The algorithm was implemented in C++ and compiled in 64 bit gcc version $4.4.5$, in a Linux kernel version $2.6.35$. The machine used for the tests has an Intel® Core™ i7 CPU with 4 cores, with 12 GB of RAM memory.

Table 6.1: List of benchmark instances used in the experiments.

Instance Name	|V|	|E|	|K|	|E|·|K|
Sioux Falls	24	76	528	40,128
Friedrichshain	224	523	506	264,638
Chicago Sketch	933	2,950	83,113	245,183,350
Berlin Center	12,100	19,570	49,689	972,413,730

6.2.2 Results

We perturbed the instances with $\sigma \in \{10^{-9}, 10^{-8}, ..., 10^{-2}\}$. We also evaluated instances with a greater perturbation, with $\sigma \in \{0.1, 0.2, ..., 0.9\}$. The algorithm was stopped when it reached a relative gap less than of 10^{-5}, except the Sioux Falls instance which the minimum relative gap was set to 10^{-6}. For each perturbation magnitude σ_i, 10 runs were executed and the average value was considered. In Table 6.2, the Price of Anarchy of the instances are presented. Also, for the perturbed instances, for among all averages for the different values of σ, the mean, the standard variation, the minimum and maximum values are presented.

Table 6.2: Original PoA and perturbed PoA related measures found for each instance.

Instance Name	POA	Mean	Minimum	Maximum	Std. Deviation
Sioux Falls	1.039682	1.039689	1.039676	1.039707	8.049609×10^{-6}
Friedrichshain	1.086374	1.086422	1.086345	1.086599	4.996005×10^{-5}
Chicago Sketch	1.023569	1.023567	1.023561	1.023572	2.137639×10^{-6}
Berlin Center	1.006141	1.006142	1.006133	1.006155	3.831177×10^{-6}

In Table 6.3 we can see the average time that the perturbed instances executed. It is clear that for these instances the perturbations did not significantly alter their execution time. Note that the Sioux Falls instance takes longer than the Friedrichshain instance due to the smaller relative gap used on the Sioux Falls instance.

Table 6.3: Execution time found for original instances and the average for the perturbed instances, in seconds. Here, "UE" refers to user equilibrium and "SO" to system optimum.

Instance Name	UE time	SO time	Average UE time	Average SO time
Sioux Falls	1.58	1.6	1.59	1.61
Friedrichshain	0.43	0.92	0.43	0.96
Chicago Sketch	27.99	36.89	27.34	36.56
Berlin Center	233.91	360.07	220.97	360.64

In Figures 6.8, 6.9, 6.10 and 6.11, we present graphics for each tested instance, with $\sigma \in \{0.1, ..., 0.9\}$ in the top and $\sigma \in \{10^{-9}, ..., 10^{-2}\}$ on the bottom for each instance.

Both the smoothed and the original Price of Anarchy are close to one for all tested instances, which gives some empirical evidence that the worst case is not so likely to occur in real world instances. Furthermore, when we look at the Smoothed Price of Anarchy for all instances, as shown in Figures 6.8, 6.9, 6.10, 6.11, we notice that even for relatively large σ, the Smoothed Price of Anarchy remains almost constant and very close to the original Price of Anarchy.

The small trend that the Smoothed Price of Anarchy tends to follows on these instances seems to be related more with the particular instance than with a more general

rule. This can be seen on the difference between the Friedrichshain instance and the Chicago instance, while in the Berlin instance it appears to remains constant.

The fact that the Smoothed Price of Anarchy does not drop significantly from the original Price of Anarchy, allied with these experimental results, shows that while perturbation does occur frequently in real world scenarios, it does not have a great influence on the actual distance from users equilibrium to the overall system optimum.

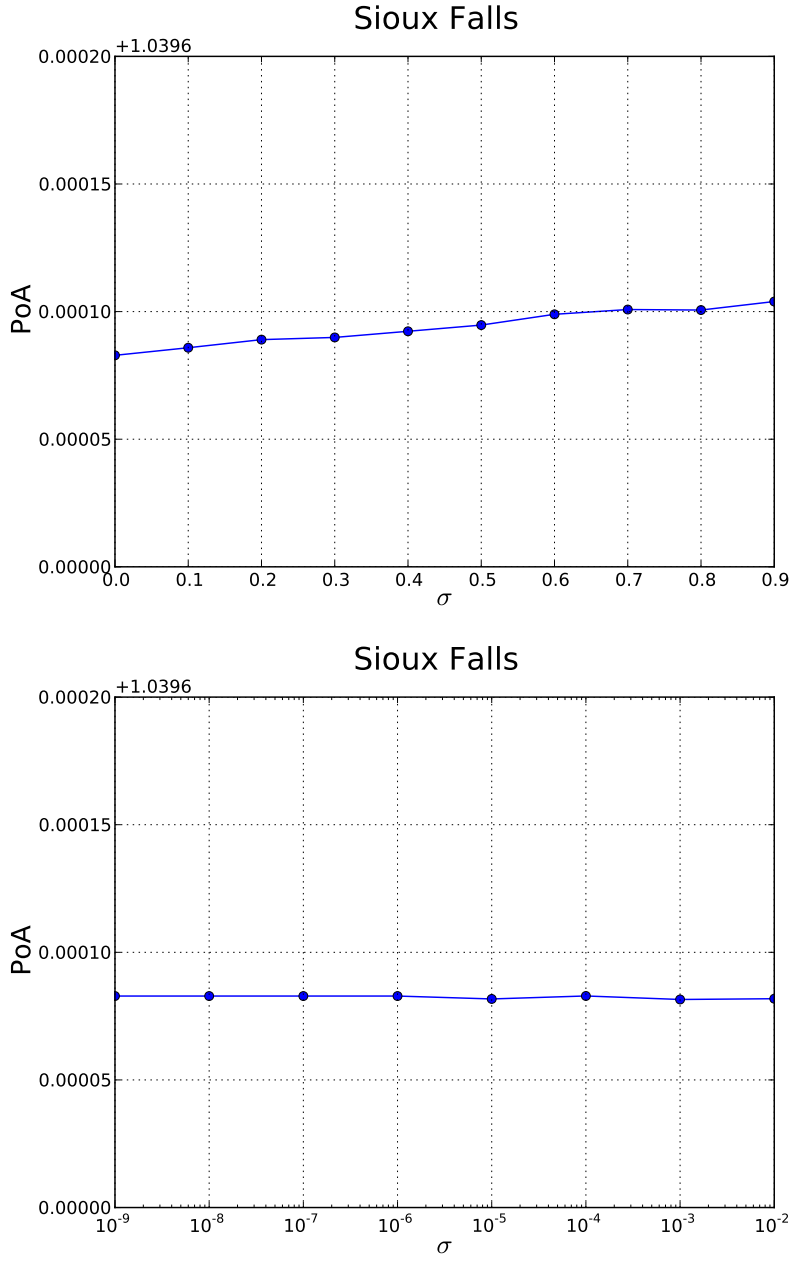

Figure 6.8: Experimental Smoothed Price of Anarchy for the Sioux Falls instance, with $\sigma \in \{0.1, ..., 0.9\}$ in the top and $\sigma \in \{10^{-9}, ..., 10^{-2}\}$ in the bottom.

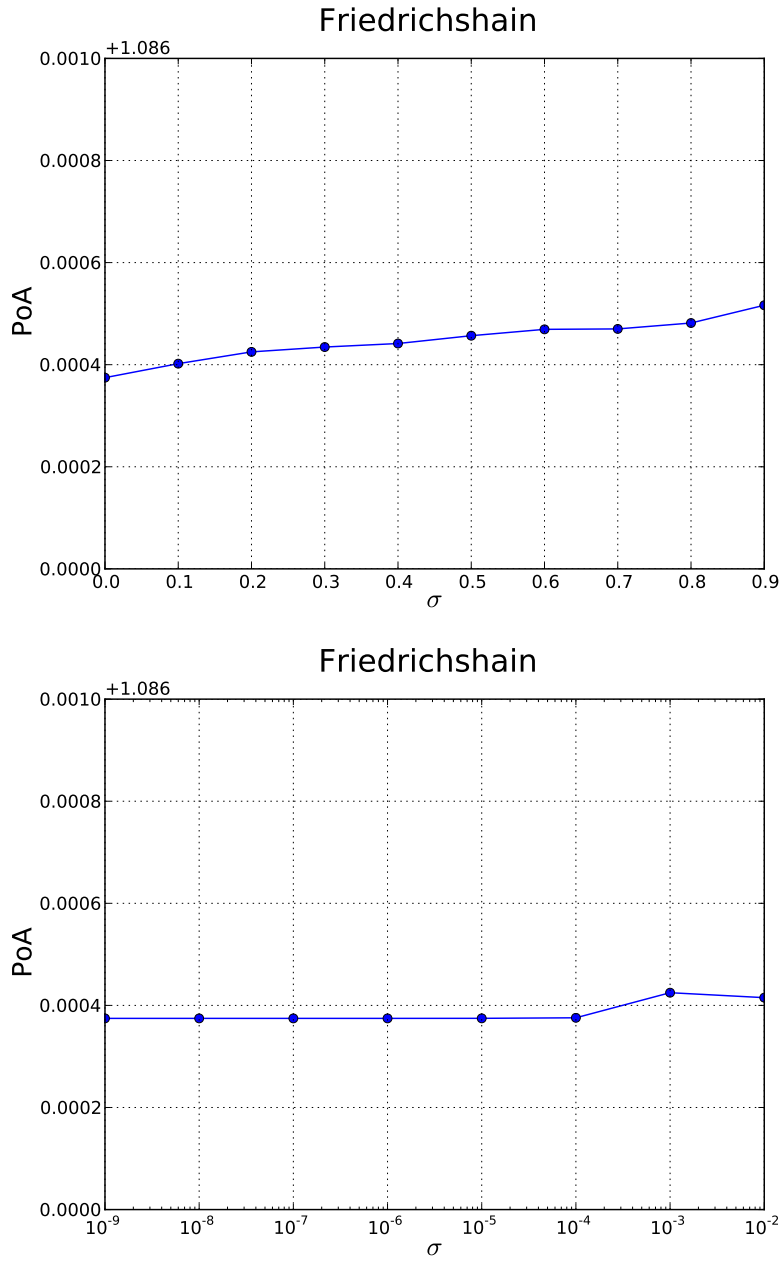

Figure 6.9: Experimental Smoothed Price of Anarchy for the Friedrichshain instance, with $\sigma \in \{0.1, ..., 0.9\}$ in the top and $\sigma \in \{10^{-9}, ..., 10^{-2}\}$ in the bottom.

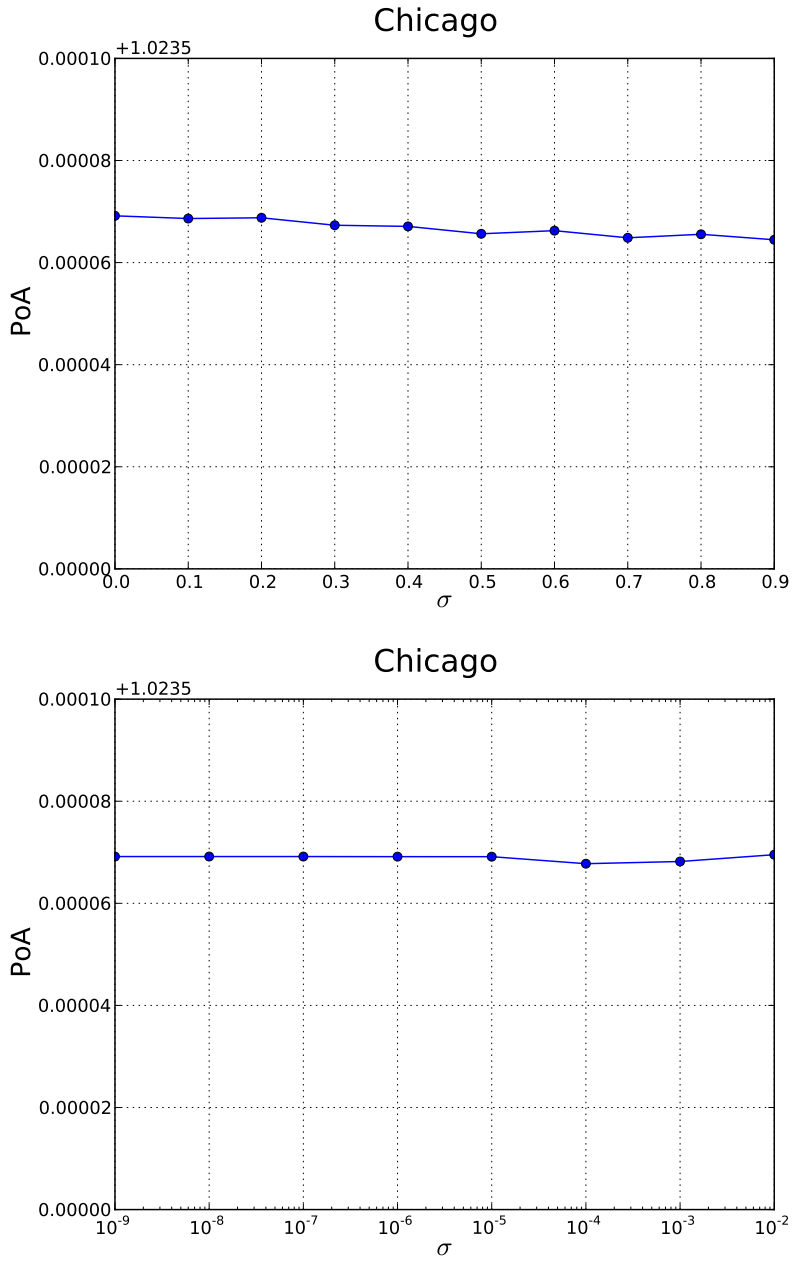

Figure 6.10: Experimental Smoothed Price of Anarchy for the Chicago sketch instance, with $\sigma \in \{0.1, ..., 0.9\}$ in the top and $\sigma \in \{10^{-9}, ..., 10^{-2}\}$ in the bottom.

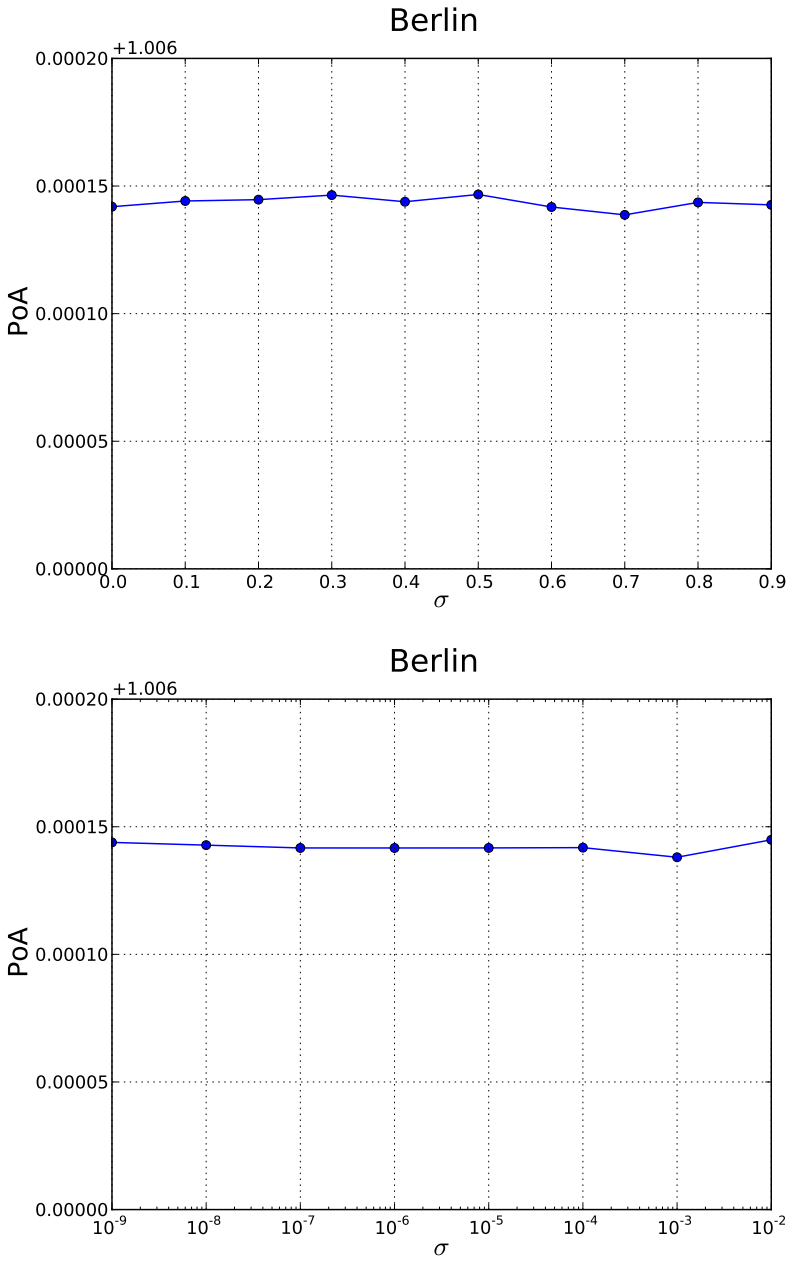

Figure 6.11: Experimental Smoothed Price of Anarchy for the Berlin Center instance, with $\sigma \in \{0.1, ..., 0.9\}$ in the top and $\sigma \in \{10^{-9}, ..., 10^{-2}\}$ in the bottom.

7 CONCLUSIONS

In this thesis we studied two important subjects in game theory with respect to the effect of perturbation. For bimatrix games, we analyzed known worst case instances for the Lemke-Howson algorithm, and displayed evidence that perturbation can greatly affect those instances.

A bimatrix game is a game of two players in normal form. The game is specified by two $m \times n$ matrices of pure strategies, with the matrices' values being the payoffs of the players. The Lemke–Howson algorithm is a complementary pivoting algorithm which is widely used for finding a Nash equilibrium of a bimatrix game. These games can be subject to a small amount of perturbation, specially if the game is based around data and situations from real-world scenarios.

One important complexity model that can analyze these cases with perturbation is called smoothed analysis (SPIELMAN; TENG, 2004). For bimatrix games, Chen et al. proved (CHEN; DENG, 2006), under some assumptions, that there exists at least one worst case instance that remains exponential even with small perturbations. In the proof, however, no explicit instance construction is provided. It also does not contain an assessment of the likelihood of these hard instances in real cases, leaving room to experimental exploration.

In this thesis we studied the effect of perturbation on the only known (to the best of our knowledge) class of games that take exponential time for the Lemke-Howson algorithm to solve. In (SAVANI; STENGEL, 2004), the author presented a class of bimatrix games for which the LH algorithm, no matter which starting pivot is chosen, uses a path that is exponential on the game size. Their construction relies on dual cyclic polytopes from polytope theory.

We perturbed the instances described in (SAVANI; STENGEL, 2004) according to smoothed analysis, using a uniform perturbation model. Even for polynomially small perturbation magnitude σ values such as $\frac{1}{n^4}$, our experiments give reason to believe that the number of steps the LH algorithm takes dropped to polynomial running times. This indicates that the worst case instance for the smoothed complexity of bimatrix games has not been discovered yet, and does not happen often on practice. Note that these conclusions are made based on a relatively small instance size of 20, since it was not feasible to run experiments for larger sizes. Also relevant is the fact that to simulate these very small perturbations, an exact implementation of the LH algorithm was built (RODRIGUES, 2011).

We extended the idea of smoothed analysis to analyze the Price of Anarchy. The Price of Anarchy (KOUTSOUPIAS; PAPADIMITRIOU, 1999) is a measure in game theory that compares the optimal social welfare of a game with the worst possible Nash equilibrium. We proposed a measure of perturbation of the Price of Anarchy based on the

smoothed analysis for algorithms, the Smoothed Price of Anarchy.

We also analyzed the effects of perturbation on a network problem that can be modeled in game theory. The Traffic Assignment Problem is concerned with the choice of routes in a road network given a set of users with an origin and a destination. It is of extreme importance for traffic planning, and, in real world cases, perturbation occurs frequently. Therefore, it is useful to have a notion of how much can this perturbation affects its instances. It is of particular interest how the Price of Anarchy is affected in these situations, since the goal of road network planning is usually to approximate the user equilibrium to the system optimum.

Using the Smoothed Price of Anarchy, we propose a perturbation model for the Traffic Assignment Problem. We give a lower bound for the Smoothed Price of Anarchy using this perturbation model that is of the same order as the worst case Price of Anarchy for polynomial latencies. A bound on the SPoA for linear latency functions is also presented.

Finally, we show experimentally that the effects of perturbation on the Price of Anarchy of real world instances, at least for the known instance benchmarks in the literature present in the Transportation Network Test Problems, are severely limited and show no general trend.

This thesis is also the basis for the paper (RODRIGUES et al., 2011), entitled "On the Smoothed Price of Anarchy of the Traffic Assignment Problem", a joint work with Guido Schäfer, Luciana Buriol and Marcus Ritt.

7.1 Discussion and Future Work

For bimatrix games, a very important extension is to find an instance where the proof by Chen et al. holds true, i.e., to find an exponential time instance where small perturbations does not make the running time drop to polynomial time.

One possible approach is as follows. In the paper by Chen et al. (CHEN; DENG; TENG, 2006), in order to prove that no ϵ-approximation for two player NASH exists, several reductions from other PPAD-complete problems are performed. Perhaps if one was to take a worst case instance from these problems and perform the transformations explained in the paper a worst case instance of such kind could appear.

The problem with this approach is that there is no guarantees that this new instance behaves in the desired manner, since the original paper does not mention an instance.

7.1.1 Hypercube based instances

Another option is to create an instance from scratch that behaves in that manner. It is possible to make instances by taking different polytopes as the origin. An interesting experiment is to create an instance that represents the Klee-Minty polytope (KLEE; MINTY, 1972), which is a well known worst case instance for the simplex algorithm, and to apply the already developed transformations to generate both matrix A as well as B required for the LH algorithm.

The path used by the Klee-Minty's instances are exponential in the dimension of the polytope being searched. In Figure 7.1 we give the path for the three dimensional case, using a cube to represent the instance. In this representation, each hyperplane is an inequality of the problem, with the solution always being in one of the polytope vertices.

To generate this instance, one would have to start with a hypercube for one of the polytopes and attempt to generate the other polytope so that at least one of the chosen starting pivots maintains its exponentially long path.

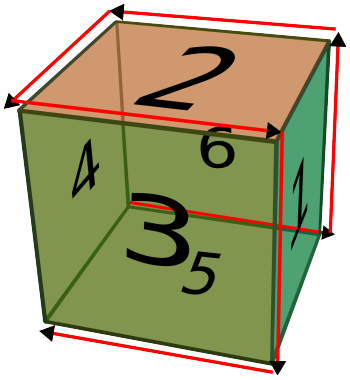

Figure 7.1: Klee-Minty's instances behavior in the three dimensional case.

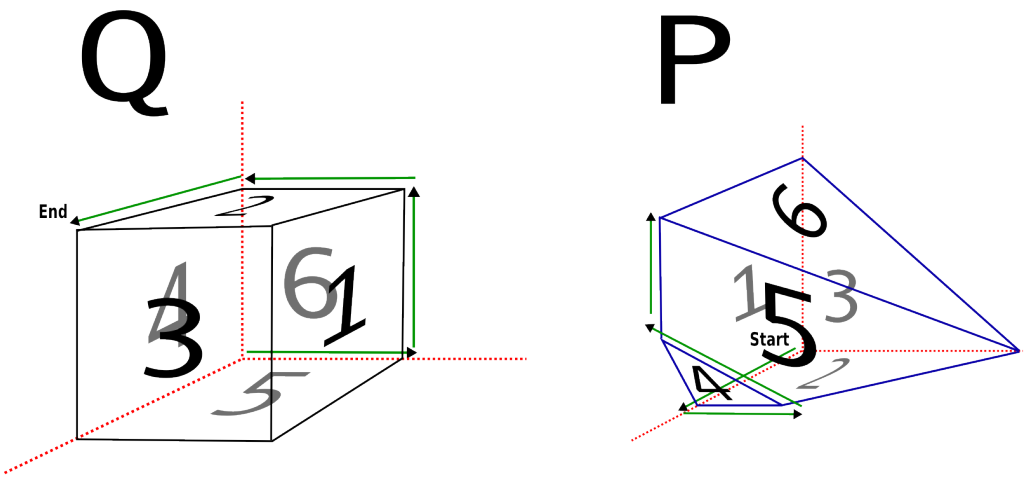

Figure 7.2: Lemke-Howson exponential instance for the three dimensional case.

Table 7.1: Normal form game for Figure 7.2.

A \ B	3	4	5
1	$1, 0$	$0, 1$	$0, 1$
2	$0, -2$	$1, 0$	$0, 1$
3	$0, 2$	$0, 1$	$1, 0$

One possible polytope which behaves this way is the one in Figure 7.2 and presented in Table 7.1. This polytope has been found solving a system of inequalities that guarantees the Lemke-Howson path to behave exactly as the one found on the Klee-Minty's instances for simplex up to about half of path, when the equilibrium is found. Since there are two polytopes in the Lemke-Howson algorithm, instead of just one as in simplex, the number of steps remains the same.

The challenge is to find a way to grow this instance to higher dimensions in a feasible way, preferably with a polynomial time algorithm. It is unclear if this approach is possible, and how to handle the possible starting pivot choices, since this instance only has an exponential path in relation to size n for a few selected pivots.

7.1.2 Smoothed Price of Anarchy in different contexts

The idea of analyzing the Price of Anarchy with respect to perturbation is interesting to a number of games other than the one studied in this thesis. For example, this could be done for several other congestion games, such as finite congestion games, potential games and others. For games with Nash equilibria with different total costs, an extension to include a definition of Smoothed Price of Stability is also of interest.

Another area for further research is on perturbation models for the Traffic Assignment Problem. More complex models for relative perturbation are needed to make the theoretical analysis closer to practice, as well as restricting further the type of latency functions allowed. Deriving bounds on the SPoA for additive perturbation models is also important, specially for situations modeled by the Traffic Assignment Problem other than road congestion.

A type of perturbation that is closer to practice and that would be interesting to study is a kind of error propagation model, where one link suffers a somewhat significant perturbation, which spreads to adjacent links on the network. This models more closely a situation where weather and roadblocks affect more vulnerable roads, such as when there is heavy rain in some cities: certain roads remain with their capacities unaltered, while other roads may experience flooding.

Finally, it would be important to find upper bounds on the Smoothed Price of Anarchy for the Traffic Assignment Problem, specially for linear latency functions. An approach to accomplish this is to try to adapt the proof in (ROUGHGARDEN, 2003) to work with our perturbed instances. That would lead to the upper bounds of the SPoA being the same as the ones found in this thesis.

REFERENCES

ARRACHE, S.; OUAFI, R. Accelerating Convergence of the Frank-Wolfe Algorithm for Solving the Traffic Assignment Problem. **IJCSNS**, [S.l.], v.8, n.5, p.181–186, 2008.

BAR-GERA, H. **Transportation Network Test Problems**. Available at `http://www.bgu.ac.il/~bargera/tntp/`. Last accessed on June 2011.

Bureau of Public Roads. **Traffic assignment manual**. Washington, DC: U.S. Department of Commerce, Urban Planning Division, 1964.

CEPPI, S.; GATTI, N.; PATRINI, G.; ROCCO, M. Local search techniques for computing equilibria in two-player general-sum strategic-form games. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS: VOLUME 1 - VOLUME 1, 9., Richland, SC. **Proceedings...** International Foundation for Autonomous Agents and Multiagent Systems, 2010. p.1469–1470. (AAMAS '10).

CHEN, X.; DENG, X. Settling the Complexity of Two-Player Nash Equilibrium. In: ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 47., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.261–272. (FOCS '06).

CHEN, X.; DENG, X.; TENG, S.-H. Computing Nash Equilibria: approximation and smoothed complexity. In: ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 47., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.603–612. (FOCS '06).

CODENOTTI, B.; ROSSI, S. D.; PAGAN, M. An experimental analysis of Lemke-Howson algorithm. **CoRR**, [S.l.], v.abs/0811.3247, 2008.

MURTY, K. G. **Computational complexity of complementary pivot methods**. [S.l.]: Elsevier Science Ltd, 1978. p.61–73.

DANTZIG, G. **Linear Programming and Extensions**. Princeton, New Jersey, USA: Princeton University Press, 1963.

DASKALAKIS, C.; GOLDBERG, P. W.; PAPADIMITRIOU, C. H. The complexity of computing a Nash equilibrium. In: STOC '06: THIRTY-EIGHTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, New York, NY, USA. **Proceedings...** ACM, 2006. p.71–78.

FATHI, Y. Computational complexity of LCPs associated with positive definite symmetric matrices. **Mathematical Programming**, [S.l.], v.17, p.335–344, 1979. 10.1007/BF01588254.

FLORIAN, M. **An Improved Linear Approximation Algorithm for the Network Equilibrium, Packet Switching, Problem**. [S.l.]: Université de Montréal, Centre de recherche sur les transports, 1977.

FRANK, M.; WOLFE, P. An algorithm for quadratic programming. **Naval Research Logistics Quarterly**, [S.l.], v.3, n.1-2, p.95–110, 1956.

FUKUSHIMA, M. A modified Frank-Wolfe algorithm for solving the traffic assignment problem. **Transportation Research Part B: Methodological**, [S.l.], v.18, n.2, p.169 – 177, 1984.

GRANLUND, T. et al. **GNU Multiple Precision Arithmetic Library**. Available at `http://gmplib.org`. Last accessed on July 2011.

GRUNBAUM, B. **Convex Polytopes**. [S.l.]: Springer, 2003.

HEARN, D. W.; RAMANA, M. V. Solving Congestion Toll Pricing Models. **Equilibrium and Advanced Transportation Modeling**, [S.l.], p.109–124, 1998.

KLEE, V.; MINTY, G. J. How good is the simplex algorithm? In: SHISHA, O. (Ed.). **Inequalities**. New York: Academic Press, 1972. v.III, p.159–175.

KOUTSOUPIAS, E.; PAPADIMITRIOU, C. Worst-case equilibria. In: ANNUAL SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE, 16., Trier, Germany. **Proceedings...** Springer, 1999. p.404–413.

KUHN, H. W.; TUCKER, A. W. Nonlinear programming. In: SECOND BERKELEY SYMPOSIUM ON MATHEMATICAL STATISTICS AND PROBABILITY, 1950, Berkeley and Los Angeles. **Proceedings...** University of California Press, 1951. p.481–492.

LEMKE, C. E.; J. T. HOWSON, J. Equilibrium Points of Bimatrix Games. **SIAM Journal on Applied Mathematics**, [S.l.], v.12, n.2, p.413–423, 1964.

MARTELLO, S.; TOTH, P. **Knapsack Problems. Algorithms and Computer Implementations**. Chichester: John Wiley and sons, 1990. 296p.

MCKELVEY, R. D.; MCLENNAN, A. M.; TUROCY, T. L. **Gambit**: Software Tools for Game Theory, Version 0.2010.09.01. [S.l.]: The Gambit Project, 2010. Available at `http://www.gambit-project.org`. Last accessed on Dec 2010.

MURTY, K. G. **Linear programming**. New York: John Wiley and Sons Inc., 1983. xix+482p. With a foreword by George B. Dantzig.

NASH, J. Non-Cooperative Games. **The Annals of Mathematics**, [S.l.], v.54, n.2, p.286–295, 1951.

FLORIAN, M.; HEARN, D. **Network Equilibrium Models and Algorithms**. [S.l.]: Elsevier Science Pub Co, 1995. v.8.

PAPADIMITRIOU, C. H. On the complexity of the parity argument and other inefficient proofs of existence. **J. Comput. Syst. Sci.**, Orlando, FL, USA, v.48, n.3, p.498–532, 1994.

RODRIGUES, F. C. **Lemke-Howson Algorithm Implementation**. Available at `http://inf.ufrgs.br/~fcrodrigues/files/lemkehowson.tar.gz`. Last accessed on May 2011.

RODRIGUES, F.; SCHÄFER, G.; BURIOL, L.; RITT, M. On the Smoothed Price of Anarchy of the Traffic Assignment Problem. In: WORKSHOP ON ALGORITHMIC APPROACHES FOR TRANSPORTATION MODELLING, OPTIMIZATION, AND SYSTEMS, 11., Dagstuhl, Germany. **Proceedings...** Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011. p.122–133. (OpenAccess Series in Informatics (OASIcs), v.20).

ROUGHGARDEN, T. The price of anarchy is independent of the network topology. **J. Comput. Syst. Sci.**, Orlando, FL, USA, v.67, p.341–364, September 2003.

ROUGHGARDEN, T.; TARDOS, E. How bad is selfish routing? **Journal of the ACM (JACM)**, [S.l.], v.49, n.2, p.259, 2002.

SAVANI, R. **Finding Nash equilibria of bimatrix games**. 2006. Tese (Doutorado em Ciência da Computação) — London School of Economics and Political Science.

SAVANI, R.; STENGEL, B. V. Exponentially Many Steps for Finding a Nash Equilibrium in a Bimatrix Game. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE (FOCS). **Proceedings...** IEEE Computer Society, 2004. p.258–267.

SPIELMAN, D. A.; TENG, S.-H. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. **Journal of the ACM**, New York, NY, USA, v.51, n.3, p.385–463, May 2004.

STEIN, W. et al. **Sage Mathematics Software (Version 4.3.3)**. [S.l.]: The Sage Development Team, 2010. Available at `http://www.sagemath.org`. Last accessed on Dec 2010.

STENGEL, B. V. New maximal numbers of equilibria in bimatrix games. **Discrete and Computational Geometry**, [S.l.], v.21, p.557–568, 1999.

STENGEL, B. von. Computing equilibria for two-person games. In: HANDBOOK OF GAME THEORY. **Proceedings...** North-Holland, 2002. p.1723–1759.

WARDROP, J. Some theoretical aspects of road traffic research. **Institution of Civil Engineers, Part II**, [S.l.], v.1, n.36, p.352–362, 1952.

ZIEGLER, G. M. **Lectures on Polytopes**. [S.l.]: Springer, 2001.

APPENDIX A ANÁLISE SUAVISADA EM EQUILÍBRIOS NASH E NO PREÇO DA ANARQUIA

RESUMO DA DISSERTAÇÃO EM PORTUGUÊS

A.1 Teoria dos Jogos

Teoria dos jogos é um ramo da matemática com diversas aplicações em áreas como economia, biologia, ciências política, entre muitas outras. Existem também muitas aplicações na área da ciência da computação, como problemas relacionados com a Internet e roteamento. A teoria dos jogos visa formalizar o comportamento em situações em que o benefício das escolhas de um indivíduo depende das escolhas dos outros.

Os jogos estudados são matematicamente definidos como um conjunto de jogadores, um conjunto de estratégias puras e um *payoff* associado a cada combinação de estratégias.

Definition A.1 (Jogo)**.** *Existem n jogadores. Cada jogador i tem um conjunto $S_i = \{s_1, ..., s_{m_i}\}$ de estratégias puras, onde $m_i \geq 2$. O conjunto $S = S_1 \times S_2 \times ... \times S_n$ representa a combinação de todas as estratégias possíveis, onde um vetor $x = (x_1, x_2, ..., x_n) \in S$ representa um perfil de estratégias puras, sendo que x_i é uma estratégia pura para o jogador i.*

Para cada jogador i existe uma função de payoff *$f_i : S \to \mathbb{R}$, a qual define o* payoff *$f_i(x)$ do jogador i para um dado perfil de estratégia $x \in S$.*

Um jogo estratégico G é então definido como a tupla consistindo do conjunto S e a função de payoff *f para n jogadores, i.e.*

$$G = (S, f).$$

O equilíbrio de Nash é uma estratégia de solução no qual cada jogador, levando em conta as estratégias dos outros jogadores, não tem nenhum incentivo para mudar a sua estratégia. Um equilíbrio de Nash é estável. Uma vez que um equilíbrio seja obtido, nenhuma alteração unilateral da estratégia é lucrativa para qualquer jogador.

Um dos exemplos mais conhecidos na área de teoria de jogos é o dilema do prisioneiro. Nele, dois suspeitos são presos pela polícia. A polícia não tem provas suficientes para condená-los. Cada um tem a escolha de confessar o crime ou ficar em silêncio. Se ambos ficarem em silêncio, não haverá provas para condená-los por esse crime e eles ficarão na cadeia por um breve período de dois anos por crimes anteriores. Caso somente um deles confesse, o seu tempo de cadeia será de apenas um ano e o outro ficará com cinco anos. Caso ambos confessem, eles terão uma sentença de quatro anos.

Table A.1: Dilema do prisioneiro

P1 \ P2	**Confessar**	**Negar**
Confessar	$-4, -4$	$-1, -5$
Negar	$-5, -1$	$-2, -2$

A Tabela A.1 apresenta a matriz de *payoffs* do jogo do dilema do prisioneiro. Nesta matriz, os números de cada célula i, j representam os *payoffs* da estratégia em que o prisioneiro P1 escolhe a estratégia i e o prisioneiro P2 escolhe a estratégia j. Esta forma de representar um jogo é conhecida como forma normal, na qual uma matriz mostra todos os jogadores, estratégias e *payoffs*.

Esse exemplo deixa claro que nem sempre um equilíbrio de Nash corresponde a uma estratégia de solução com o maior benefício mútuo. Outra característica importante é que um jogo pode ter múltiplos equilíbrios de Nash, com *payoffs* completamente diferentes entre eles. O perfil de estratégias que fornece o maior benefício mútuo, isto é, a estratégia que otimiza a soma dos *payoffs* presentes nela em relação a todas as outras possíveis estratégias é chamada de *ótimo do sistema*.

Um equilíbrio de Nash pode ser puro ou misto. Um equilíbrio é puro se cada jogador i utiliza somente um perfil de estratégias puras x_i, enquanto que em um equilíbrio misto é permitido aos jogadores utilizar um conjunto de estratégias com uma certa probabilidade associada a cada perfil de estratégia pura.

A.2 Jogos *Bimatrizes*

Em jogos bimatrizes, dois jogadores P1 e P2 tem suas estratégias puras determinadas por matrizes A e B, respectivamente, com m estratégias para o jogador P1 e n estratégias para o jogador P2. Um equilíbrio de Nash neste tipo de jogo pode ser definido como um vetor de probabilidades para cada jogador indicando qual a chance de cada estratégia pura ser utilizada neste equilíbrio.

A.2.1 O algoritmo de Lemke-Howson

O algoritmo de Lemke-Howson (LEMKE; J. T. HOWSON, 1964) foi formulado para encontrar um equilíbrio de Nash em um jogo de dois jogadores em forma normal. No seu pior caso, ele pode ter um número exponencial de passos para achar um equilíbrio válido, sendo que instâncias que provoquem esse comportamento podem ser encontradas em (SAVANI; STENGEL, 2004). Apesar de existirem esses piores casos, pouco estudo experimental foi realizado nestas instâncias, particularmente em como perturbações alteram a performance do algoritmo.

O algoritmo utiliza uma técnica de pivoteamento iterado (*iterated pivoting*). Seja a matriz A e B a representação dos *payoffs* dos jogadores P1 e P2, respectivamente, uma operação de pivoteamento em A é realizada como um primeiro passo e usa-se o resultado desta operação como parte na decisão de onde fazer o pivoteamento na matriz B do jogador P2.

Outra forma de visualizar esse processo é através da representação baseada em polítopos. Nesta representação, o polítopo P é formado pela matriz B (transposta) como um conjunto de desigualdades e o polítopo Q é formado pela matriz A transposta. Eles são definidos como:

$$P = \left\{x \in \mathbb{R}^m \mid x \geq \mathbf{0},\ B^T x \leq \mathbf{1}\right\}, \quad Q = \left\{x \in \mathbb{R}^n \mid y \geq \mathbf{0},\ A^T y \leq \mathbf{1}\right\}$$

A.3 Preço da Anarquia

O Preço da Anarquia (*Price of Anarchy* (PoA)) é uma medida da possível ineficiência de equilíbrios Nash introduzido por Koutsoupias e Papadimitriou (KOUTSOUPIAS; PAPADIMITRIOU, 1999). Em um dado jogo, o ela compara o melhor resultado possível do sistema como um todo em relação ao pior equilíbrio Nash possível neste jogo, tendo como métrica a soma dos *payoffs* de todos os jogadores neste equilíbrio.

Definition A.2 (Preço da Anarquia)**.** *Seja G um jogo estratégico com n jogadores, com um conjunto de estratégias S_i para cada jogador i e uma função de custo $c_i : S \to \mathbb{R}$, onde $S = S_1 \times \cdots \times S_n$. Ainda, seja $C : S \to \mathbb{R}$ uma função do custo social que mapeie todo perfil de estratégia $s \in S$ para um valor não negativo que represente o custo do jogo. Dada a instância $I = (G, (S_i), (c_i))$, NE(I) é o conjunto de perfis de estratégias $s \in S$ que são equilíbrios de Nash para I.*

O Preço da Anarquia de I é definido como

$$\textit{PoA}(I) = \frac{\max_{s \in \textit{NE}(I)} C(s)}{\min_{s \in S} C(s)} \quad ,$$

e o Preço da Anarquia da classe de jogos $\mathcal{G}$ é definido como

$$\textit{PoA}(\mathcal{G}) = \max_{I \in \mathcal{G}} \textit{PoA}(I)$$

A.4 Problema de Atribuição de Tráfego

No Problema de Atribuição de Tráfego (*Traffic Assignment Problem (TAP)*) , cada usuário navega em um grafo representando uma rede viária a partir de uma origem para um destino. Cada aresta desse grafo tem um determinado custo associado. De que maneira que os usuários desta rede se comportam conduz a dois problemas de otimização. Os usuários podem ter como objetivo simplesmente minimizar o seu próprio tempo, se comportando de maneira egoísta, análoga a como os jogadores se comportam segundo a teoria dos jogos. Em contrapartida, os usuários podem se comportar de forma cooperativa, buscando minimizar o tempo médio de viagem de todos os usuários.

Esses comportamentos foram definidos por Wardrop, em (WARDROP, 1952). O primeiro princípio de equilíbrio de Wardrop corresponde ao comportamento egoísta dos usuários, enquanto que o segundo princípio corresponde ao comportamento cooperativo entre os usuários. Se considerarmos os usuários como jogadores, o primeiro princípio de equilíbrio é equivalente, neste contexto, ao equilíbrio de Nash. Além disso, o segundo princípio de equilíbrio é análogo ao ótimo do sistema, no qual o custo total dos usuários é minimizado.

Uma rede viária neste contexto é representado como um multigrafo direcionado $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas. Os usuários (jogadores) são modelados por um conjunto de *commodities* K com cada *commodity* $i \in K$ tendo um par de vértices associado $(s_i, t_i) \in V \times V$. Usuários que possuem o mesmo par origem-destino (s_i, t_i) pertencem a mesma *commodity* i. Para cada *commodity* $i \in K$

existe uma demanda d_i que especifica o fluxo total (correspondendo aos usuários da *commodity* i) que necessita viajar de s_i para t_i.

O conjunto de caminhos de s_i para t_i é denominado como $\mathcal{P}_i$. Seja $\mathcal{P} = \cup_{i \in K} \mathcal{P}_i$. Um fluxo f especifica para cada caminho $P \in \mathcal{P}$ um valor não negativo de fluxo que atravessa P, i.e., f é uma função $f : \mathcal{P} \to \mathbb{R}^+$. O fluxo na aresta $e \in E$ é definida como $f_e = \sum_{P:e \in P} f_P$, onde $P \in \mathcal{P}$. Um fluxo f é *factível* se ele satisfaz a demanda para cada *commodity*, i.e., $\sum_{P \in \mathcal{P}_i} f_P = d_i$ para cada $i \in K$.

Para cada aresta $e \in E$ é dado uma função de latência $l_e : \mathbb{R}^+ \to \mathbb{R}^+$ que mapeia o fluxo f_e de uma aresta e para o tempo de travessia $l_e(f_e)$. É assumido para esse problema que todas as funções de latência são não-negativas, diferenciáveis e não-decrescentes. A latência do caminho $P \in \mathcal{P}$ é definida como $l_P = \sum_{e \in P} l_e(f_e)$. Por consequência, usa-se (G, d, l) para se referir a uma instância do Problema de Atribuição de Tráfego.

Para se avaliar o tempo total de travessia de uma rede, é definido uma função de custo $c(f) = \sum_{e \in E} l_e(f_e) f_e$. O ótimo do sistema (*system optimum*), ou *fluxo ótimo* é um fluxo factível f^* que minimize esse tempo total de travessia determinado por $c(f^*)$.

Um fluxo factível f é denominado um *equilíbrio de usuário*, ou um *fluxo Wardrop*, ou ainda um equilíbrio Nash para o TAP se o fluxo de cada *commodity* i viaja por um caminho de latência mínima disponível. Isto é, para cada *commodity* i todos os caminhos que tenham algum fluxo positivo tem o mesmo valor de latência e todos os outros caminhos não possuem latência menor.

Definition A.3 (Fluxo Wardrop)**.** *Um fluxo f é um fluxo Wardrop se*

$$\forall i \in K, \ \ \forall P_1, P_2 \in \mathcal{P}_i, \ \ f_{P_1} > 0 : \quad l_{P_1}(f) \leq l_{P_2}(f). \tag{A.1}$$

Note que, enquanto é possível existir múltiplos equilíbrios, o custo $c(f)$ de um fluxo Wardrop f é único (mais detalhes em (ROUGHGARDEN, 2003)).

Um fluxo ótimo corresponde a um fluxo Wardrop com relação a funções de custo marginais (*marginal cost functions*). A função de custo marginal de uma aresta e é definida como $l_e^*(x) = l_e(x) + x\frac{d}{dx}(l_e(x))$. Um fluxo factível f^* é um fluxo ótimo para (G, d, l) se e somente se ele é um fluxo Wardrop para a instância (G, d, l^*) (mais detalhes em (ROUGHGARDEN, 2003)).

A.4.1 Preço da Anarquia para o Problema de Atribuição de Tráfego

No contexto do Problema de Atribuição de Tráfego (TAP), a definição descrita em A.2 pode ser simplificada para o seguinte:

Definition A.4 (Preço da Anarquia para o TAP)**.** *Seja $I = (G, d, l)$ uma instância do TAP. O Preço da Anarquia de I é*

$$\mathit{PoA}(I) = \frac{c(f)}{c(f^*)},$$

onde f e f^* é um fluxo Wardrop e um fluxo ótimo de I, respectivamente.

O Preço da Anarquia para o TAP é definido como

$$\mathit{PoA} = \max_I \mathit{PoA}(I),$$

onde o máximo é tomado sobre todas as possíveis instâncias de entrada.

O Preço da Anarquia é dependente diretamente de quais os tipos de funções de latência as instâncias do TAP podem ter. Roughgarden e Tardos (ROUGHGARDEN; TARDOS,

2002) provaram que para latências lineares o Preço da Anarquia é $\frac{4}{3}$. Além disso, Roughgarden provou que o Preço da Anarquia é independente da topologia da rede viária (ROUGHGARDEN, 2003). Entre outros resultados, estes estudos revelam que o Preço da Anarquia para funções de latência polinomiais acontecem em instâncias bastante simples que consistem de somente dois arcos paralelos, também conhecidos como *instâncias Pigou* (Figura A.1). Para o caso polinomial, foi provado que o PoA é $\text{PoA}(I) = \frac{(p+1)\sqrt[p]{p+1}}{(p+1)\sqrt[p]{p+1}-p} \in \Theta(\frac{p}{\ln p})$, onde p é o grau do polinômio permitido.

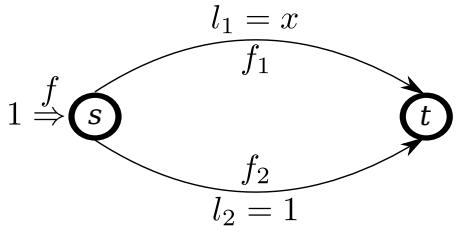

Figure A.1: Instância Pigou com funções de latência lineares.

A.5 Análise Suavizada (*Smoothed Analysis*)

A análise suavizada (*Smoothed Analysis*) de algoritmos foi definida por Spielman e Teng em (SPIELMAN; TENG, 2004). Ela é uma abordagem relativamente nova à análise de algoritmos, em contraste à análise de pior caso e à análise de caso médio. Uma das principais motivações desse tipo de análise é para tentar entender o algoritmo Simplex, que tem uma complexidade de pior caso exponencial e ainda sim é um algoritmo muito eficiente na prática.

A complexidade suavizada de um algoritmo é o máximo sobre as entradas do tempo de execução esperado desse algoritmo sob pequenas perturbações nestas entradas. Isso significa que a complexidade suavizada é medida não somente no tamanho n da entrada mas também sobre uma magnitude de perturbação σ.

Definition A.5 (Complexidade Suavizada (*Smoothed Complexity*))**.** *Seja $T_A(x)$ o tempo de execução do algoritmo A para a instância x. Seja X_n o conjunto de instâncias de A com o tamanho n.*

Para uma instância x e um parâmetro de magnitude σ, e $\mu_\sigma(x)$ denote um conjunto de instâncias que podem ser obtidas a partir de x ao se aplicar perturbações aleatórias de magnitude σ. Então, a complexidade suavizada do algoritmo A sob perturbações de magnitude σ é:

$$S(A, n, \sigma) = \max_{x \in X_n} \left(\mathop{\mathbf{E}}_{\bar{x} \in \mu_\sigma(x)} [T_A(\bar{x})] \right).$$

A.6 Complexidade Suavizada de Jogos Bimatrizes

Em (CHEN; DENG, 2006), é provado que a complexidade suavizada do algoritmo Lemke-Howson não é polinomial, tanto para perturbações uniformes quanto para perturbações Gaussianas. Os autores provam que não existe um algoritmo que computa um equilíbrio de Nash ϵ-aproximado em tempo polinomial. Segue-se do artigo que esse fato

é equivalente ao problema de se encontrar um equilíbrio de Nash não estar em tempo polinomial suavizado sob perturbações σ.

A maneira que a prova é construída entretanto não fornece uma forma direta de construir uma instância que cause que o algoritmo LH se comporte exponencialmente mesmo com perturbações a esta instância. Isso torna mais difícil verificar o quão provável é de se encontrar esse tipo de instância em um cenário realista. Consequentemente, nos experimentos realizados as instâncias geradas foram as descritas por Savani e Stengel (SAVANI; STENGEL, 2004), as quais são as únicas conhecidamente exponenciais para o algoritmo LH.

A.6.1 Resultados Experimentais para Jogos Bimatrizes

Foi desenvolvido uma implementação do algoritmo LH baseado na implementação de Codenotti et al. (CODENOTTI; ROSSI; PAGAN, 2008). O *software* foi escrito em C/C++ e foi adaptado para utilizar números racionais, utilizando-se a biblioteca GMP (*GNU Multiple Precision Arithmetic Library*) (GRANLUND et al., 2011), ao invés da representação em ponto flutuante utilizada originalmente. Esta implementação com estas modificações provou ser mais estável e com melhor performance do que a implementação usada no pacote GAMBIT (MCKELVEY; MCLENNAN; TUROCY, 2010), que é o pacote mais utilizado atualmente para a análise de equilíbrios Nash.

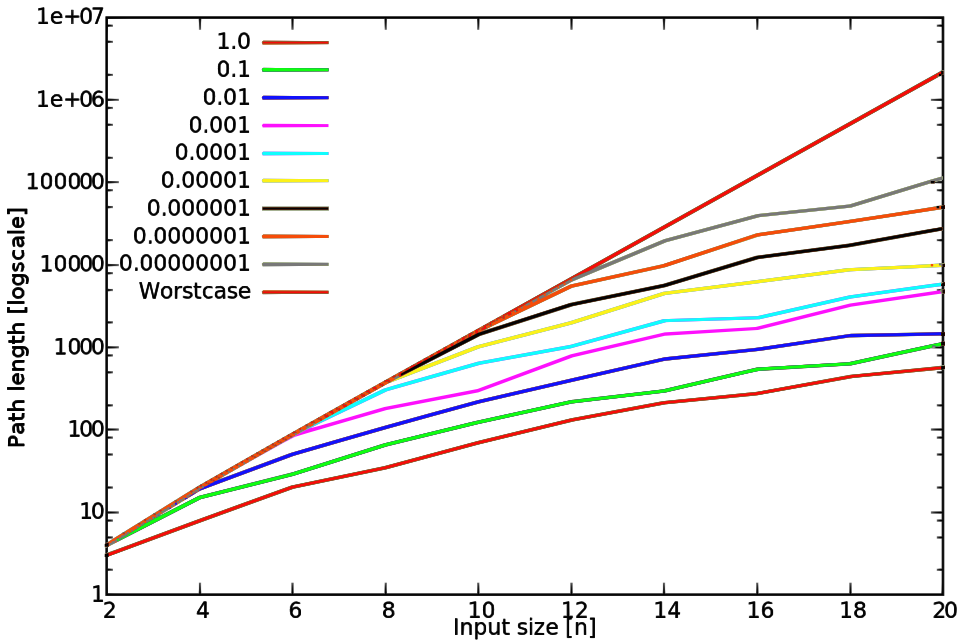

Figure A.2: Tamanho do caminho utilizado pelo LH para cada instância perturbada em função do tamanho da instância e com perturbações constantes. Note que a lista na legenda está em ordem inversa aos dados no gráfico.

Na Figura A.2, o parâmetro σ foi definido como 10^i, para $i = 0, -1, ..., -8$. O pivô inicial foi fixado como n, uma vez que este é o pior caso para o algoritmo. Para cada uma das instâncias de pior caso (com $n \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$) e para cada σ, 100 instâncias uniformemente perturbadas aleatoriamente foram geradas e executadas, e suas médias foram usadas como os dados para o gráfico. Para a Figura A.3, a magnitude σ é definida como uma função de n.

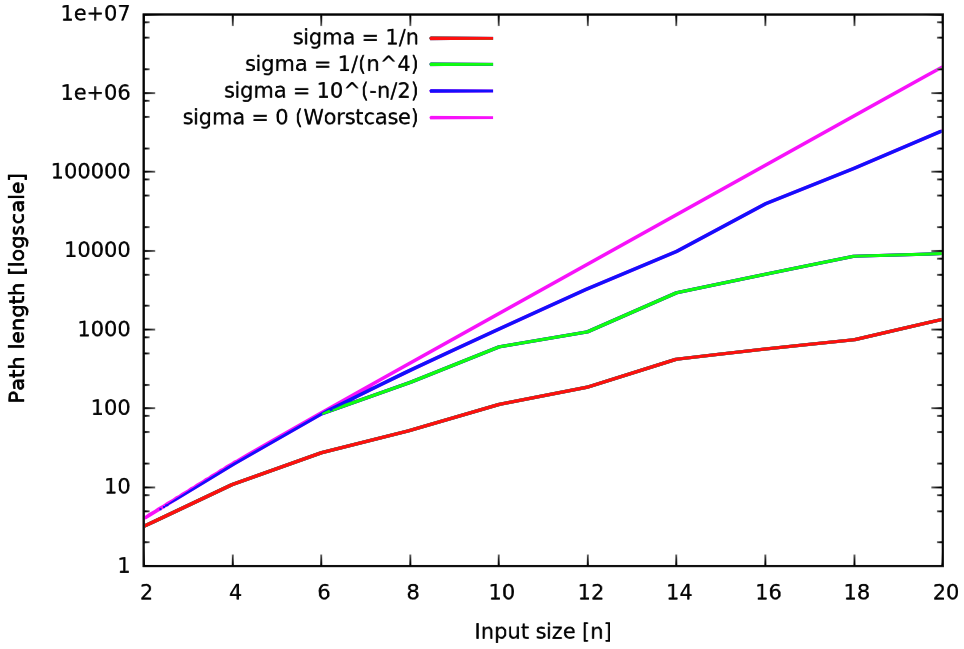

Figure A.3: Tamanho do caminho utilizado pelo LH para cada instância perturbada em função do tamanho da instância e com perturbações em função de n. Note que a lista na legenda está em ordem inversa aos dados no gráfico.

Ambos os gráficos indicam que, ao contrário do que se possa esperar pela prova de Chen et al., o tempo de execução para as instâncias de pior caso de Savani e Stengel, quando aplicadas perturbações, mesmo que muito pequenas, realmente tem um tempo de execução polinomial ao invés de exponencial. Note que esta conclusão é feita levando em conta um tamanho de instância n relativamente pequeno de somente 20.

A.7 Preço da Anarquia Suavizado (*Smoothed Price of Anarchy*)

Enquanto que a ideia original da análise suavizada é para medir a complexidade temporal de algoritmos, o conceito é também adequado para analisar outras medidas de qualidade. Uma medida de qualidade ainda não explorada a fundo até o presente momento é utilizar a análise suavizada para analisar a medida Preço da Anarquia presente em teoria dos jogos.

A ideia de análise suavizada é estendida aqui para analisar o Preço da Anarquia. A ideia é para se perturbar aleatoriamente uma dada entrada e analisar a esperança (valor esperado) do Preço da Anarquia nas entradas perturbadas. Primeiro é definido o Preço da Anarquia Suavizado para uma instância I.

Definition A.6 (Preço da Anarquia Suavizado de I)**.** *Seja $\mathcal{G}$ uma classe de jogos. Dado uma instância $I = (G, (S_i), (c_i)) \in \mathcal{G}$, ela é aleatoriamente perturbada adicionando-se ruído randômico aos dados de entrada.*

Seja $\bar{I}$ uma instância que possa ser obtida de I por perturbações aleatórias e seja σ o parâmetro para a magnitude desta perturbação. O Preço da Anarquia Suavizado (SPoA) *de I é*

$$SPoA(I, \sigma) = \frac{\max_{s \in NE(\bar{I})} \mathbf{E}[\bar{C}(s)]}{\min_{s \in S} \mathbf{E}[\bar{C}(s)]},$$

onde a expectância é tomada sobre todas as instâncias $\bar{I}$ que podem ser obtidas a partir de I por perturbações aleatórias de magnitude σ. A função $\bar{C}$ se refere aqui ao custo da instância perturbada.

Finalmente, o Preço da Anarquia Suavizada de um jogo $\mathcal{G}$ pode ser definido.

Definition A.7 (Preço da Anarquia Suavizado)**.** *Usando as mesmas suposições de A.6, o Preço da Anarquia Suavizado de $\mathcal{G}$ é*

$$\mathit{SPoA}(\mathcal{G}, \sigma) = \max_{I \in \mathcal{G}} \mathit{SPoA}(I, \sigma) \ .$$

A.8 Preço da Anarquia Suavizado para o Problema de Atribuição de Tráfego

No contexto do Problema de Atribuição de Tráfego (TAP), as instâncias são perturbadas ao se adicionar um ruído aleatório para as funções de latência. Estas perturbações refletem as flutuações comuns em casos reais que podem acontecer nas arestas da rede viária. Mais especificamente, suponha que é dada uma instância $I = (G, d, l)$ do TAP. As funções de latência perturbadas $\bar{l}$ são definidas abaixo.

Definition A.8 (Funções de Latência Perturbadas)**.** *Dado uma instância $I = (G, d, l)$, então funções de latência perturbadas são definidas como:*

$$\forall e \in E: \quad \bar{l}_e = (1 + \varepsilon_e) l_e, \quad \varepsilon_e \xleftarrow{i.u.r} [0, \sigma]. \tag{A.2}$$

Note que ε_e é escolhido uniformemente independentemente ao acaso do intervalo $[0, \sigma]$ para cada aresta $e \in E$. O Preço da Anarquia Suavizado para o Problema de Atribuição de Tráfego pode então ser definido como abaixo.

Definition A.9 (Preço da Anarquia Suavizado de I para o TAP)**.** *Seja f_I e f_I^* fluxos Wardrop e ótimos, respectivamente, para uma dada instância $I = (G, d, l)$. O Preço da Anarquia Suavizado de I é definido como*

$$\mathit{SPoA}(I, \sigma) = \frac{\mathbf{E}[\bar{c}(f_{\bar{I}})]}{\mathbf{E}[\bar{c}(f_{\bar{I}}^*)]},$$

onde $\bar{c}$ se refere ao custo total com respeito as funções de latência perturbadas, i.e., $\bar{c}(f) = \sum_{e \in E} \bar{l}_e(f_e) f_e$ para um dado fluxo f.

Como na Definição A.6, a expectância é tomada sobre todas as instâncias $\bar{I}$ que possam ser obtidas a partir de I por perturbações de magnitude σ.

Considere as instâncias Pigou mostradas na Seção A.4.1 com funções de latência polinomiais. São derivados limites exatos no Preço da Anarquia Suavizado para perturbações uniformes para estas instâncias. Esses limites também estabelecem um limite inferior no Preço da Anarquia Suavizado para instâncias *multi-commodity* gerais.

Theorem A.1. *O Preço da Anarquia Suavizado da instância Pigou com funções de latência polinomiais de grau p é*

$$\mathit{SPoA}(I, \sigma) = \frac{3 + \sigma}{3 + \frac{3\sigma}{2} + \frac{3p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)(-1+p^2)\sigma^2}} \tag{A.3}$$

Prova do Teorema A.1. Seja I a instância Pigou original com dois vértices s e t, uma demanda de 1 unidade de fluxo e latências $l_1(x) = x^p$ para e_1 e $l_2(x) = 1$ para e_2. Após perturbar as funções de latências de I, é obtido a instância representada na Figura A.4 com latências

$$l_1(x) = (1 + \varepsilon_1)x^p \quad \text{and} \quad l_2(x) = 1 + \varepsilon_2,$$

onde $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$ são variáveis aleatórias.

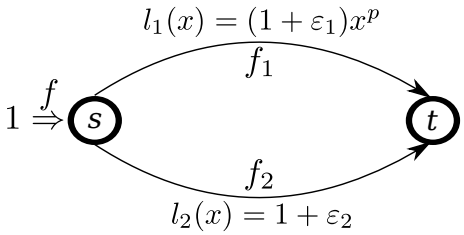

Figure A.4: Instância Pigou com latências polinomiais e perturbações $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$.

Primeiro é determinado um fluxo Wardrop. Com a adição da perturbação, o arco e_1 é utilizado na sua totalidade desde que sua latência seja menor que a latência de e_2. Portanto,

$$f_1 \leq \sqrt[p]{\frac{1 + \varepsilon_2}{1 + \varepsilon_1}}.$$

Uma vez que esse valor pode ser maior que o fluxo máximo factível,

$$f_1 = \min\left(\sqrt[p]{\frac{1 + \varepsilon_2}{1 + \varepsilon_1}}, 1\right) \quad \text{e} \quad f_2 = 1 - \min\left(\sqrt[p]{\frac{1 + \varepsilon_2}{1 + \varepsilon_1}}, 1\right).$$

O fluxo f_1 será 1 desde que $\varepsilon_2 \geq \varepsilon_1$. Se este é o caso, então $c(f) = 1 + \varepsilon_1$. Se $\varepsilon_2 \leq \varepsilon_1$, então

$$c(f) = \sqrt[p]{\frac{1 + \varepsilon_2}{1 + \varepsilon_1}}(1 + \varepsilon_1)\left(\sqrt[p]{\frac{1 + \varepsilon_2}{1 + \varepsilon_1}}\right)^p + \left(1 - \sqrt[p]{\frac{1 + \varepsilon_2}{1 + \varepsilon_1}}\right)(1 + \varepsilon_2) = 1 + \varepsilon_2.$$

Para determinar $\mathbf{E}[c(f)]$, para $\varepsilon_1, \varepsilon_2$ escolhidos uniformemente ao acaso de $[0, \sigma]$, é necessário calcular a seguinte integral dupla. (Note que a função densidade combinada é $\frac{1}{\sigma^2}$).

$$\begin{aligned}\mathbf{E}[c(f)] &= \int_0^\sigma \int_0^{\varepsilon_2} \frac{1 + \varepsilon_1}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 + \int_0^\sigma \int_{\varepsilon_2}^\sigma \frac{1 + \varepsilon_2}{\sigma^2}\ d\varepsilon_1 d\varepsilon_2 \\ &= \frac{3 + \sigma}{6} + \frac{3 + \sigma}{6} = 1 + \frac{\sigma}{3}\end{aligned}$$

Para se computar o fluxo ótimo do sistema, é explorado o fato que um fluxo ótimo é um fluxo Wardrop com respeito as funções de custo marginais $l_1^*(x) = (p + 1)(\varepsilon_1 + 1)x^p$ e $l_2^*(x) = 1 + \varepsilon_2$. Então

$$f_1^* = \sqrt[p]{\frac{1 + \varepsilon_2}{(p + 1)(\varepsilon_1 + 1)}} \quad \text{e} \quad f_2^* = 1 - \sqrt[p]{\frac{1 + \varepsilon_2}{(p + 1)(\varepsilon_1 + 1)}}.$$

Note que σ precisa ser menor do que p para que o fluxo ótimo f_1^* permaneça abaixo do fluxo máximo. O custo de f^* é

$$\begin{aligned} c(f^*) &= f_1^*(1+\varepsilon_1)\,(f_1^*)^p + (f_2^*)\,(1+\varepsilon_2) \\ &= 1+\varepsilon_2 - p(1+p)^{-1-\frac{1}{p}}(1+\varepsilon_1)^{-\frac{1}{p}}(1+\varepsilon_2)^{1+\frac{1}{p}}. \end{aligned}$$

Calculando a expectância sobre as escolhas aleatórias $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$, é obtido

$$\begin{aligned} \mathbf{E}[c(f^*)] &= \int_0^\sigma \int_0^\sigma \frac{1+\varepsilon_2 - p(1+p)^{-1-\frac{1}{p}}(1+\varepsilon_1)^{-\frac{1}{p}}(1+\varepsilon_2)^{1+\frac{1}{p}}}{\sigma^2}\; d\varepsilon_1 d\varepsilon_2 \\ &= 1+\frac{\sigma}{2} + \frac{p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)\,(-1+p^2)\,\sigma^2} \end{aligned}$$

Portanto,

$$\mathrm{SPoA}(I,\sigma) = \frac{3+\sigma}{3+\frac{3\sigma}{2} + \frac{3p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)(-1+p^2)\sigma^2}} \tag{A.4}$$

□

Conforme mostrado na Seção A.4.1, o PoA da instância Pigou com latências polinomiais $\mathrm{PoA}(I) = \frac{(p+1)\sqrt[p]{p+1}}{(p+1)\sqrt[p]{p+1}-p} \in \Theta(\frac{p}{\ln p})$ é o pior caso para o Preço da Anarquia para o TAP. O limite inferior provado aqui mostra que apesar de o Preço da Anarquia melhorar com as perturbações, esta melhora é pequena. Mesmo para perturbações de magnitude $\sigma = 1$, a diminuição no PoA é de apenas 10%. Note que neste caso as funções de latência podem no mais extremo até mesmo dobrar de valor comparadas com as originais. Conforme o grau do polinômio cresce, esta melhora se torna mais significante. Se o σ for restrito a menor ou igual a $\frac{1}{p}$, então o Preço da Anarquia Suavizada permanece assimptoticamente $\Theta(\frac{p}{\ln p})$, assim como no caso determinístico.

A.8.1 Resultados Experimentais em Instâncias de *Benchmark*

Para avaliar se instâncias em cenários realistas se comportam de maneira diferente em relação aos piores casos para o Preço da Anarquia, foram testadas algumas instâncias de *benchmark* disponíveis gratuitamente para pesquisa acadêmica em *Transportation Network Test Problems* (BAR-GERA, 2011).

Para computar o equilíbrio de usuário e o ótimo do sistema, foi utilizado o algoritmo de Frank-Wolfe (FRANK; WOLFE, 1956). O algoritmo foi implementado em C/C++ e compilado para um sistema Linux, com um *kernel* versão $2.6.35$. A máquina utilizada para os testes tem um Intel® Core™ i7 CPU com 4 *cores*, com 12 GB de memória RAM.

As instâncias foram perturbadas com $\sigma \in \{10^{-9}, 10^{-8}, ..., 10^{-2}\}$. Foram também avaliadas instâncias com perturbações maiores, com $\sigma \in \{0.1, 0.2, ..., 0.9\}$. O algoritmo foi terminado no momento em que fosse atingido um *gap* relativo de menos de 10^{-5}, exceto para a instância *Sioux Falls*, na qual o *gap* relativo mínimo foi definido como 10^{-6}. Para cada magnitude de perturbação σ_i, foram executadas 10 computações e o valor médio foi considerado. Na Tabela A.2, o Preço da Anarquia das instâncias é apresentado. Na mesma tabela também é apresentado os resultados para as instâncias perturbadas, sendo que a média, o desvio-padrão, os valores mínimo e máximo são apresentados para todos os diferentes valores de σ combinados.

Table A.2: PoA original e medidas para o PoA perturbado encontrados para cada instância.

Nome	POA	Média	Mínimo	Máximo	Desvio-Padrão
Sioux Falls	1.039682	1.039689	1.039676	1.039707	8.049609×10^{-6}
Friedrichshain	1.086374	1.086422	1.086345	1.086599	4.996005×10^{-5}
Chicago Sketch	1.023569	1.023567	1.023561	1.023572	2.137639×10^{-6}
Berlin Center	1.006141	1.006142	1.006133	1.006155	3.831177×10^{-6}

Tanto o Preço da Anarquia original quanto o Preço da Anarquia Suavizado para as instâncias de *benchmark* estão bastante próximos de 1 para todas as instâncias testadas, o que fornece um pouco de evidência de que o pior caso não é muito provável de ocorrer em cenários realistas. Além disso, quando o Preço da Anarquia Suavizado é analisado, mesmo para σ relativamente grandes, o Preço da Anarquia Suavizado permanece quase constante e extremamente próximo ao Preço da Anarquia original.

A.9 Conclusões

Nesta dissertação foram estudados dois importantes assuntos em teoria dos jogos com relação ao efeito de perturbações. Para jogos Bimatrizes, foi analisado piores casos conhecidos na literatura para o algoritmo de Lemke-Howson (SAVANI; STENGEL, 2004) utilizando-se da análise suavizada, e foi encontrado evidência que perturbações nestas instâncias têm grande influência no seu tempo computacional. Para poder simular corretamente estas perturbações extremamente pequenas, uma implementação exata do algoritmo Lemke-Howson foi construída (RODRIGUES, 2011).

A ideia da análise suavizada foi estendida para se estudar o efeito de perturbações no Preço da Anarquia. O Preço da Anarquia (KOUTSOUPIAS; PAPADIMITRIOU, 1999) é uma medida em teoria dos jogos que compara o ótimo do sistema com o pior equilíbrio de Nash possível. Foi proposto uma medida da perturbação do Preço da Anarquia baseado em *smoothed analysis* para algoritmos, que foi denominada de *Smoothed Price of Anarchy* (Preço da Anarquia Suavizado, SPoA).

Utilizando o *Smoothed Price of Anarchy*, foi proposto um modelo de perturbação para o Problema de Atribuição de Tráfego (*Traffic Assignment Problem*). Foi provado um limite inferior para o *Smoothed Price of Anarchy* utilizando esse modelo de perturbação que é da mesma ordem do pior caso para latências polinomiais para o Preço da Anarquia. Um limite para o SPoA para latências lineares também foi provado.

Finalmente, é mostrado experimentalmente que os efeitos de perturbações no Preço da Anarquia para instâncias realistas, pelo menos para as instâncias de *benchmark* presentes na literatura atual, são severamente limitadas e não mostram nenhuma tendência em particular.

Essa dissertação é também a base para o artigo (RODRIGUES et al., 2011) aceito no "11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems", com o título de "On the Smoothed Price of Anarchy of the Traffic Assignment Problem", um trabalho conjunto com Guido Schäfer, Luciana Buriol e Marcus Ritt.