A simulator for non-cooperative games

VALENTINO PACIFICI

KTH Elektro-
och systemteknik

Examensarbete

Stockholm, Sverige October 2010

-

Advisor at KTH: György Dán

Advisor at Politecnico: Matteo Cesana

Examiner: Viktória Fodor

XR-EE-LCN 2010:006

"What's money? A man is a success if he gets up in the morning and goes to bed at night and in between does what he wants to do."

Bob Dylan

Sommario

Questa tesi specialistica descrive il lavoro teorico e le scelte di progettazione che hanno portato alla realizzazione di un *simulatore di giochi non cooperativi*. Il lavoro include inoltre diversi risultati teorici raggiunti grazie all'analisi formale di una classe di giochi conosciuti come *Replication games*. Questo report descrive infine i risultati dello studio relativo a questa particolare classe di giochi, ottenuti usando il simulatore in questione.

Il simulatore offre tutti gli strumenti che permettono di rappresentare in modo efficiente qualunque gioco che abbia insiemi delle strategie finiti, e di simularlo seguendo i processi di apprendimento più studiati in letteratura. La sua architettura completamente modulare permette, tra le altre cose, di aggiungere facilmente particolari controlli sul processo di apprendimento, consentendo di verificare proprietà generiche risultanti, ad esempio, da studi teorici. Inoltre, la sua capacità di sfruttare architetture hardware parallele, congiuntamente con la possibilità di definire computazioni indipendenti, permette di portare a termine simulazioni in modo particolarmente efficiente.

Nella parte formale della tesi è riportata la dimostrazione di equivalenza di alcune particolari sottoclassi dei *Replication games* con la classe dei *Player-Specific Congestion games* e dei *Matroid Congestion games*. Segue una revisione della letteratura effettuata con lo scopo di riunire i teoremi validi anche per questa sottoclasse. I risultati dimostrano che, imponendo alcuni vincoli, è possibile attribuire ai *Replication games* proprietà come l' *aciclicità debole* o quella degli *incrementi finiti*.
L'analisi formale fa supporre inoltre la totale assenza di cicli in alcune altre sottoclassi dei *Replication games*, ma tale risultato non è ancora stato provato.

Abstract

This dissertation describes the theoretic work and the practical design choices which brought to the realization of a *non-cooperative game simulator*. It also reports some theoretic results achieved from the formal analysis of a class of games called *Replication games*. It finally depicts the outcomes of an analysis of that particular class of games, obtained through the utilization of the simulator in question.

The simulator offers all the instruments to efficiently represent whichever game with finite action sets and to play it, following the learning rules mostly studied in the literature. Its completely modular architecture allows, among everything else, to easily add specific checks on the learning process, permitting to verify generic properties coming for example from theoretic results. Additionally, its capability to exploit hardware parallelism, together with the ability to take advantage of independent game actions, allows to perform particularly efficient simulations.

In the formal part of the dissertation is reported the proof of the equivalence of some particular subclasses of the *Replication games* to the class of *Player-Specific Congestion games* and the class of *Matroid Congestion games*. Follows a literature study to collect the theorems which hold also for this subclass. The results show that, under certain conditions, some fundamental properties like the *finite improvement* and the *weakly acyclicity* hold. The formal analysis suggests also the absence of cycles in some other subclasses of the *Replication games*, but this has not been proved yet.

Contents

Chapter 1

Introduction

THE main reason which leaded to the design and the development of a *non-cooperative game simulator* was the lack of a general purpose and efficient software, able to analyze particular scenarios and to verify in practice the theoretic results achieved in game theory in the last years. While realizing a general purpose simulator, one of the most important problems to cope with is how to allow a user to represent every possible game, with its specific action sets and utility functions. Note that a game description can be expressed using different algebric structures, such as multi dimensional matrixes, and the calculation of the utility function is subordinate to that.

Another decisive factor concernig game simulation, is the efficiency of the software in charge to replicate the behaviour of the game. Consider the main trend of the last years in computer architecture: the limits in the realization of higher frequency processor have been overcome by an higher level of computational parallelism. The modern CPUs have multiple cores, which allow to exploit the parallelism of multi tasking operating systems. Every individual application though, is generally far to offer a instrinsic level of parallelism which can make use of multi core architectures. This is maily because computer programmers think sequentially and it is generally hard to identify the independent parts of an algorithm that can be programmed

concurrently.
Hence, an important goal for a game simulation software would be the capability to exploit the parallel calculation power offered by modern architectures, by for example, locating and handling concurrently independent calculations.
Furthermore, in order to keep up with the new results in literature, the simulator should have a modular architecture, which should allow, among everything else, to easily add specific checks on the learning process, permitting to verify generic properties coming from theoretic results.

The class of games I mainly coped with, models the interaction among players who need to retrieve some objects from different storage places. Each player owns a storage place and can fully customize its content. The retrieval from different places has different costs, thus each player is evidently affected by the actions of the others. Such a model has been designed to represent the interaction among Internet service providers which have at their disposal a content cache, but can also draw on other ISPs' cache.
The main purpose of this study is to check if it is really worth for an ISP to establish peering agreements with other ISPs: agreements which permit cooperation between their caches. Some important results concerning this issue are presented by Dán in [5].
Positive results related to this problem would probably change the conception of cooperation among ISPs. Actually, from 50 to 80 percent of the total Internet traffic is generated by *peer-to-peer* content distribution systems. Cooperative caching may allow ISPs to reduce their total traffic cost, even if they follow their selfish interest in choosing the content of their own cache.

1.1 Methodology

The methodology, followed in order to complete the entire study, consists, first of all, of a comprehensive literature inspection concerning *Game theory*

together with many important results achieved in the same field.
I learned the basics of *Game theory* from [19] and [20] and I then focused on articles about different class of games. The aim of the latter study was to deeply understand the different properties and features of already existent game classes, in order to be able in the future to recognize possible analogies with the new class I was about to analyze.
The literature study included also the most common learning processes investigated at present, together with the main results concerning their capacity to convergence to a Nash Equilibrium under different assumptions. This research provided the instruments necessary to design the general purpose simulator and to investigate the behavior and properties of the *Replication games*. I give an account of all this preliminary study in Chapter 2, paying particular attention to provide an unified notation.
Afterwards, keeping in mind the main algebraic structures of *Game theory*, I have been engaged in the design of the simulator: the main focus during this process was to keep an efficient structure, which could also allow future users to add capabilities and features. The decisions on the structure has been taken trying to extract common elements from the different learning rules found in literature, so that the eventual process to add a learning process would have been easy.
As soon as the simulator was capable to run games' instances I started the analysis, both formal and practical, of the *Replication games*.
The very final stage of the study was the collection of the simulations' outcomes and their analysis. The achievements and the results of the previous phase were used to extract regularities and to infer general rules concerning the behavior of *Replication games*.

1.2 Structure of the report

The outcome of this study will be deeply reported and explained in the rest of the dissertation:

Chapter 2 provides a high level overview about *Game theory*, together with the formal definition and explanation of all the concepts the reader needs in order to understand the core of this thesis. The last part of the chapter concerning the different learning rules is particularly important to get into the remaining sections. It also describes some relevant results in the literature, related to different class of games.
Chapter 3 introduces the *Replication game* class, together with its formal definition and properties. Follow the equivalence proof I already introduced earlier in this introduction, and some considerations regarding the utility function of these games.
Chapter 4 illustrates the most important design choices which have been made during the realization of the simulator. It describes the main features and how the simulator can achieve a high level of parallelism, given that the user provides the right information about the game she wishes to play.
Finally Chapter 5 depicts the methodology followed to tune and realize the simulations, and the approach to evaluate the results. Every result has been inspected for regularities and common trends, and in all such cases a theoretic explanation has been provided.

Chapter 2

Game Theory

Game theory addresses the problem of modeling a broad set of situations where different decision-makers interact. Each one of these situations is a *game*. Every *decision-maker*, or *player* of the game, choose his actions according to his well-defined objectives and using whatever information is available to him. In general the players' actions will cause a general outcome that may reward some players more than others. Thus, what one decides to do strictly depends on one's *beliefs* about the other players' behavior, and vice-versa.
This abstraction is flexible enough to model lots of real-life scenarios, thus it finds application in different subjects such as economics, biology, engineering, political and social science, as well as international relations, computer science, and philosophy.

Some of the most common examples are the competition among firms (economics), war and peace negotiations among countries (international relations) or the opposition between two candidates competing for votes (political science). There are also psychological games played on a personal level, where the actions are words, and the payoffs are good or bad feelings (social science). In the competition among species (biology) the natural selection may be modeled as a game played among genes.

The purpose of *game theory* is to study such models, in order to suggest ways in which a player behavior may be defined to achieve his own highest possible benefit. In order to better analyze these models they will be expressed in a formal way, using the language of mathematics. Nevertheless in the following sections I will provide a non-formal introduction to the main concepts that will be better defined later. This should help the reader to keep in mind the whole picture I am going to present.

2.1 Game theory overview

A *game* consist of strategic interactions. Its mathematical description includes the constraints on the actions that each player can take in a certain situation and specifies in different ways the players' interests, generally defined as functions of the possible outcomes. What the game description does not specify are the actions that the players *do* take. To find out what are the *best* actions for each player, in respect to the other players' actions, is in fact the final aim of the model.
The problem, of course, is that there is no simple way to define what is the *best*, since what is the best outcome for one agent may not be good for another. Thus is usually unlikely that taking the best action for a player will actually bring to the desired outcome, since the other players do not have the same preferences on the results.

This problem introduces the concept of *solution* of a game. What strategy should actually choose each player? Before answering this question we need to go through some basic concepts and classifications related to games.

Strategic Games and Extensive Games

Both strategic and extensive games model interactions among players competing on a set of possible outcomes.

In a *strategic game* (or game in *normal form*) each player chooses his action once and for all; this decision is taken simultaneously with the other players. In such a way, in the moment of making the decision, he is not informed on the actions chosen by all other players.
By contrast, an *extensive game* includes the concept of order of events: each player can consider his plan of action not only at the beginning, but also whenever he has to make a decision.
The games in strategic form will be further defined in the following sections; the extensive form though, is not relevant for this dissertation's purpose.

Pure and Mixed Strategies

When a player makes his decision he chooses one action from the set of the possible ones. The set of the actions all the players decided to do is called *strategy* or *action profile*. Thus a strategy is an n-tuple s, where n is the number of players, consisting of the actions chosen by each player. We refer to player i's strategy in s as s_i, that indicates the action chosen by player i in strategy s.
When a strategy is a set of simple actions is called *pure strategy*. We refer to a *mixed strategy* when agents can take different actions, each with some fixed probabilities. Of course in a mixed strategy, the sum of the probabilities for all actions of each player must be 1.
To better clarify these concepts the following example is provided:

Example 2.1.1. *We define a two players game G. For each player i of the game we have the set of his possible strategies or actions indicated with A_i. We may take for example $A_1 = \{a, b\}$ and $A_2 = \{c, d\}$. An example of a pure strategy profile s can be $s = (b, c)$: this express both the actions chosen by player 1 and 2, and defines an outcome of the game. Player 1 and 2 may have of course different valuations of this outcome. I will describe how to formally express these preferences over outcomes in the following section.*
A mixed strategy s for the game G can be $s = ((0.5, 0.5), (0.2, 0.8))$. Each

element of the 2-tuple s is itself a 2-tuple. In particular $(0.2, 0.8)$ *states that player* 2 *takes action c with probability 0.2, and action d with 0.8.*

Non-cooperative Games

We may distinguish between two classes of theoretical game models: *cooperative* and *non-cooperative*. The former assumes that each player is interpreted as an individual decision maker, while the latter allows binding commitments among different players, considering the sets of possible joint actions of groups of players as primitives.
An important clarification is obliged since the first class of games does not totally preclude agreements among players: players can eventually make agreements, but they are meaningless unless they are self-enforcing. In other words in the class of non-cooperative games a player cannot be forced to follow a strategy because of an agreement (binding commitments), but can still "talk" with other players in order to acknowledge a strategy that under certain conditions is better for everybody. The concept of *non-cooperative* games will be further explained in the following sections of the document. *Cooperative* games on the contrary are not interesting for the aim of this thesis.

The rational assumption

A typical game theory assumption is that players are *rational*, in the sense that they take their decision only regarding their valuation of the possible outcomes. They are "selfish", acting just to maximize their own reward. Usually the decision-maker's preferences, or rewards, are specified by defining a preference relation over the set of all possible outcomes of the game. Exploiting the knowledge or expectations about the other players preferences, the rational decision maker acts in order to bring the game to an outcome that is the best according to his preference relation.

The perfect information assumption

If every player of a game has complete knowledge of the preference relations of all the other players and knows that the same hypothesis holds for the other players as well, then the game has *perfect information*. Note that keeping only the first assumption and ignoring the information that all the other players know about all the preference relations, describes a notably different situation: in fact a player might for example exploit the ignorance of the others about his preference relation.
The situations in which some of the decision makers are not certain about the knowledge or the preference relations of some of the others are modeled using *Bayesian games*. These models use the theories of von Neumann and Morgenstern [18] and of Savage [24] but they are essentially out of the scope of this thesis since I will exclusively focus on games with perfect information.

The solution concepts

There are many concepts of solution which have been developed and studied in the theory of games. Each solution concept is a formal rule that describes which strategies will be selected by players. Thus, a solution of a rational game is an answer to the question about the strategy each player should choose in order to achieve the maximum gain. This action is strictly dependent on what the other participants of the game will choose.

A first approach to the problem may be searching and discarding all the strategies that are never better than others. This introduces the concept of *strict dominance*: a player i's action a *strictly dominates* action b if player i always prefers the outcome brought by a over the one brought by b, regardless the actions taken by the other players. Thus, to reach the *iterated dominance* solution, the dominated strategies are eliminated in succession, agent by agent, until any player has no dominated strategy left. This algorithm eventually causes the choice, by player i, of his *dominant strategy*:

a strategy that dominates every other action player i can choose. Unfortunately it is very unlikely that a player of a game has a dominant strategy.

Another concept I must present when talking about solution concepts is the *Pareto optimality*. This concept describes outcomes where there is no player that can get more reward without hurting someone else: *Pareto optimal* (or *efficient*) situations are those in which it is impossible to make one person better off without necessarily making someone else worse off.
Pareto optimal solutions represent an important concept in a social welfare perspective but unfortunately they are usually *unstable* in a rational game: a player may have available other actions that provide him a higher reward and, following the rational assumption, there is no reason why he should not prefer them, even if this would mean that other players get lower rewards.

A solution to the lack of stability in solution concepts was presented by John Nash [16]. He defined the concept of *equilibrium*:
is an action profile a^* where no player i can do better by choosing an action different from a_i^*, given that every other player j adheres to a_j^*.
This definition suggests an intuitive interpretation of this solution concept: each player, given that everybody else observes the Nash Equilibrium, has no wishes to deviate from it, since his strategy is the best one under that assumption.
The assumption underlying the concept is that, in order to properly choose an action, each player must have a *belief* about the other player's actions. We assume the *belief* of each player being sufficiently extensive so that he knows how his opponents will behave. Note that this does not represent an advantage for any player, since the same holds for everybody. We can think about this belief as the experience derived from enough past plays of the game, each time against different opponents, in such a way to form a belief related to a "typical" opponent behavior.

All the notions I introduced in this overview will be further defined and discussed in the rest of the document. The aim of this section was only to present them, in order to provide a clear picture of what will follow. In the next section I will describe the formal notation used in the rest of the document and use it to define the concept I just presented. Some of the most famous game example will be provided as well, and it will be possible to apply to them the solution concepts already introduced.

2.2 Strategic Games

The aim of this section is to provide formal definitions for all the concepts required in the rest of this thesis. In the previous section I introduced the notion of *strategic game*: a model of a decision-making process where different agents or *players* interact by choosing among their *actions* in order to achieve the best *outcome* according to their *preference relation*. This model fits the following definition:

Definition 2.2.1. A *strategic game* is a tuple $\langle N, (A_i), (\succsim_i) \rangle$ where:

- N is a finite set of *players*
- A_i is the non-empty set of *actions* available to player i
- $\succsim_i$ is the *preference relation* of player i over the set $A = \times_{j \in N} A_j$

The set $A = \times_{j \in N} A_j$ is the cartesian product among the sets of actions A_i of each player i. Thus, an element $a \in A$ is a tuple $a = (a_1, a_2, ..., a_N)$ consisting of an action for each player of the game.
Each element $a \in A$ is a possible *outcome* of the game, and we call it *action profile*. With an action profile we can indicate the action that each player chooses in the game. If the tuple $a = (a_1, a_2, ..., a_N)$ is meant in this way, we call it *pure strategy*.

The preference relation $\succsim_i$ of player i can be often represented by a *utility function* $u_i : A \to \mathbb{R}$ that associates a real value to every action profile, in such a way that $a \succsim_i b \implies u_i(a) \geq u_i(b)$. This utility function provides only ordinal information about the action profile set: if player i prefers the action profile a to b then his utility function's value in a is higher than in b.

Now that the concept of *strategic game* is formally defined, we can provide the most well-known example of strategic games. These games model a wide variety of real-world situations. Each game, after a brief verbal description,

will be expressed through a *payoff matrix*, that shows for each player and action, the associated utility value.

Prisoner's Dilemma

In its classical form, the game is presented as follows:
Two suspects, A and B, are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal. If one testifies (defects from the other) for the prosecution against the other and the other remains silent (cooperates with the other), the betrayer goes free and the silent accomplice receives the full 10-year sentence. If both remain silent, both prisoners are sentenced to only six months in jail for a minor charge. If each betrays the other, each receives a five-year sentence. Each prisoner must choose to betray the other or to remain silent. Each one is assured that the other would not know about the betrayal before the end of the investigation. How should the prisoners act?

In order to express this interaction through the model we previously presented, we must identify:

1. The players set N, that includes both the suspects A and B:
   $N = \{A, B\}$.

2. The action sets A_1 and A_2, respectively for suspect A and suspect B. These two sets contain the same actions, since each player can choose either to cooperate with the other being quiet C, or to defect from the coalition betraying the other D:
   $A_1 = \{C, D\}$ and $A_2 = \{C, D\}$.

3. The two preference relations $\succsim_1$ and $\succsim_2$, that must represent each suspect's preferences. For suspect A (a.k.a. player 1) we order the possible action profiles (a_1, a_2) in this way:

$$(D, C) \succsim_1 (C, C) \succsim_1 (D, D) \succsim_1 (C, D)$$

In the same manner we express $\succsim_2$:

$$(C, D) \succsim_2 (C, C) \succsim_2 (D, D) \succsim_2 (D, C)$$

As introduced before, we can express these ordinal preferences using a *utility function* u_i:
We may choose: $u_1(D, C) = 3$, $u_1(C, C) = 2$, $u_1(D, D) = 1$, and $u_1(C, D) = 0$.
In like manner: $u_2(C, D) = 3$, $u_2(C, C) = 2$, $u_2(D, D) = 1$, and $u_2(D, C) = 0$.

This game can be now represented through its payoff matrix, shown in Table 2.1. The two rows correspond to player 1's actions, and the two columns to player 2's ones. In each one of the 4 internal cells of the table we put the utility value of that action profile, respectively for player 1 and 2. For example in the internal cell [1,2] we find $u_1(C, D) = 0$ and $u_2(C, D) = 3$.

The *Prisoner's dilemma* game has been widely studied since it applies to many real situations: both players gain from cooperation, but each player has an incentive to defect from the coalition whatever the other player does.

	C (Quiet)	D (Betray)
C (Quiet)	2,2	0,3
D (Betray)	3,0	1,1

Table 2.1: Payoff matrix for the *Prisoner's Dilemma* game

Battle of Sexes

In the *Battle of Sexes* game (a.k.a. *BoS*) both players prefer to cooperate than not to cooperate but they disagree about the best outcome. The game's

general formulation is:
Two persons, Alice and Bob, would like to spend a day together and each one must decide, without knowing the decision of the other, where to go. Two options are available: a concert of Bob's favorite band and the premiere of Alice's most eagerly-awaited movie. Of course Bob prefers the concert and Alice prefers the premiere, but they both would mostly like to go together.
The strategic game that corresponds to this specification is:

1. *Players*: Alice (player 1) and Bob (player 2).
2. *Action sets*: For both the players, the possible actions are "concert" (C) or "premiere" (P).
3. Preference relations:

$$(C,C) \succsim_1 (P,P) \succsim_1 \begin{matrix}(C,P)\\(P,C)\end{matrix}$$

$$(P,P) \succsim_2 (C,C) \succsim_2 \begin{matrix}(P,C)\\(C,P)\end{matrix}$$

The situation is summarized in Table 2.2, where Alice chooses a row and Bob a column.

	C (Concert)	P (Premiere)
C (Concert)	1,2	0,0
P (Premiere)	0,0	2,1

Table 2.2: Payoff matrix for the *BoS* game

The Stag Hunt

This game describes a conflict between safety and social cooperation. It is also known as "trust dilemma" and was described by Jean-Jacques Rousseau in his *Discourse on the origin and foundations of inequality among*

men (1755) [23]. He discusses a group of hunter who wish to catch a stag and points out the problem with the following words:

> If it was a matter of hunting a deer, everyone well realized that he must remain faithful to his post; but if a hare happened to pass within reach of one of them, we cannot doubt that he would have gone off in pursuit of it without scruple...

This sentence leaded to the formulation of *the Stag Hunt* game as follows: Each hunter has the choice of hunting a hare or a stag. Each hunter prefers a share of the stag, to a hare. There is no chance of catching a stag by oneself but if a hunter pursues a hare, then he will be able to catch it. However if any hunter puts effort on pursuing a hare, the stag escapes.
The game's players and actions are clear. The preference relation, for a two player game, must state the following:

$$(Stag, Stag) \succsim_1 \begin{matrix}(Hare, Stag)\\(Hare, Hare)\end{matrix} \succsim_1 (Stag, Hare)$$

$$(Stag, Stag) \succsim_2 \begin{matrix}(Stag, Hare)\\(Hare, Hare)\end{matrix} \succsim_2 (Hare, Stag)$$

And the table of the game, according to one of the possible utility functions, is reported in Table 2.3.

	Stag	Hare
Stag	2,2	0,1
Hare	1,0	1,1

Table 2.3: Payoff matrix for the *Stag Hunt* game

Matching Pennies

This game is the first purely conflictual we present, since there is no cooperation aspect in it. It is the two strategy equivalent of *Rock, Paper,*

Scissors and it is commonly formulated as follows:
Each of the two players of the game shows either heads or tails from a coin. If both are heads or both are tails then player 1 wins, otherwise player 2 wins.
As shown in Table 2.4, a gain for one participant is always at the expense of another. This kind of games are called *strictly competitive* or *zero-sum*.

	Head	Tail
Head	1,-1	-1,1
Tail	-1,1	1,-1

Table 2.4: Payoff matrix for the *Matching Pennies* game

2.2.1 Best response functions

In this subsection I will introduce the concept of *best response function* of a player i. This will help for the further definition of other important notions such as *Nash Equilibrium*.
The best response is the strategy (or strategies) which produces the best (according to his preference relation) outcome for a player, taking other players' strategies as given. In order to formally define this concept we need to introduce, with the notation of a_{-i}, the list of elements of the action profile a, for all players except i:

$$a_{-i} = (a_j)_{j \in N \setminus \{i\}}$$

Definition 2.2.2. The *best response function* of player i is the function B_i which, for any a_{-i}, defines the set of player i's best actions given a_{-i}:

$$B_i(a_{-i}) = \{a_i \in A_i | (a_{-i}, a_i) \succsim_i (a_{-i}, a_i'), \forall a_i' \in A_i\} \tag{2.1}$$

We can provide some examples, related to the previously described games, in order to clarify the definition:

Example 2.2.1. *In the* Prisoner's Dilemma *the best response of player 1, to player 2's action* C (Quiet) *is* D(Betray)*, since in this way player 1 will earn the highest possible reward. For the same reason the best response to player 2's action* D*, is* D.
For BoS*, summarized in Table 2.2 we provide the two complete best response functions:*

$$B_1(P) = \{P\} \qquad B_2(P) = \{P\}$$
$$B_1(C) = \{C\} \qquad B_2(C) = \{C\}$$

2.2.2 Mixed strategy games

If we want to model an interaction when the participants' choices are not deterministic but are regulated by probabilistic rule we need to introduce the model of a *mixed strategy game*. This is a more general model which allows non-deterministic choices over the set A_i of each player. We will extend the definition of a pure strategy game $G = \langle N, (A_i), (u_i)\rangle$ to deal with these non-deterministic choices.
First of all the definition of *pure strategy* we provided before must be revised to fit such a concept.

Definition 2.2.3. Suppose a player i has M pure strategies (or actions), defined by the set $A_i = \{a_i^1, a_i^2, ..., a_i^M\}$. A *mixed strategy* α_i for player i is a probabilistic distribution over his pure strategies set A_i. That is, it is a probability tuple $(p_i^1, p_i^2, ..., p_i^M)$ with $p_i^k \geq 0, k = 1, ...M$ and $\sum_{k=1}^{M} p^k = 1$.

To better clarify the definition, a sample mixed strategy was provided in Example 2.1.1.
We indicate with $\Delta(A_i)$ the set of probability distributions over A_i, so a member of $\Delta(A_i)$ is a mixed strategy of player i: $\alpha_i \in \Delta(A_i)$.
A *mixed strategy profile* α is a strategy profile composed by the mixed strategy of every player: $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$. Given a mixed strategy profile $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ the probability that a particular action tuple $a = (a_1, a_2, ..., a_n)$ will occur, $p(a)$, is formed from the product of the probabilities assigned to $a = (a_1, a_2, ..., a_n)$ by α, that is $p(a) = \prod_{j \in N} \alpha_j(a_j)$. In other

words, α induces a probability distribution (lottery), over the set A.
Thus, we can now define another utility function U_i over the set of mixed strategies $\times_{j \in N} \Delta(A_j)$, using the utility function u_i of the corresponding pure strategy game G:

$$U_i : \times_{j \in N} \Delta(A_j) \to \mathbb{R}$$

$$U_i(\alpha) = \sum_{a \in A} p(a) u_i(a) \tag{2.2}$$

We now derive from G another strategic game:

Definition 2.2.4. The *mixed extension* of the strategic game $G = \langle N, (A_i), (u_i) \rangle$ is the strategic game $\langle N, (\Delta(A_i)), (U_i) \rangle$, where N, $\Delta(A_i)$ and U_i are defined above.

We will provide an example of mixed extension of a game and related mixed strategies while talking about the *Nash Equilibrium* in the following section.

2.3 Nash Equilibrium

This section describes formally the concept of *Nash Equilibrium* [16] we have already introduced in the first section. We will define different formulations of this concept that fit the different kinds of game we went through the previous section. The discourse will move ahead providing some basic theorems about the existence of an equilibrium in different game formulations. The last part presents a different formulation of this concept, in order to adapt it to deal with possible coalitions among players.

As we already explained, the concept of equilibrium aims to define a solution of a game, specifying a strategy profile that connotes a *steady state*. If everyone else adheres to this profile, no individual wishes to deviate from it. This notion assumes that each player holds the correct expectation about the other player's behavior, and acts rationally. The formalization of this concept is the following:

Definition 2.3.1. A *Nash equilibrium* of a strategic game $\langle N, (A_i), (\succsim_i)\rangle$ is an action profile $a^* \in A$ such that for every player $i \in N$:

$$(a^*_{-i}, a^*_i) \succsim_i (a^*_{-i}, a_i) \quad \forall a_i \in A_i$$

Equivalently, if the preference relation is defined through a utility function u_i, a *Nash equilibrium* is an action profile a^* such that, for every player $i \in N$:

$$u_i(a^*) \geq u_i(a_i, a^*_{-i}) \quad \forall a_i \in A_i \tag{2.3}$$

Thus a Nash Equilibrium is an action profile a^* with the property that no player i can get a better utility by choosing an action different from a^*_i, given that every other player chooses a^*_{-i}.
The definition (2.3) allows the strategy profile of the equilibrium a^* to be at least as good as another profile (a_i, a^*_{-i}), according to player i's utility function. This defines a *non-strict* equilibrium. On the contrary, in a *strict* Nash equilibrium the strategy profile is strictly better that any other profile,

according to each player's utility function.

The same definition can be written using the *best response function* of every player B_i, according to (2.1):

Definition 2.3.2. The action profile a^* is a *Nash equilibrium* of the strategic game G, if and only if every player's action a_i^* is a best response to the other player's actions a_{-i}^*:

$$a_i^* \in B_i(a_{-i}^*) \quad \forall i \in N \tag{2.4}$$

This definition implicitly suggests a way to find Nash equilibria of a game G:
Once computed the best response function of each player, we can find a strategy profile a^* such that it satisfies (2.4). The following example intends to clarify all the definitions and concepts defined so far:

Example 2.3.1. *The* BoS *game described in Table 2.2, has two Nash equilibria* (C, C) *and* (P, P)*. Let's examine the action profile* (C, C)*:*

- ***Player 1****: Assuming the action chosen by player 2 (a.k.a. Bob) as given* (C)*, player 1 (a.k.a Alice) does not wish to deviate from the action specified by the equilibrium* (C) *since the other action profile* (P, C) *would give her a utility of 0, versus the utility of 1 given by the equilibrium strategy.*

- ***Player 2****: The same situation holds for Bob. Assuming Alice's* C *as given, Bob does not wish to deviate from his equilibrium action* C*, since the possible deviation* (C, P) *would mean a payoff of 0 instead of 2.*

Note that both the equilibria are strict*.*
The same results comes from the utilization of the alternative formulation of Nash equilibrium with best response functions (BRFs). As already presented in Example 2.2.1, the BRFs of Alice and Bob are:

$$B_{Alice}(P) = \{P\} \qquad B_{Bob}(P) = \{P\}$$
$$B_{Alice}(C) = \{C\} \qquad B_{Bob}(C) = \{C\}$$

*In order to easily find the Nash equilibria we summarize the game in Table 2.5 and evidence the best responses of each player with an * over the corresponding utility value. The table clearly identifies the two actions profiles for which holds the (2.4). They are the two Nash equilibria of the game.*

	C (Concert)	P (Premiere)
C (Concert)	1*,2*	0,0
P (Premiere)	0,0	2*,1*

Table 2.5: Payoff matrix for the *BoS* game, and best responses

2.3.1 Cournot's model of oligopoly

In this subsection I present a key model that uses the notion of Nash equilibrium in order to study the competition among a small number of firms. This model is also useful to present sets of actions that are not finite. In fact, in all the examples presented so far, we dealt with finite action sets. In this subsection I show also how to analytically find Nash equilibria of a game studying continue best response functions.

In the general model proposed by Cournot in 1838 [4] there is a set N of firms producing the same product. Firms have market power, i.e. each firm i's output decision q_i affects the good's price $P(Q)$, where $Q = \sum_{i \in N} q_i$. P (a.k.a. *inverse demand function*) is a commonly known decreasing function of total output Q, since if the firms' total output increases it is likely that the product's price decreases. The cost to firm i to produce q_i units of the product is $C_i(q_i)$, where C_i is an increasing function.
We can now express the revenue r_i and the profit π_i of each firm i, using the

quantities defined above:

$$r_i(q_1, ..., q_n) = q_i P(q_1 + ... + q_n) = q_i P(Q)$$

$$\pi_i(q_1, ..., q_n) = r_i - C_i(q_i) = q_i P(q_1 + ... + q_n) - C_i(q_i)$$

Thus, the *Cournot's oligopoly game* is modeled as follows:

1. *Players*: The n firms.

2. *Action sets*: For each firm the action set is identified by the possible quantities of product that the firm can produce. Note that it can be a not finite set.

3. Preference relations: We can express the preference relation of a player i through a utility function that corresponds to the profit π_i.

The following particular instance of Cournot's oligopoly will allow us to easily study the best response functions in order to find the game's equilibria. We consider the special case of a duopoly: only two firms are competing among them. The unit cost is a constant c and the inverse demand function is linear and in the form:

$$P(Q) = \begin{cases} \alpha - Q & \text{if } Q \leq \alpha \\ 0 & \text{if } Q > \alpha \end{cases} \tag{2.5}$$

Thus the profit expressions of the two firms become:

$$\pi_1(q_1, q_2) = q_1 P(q_1 + q_2) - q_1 c = \begin{cases} q_1(\alpha - q_1 - q_2 - c) & \text{if } Q \leq \alpha \\ -cq_1 & \text{if } Q > \alpha. \end{cases}$$

$$\pi_2(q_1, q_2) = \begin{cases} q_2(\alpha - q_1 - q_2 - c) & \text{if } Q \leq \alpha \\ -cq_2 & \text{if } Q > \alpha. \end{cases}$$

In order to find the best response functions of both the firms we need to find the maximum of the profit as a function of the produced quantity of good. We can use calculus to find the derivative of each firm's profit in respect of

the produced quantity, set it equal to zero and solve for q_i:

$$br_1(q_2) = \begin{cases} \frac{1}{2}(\alpha - c - q_2) & \text{if } Q \leq \alpha \\ 0 & \text{if } Q > \alpha. \end{cases}$$

$$br_2(q_1) = \begin{cases} \frac{1}{2}(\alpha - c - q_1) & \text{if } Q \leq \alpha \\ 0 & \text{if } Q > \alpha. \end{cases}$$

The two best response functions $br_1(q_2)$ and $br_2(q_1)$ are shown in Figure 2.1.

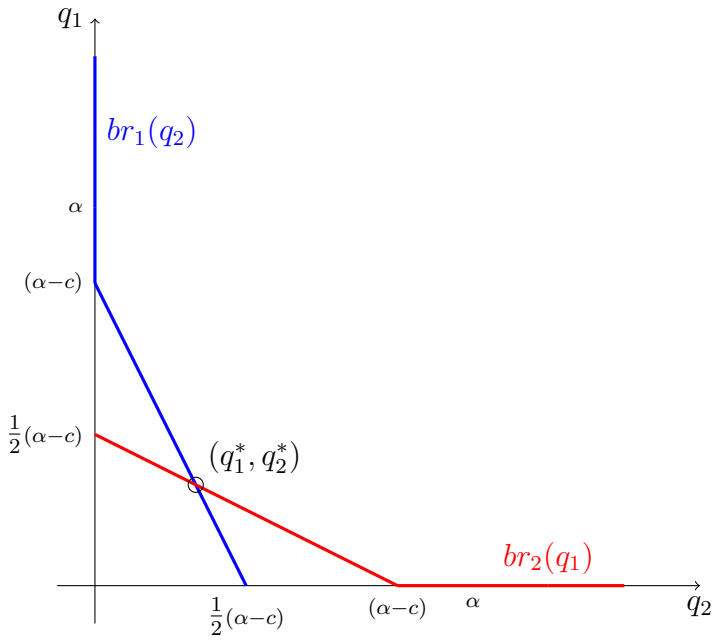


Figure 2.1: Best response functions $br_1(q_2)$ and $br_2(q_1)$

They are plotted relative to different axes, respectively q_2 and q_1. The strategies that satisfy the Nash Equilibrium definition given in Definition 2.3.2 are the set of points at which the two best response functions intersect. In this case there is exactly one such point (evidenced in Figure 2.1 by a small circle) which follows from the solution of the system of equations:

$$\begin{cases} q_1 = \frac{1}{2}(\alpha - c - q_2) \\ q_2 = \frac{1}{2}(\alpha - c - q_1) \end{cases}$$

The unique Nash Equilibrium of this game is (q_1^*, q_2^*) where $q_1^* = q_2^* = \frac{1}{3}(\alpha - c)$.

2.3.2 Mixed strategy Nash Equilibrium

The notion of *Nash equilibrium* can be also presented for the *Mixed extension of a strategic game G* we described in Definition 2.2.4. As stated before, the participants' choices are regulated by probabilistic rules.

Definition 2.3.3. A *mixed strategy Nash equilibrium* of a strategic game is a Nash equilibrium of its mixed extension.

Thus, since the actions of the mixed extension are mixed strategies α_i (elements of $\Delta(A_i)$), to express a mixed strategy equilibrium we have to specify a mixed strategy profile $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$.
It is possible to show that the set of Nash equilibria of a strategic game is a subset of its set of mixed strategy Nash equilibria. This holds because A_i can be identified with a subset of $\Delta(A_i)$: in fact we can consider a_i^* as a degenerate player i's mixed strategy α_i^* which assigns probability one to that single member of A_i.
In order to clarify the notion of Mixed strategy Nash Equilibrium we provide the following example:

Example 2.3.2. *The Matching Pennies game we presented in Section 2.2 does not possess a pure strategy Nash Equilibrium. Instead, the unique Nash equilibrium of this game is in mixed strategies: each player chooses heads or tails with equal probability. This mixed strategy action profile can be expressed as* $\alpha^* = (\alpha_1^*, \alpha_2^*)$ *where* $\alpha_1^* = (\frac{1}{2}, \frac{1}{2})$ *and* $\alpha_2^* = (\frac{1}{2}, \frac{1}{2})$. *We can calculate the value of the "extended" utility function defined in [2.2] as* $U_i(\alpha) = \sum_{a \in A} p(a) u_i(a)$ *where* $p(a) = \prod_{j \in N} \alpha_j(a_j)$. *Note that in this example* $p(a) = \frac{1}{4}$ $\forall\ a \in A$*:*

$$U_1(\alpha^*) = \frac{1}{4}(1) + \frac{1}{4}(-1) + \frac{1}{4}(1) + \frac{1}{4}(-1) = 0$$
$$U_2(\alpha^*) = \frac{1}{4}(-1) + \frac{1}{4}(1) + \frac{1}{4}(-1) + \frac{1}{4}(1) = 0$$

Note that this is a non strict mixed strategy Nash Equilibrium.

2.3.3 Basic existence theorems

In this subsection I will provide some major results concerning the existence of a Nash Equilibrium for particular kinds of games. At first I will introduce the Kakutani's fixed point theorem [8], that gives conditions under which there indeed exists a Nash Equilibrium in a game. Once this theorem is formulated, we can show an important result proved by John Nash, concerning the existence of a mixed strategy Nash Equilibrium for finite strategic games.
According to the definition given in 2.3.2, in order to show that a game has a Nash Equilibrium it suffices to show that there exist an action profile a^* such that $a_i^* \in B_i(a_{-i}^*) \quad \forall i \in N$. We can define a set valued function $B(a)$ as the cartesian product of the $B_i(a_{-i})$ of all the N players. The Kakutani's fixed point theorem gives conditions on B under which there exists a value of a^* such that $a^* \in B(a^*)$ and thus a^* is a Nash Equilibrium.

Theorem 2.3.1. (Kakutani's fixed point theorem) *Let X be a compact convex subset of $\mathbb{R}^n$ and let $f : X \to X$ be a set-valued function for which*

- *for all $x \in X$ the set $f(x)$ is nonempty and convex*
- *the graph of f is closed (i.e. for all sequences $\{x_n\}$ and $\{y_n\}$ such that $y_n \in f(x_n)$ for all n, $x_n \to x$, and $y_n \to y$, we have $y \in f(x)$).*

Then there exists $x^* \in X$ such that $x^* \in f(x^*)$.

We can use this theorem to assert some conditions on the elements of a strategic game as follows:

Proposition 2.3.2. *The strategic game $\langle N, (A_i), (\succsim_i) \rangle$ has a Nash Equilibrium if for all $i \in N$*

- *the set A_i of actions of player i is a nonempty compact convex subset of a Euclidean space*

and the preference relation $\succsim_i$ is

- *continuous*
- *quasi-concave on* A_i

Note that we define a preference relation $\succsim_i$ over A to be *quasi-concave on* A_i if for every $a^* \in A$ the set $\{a_i \in A_i : (a^*_{-i}, a_i) \succsim_i a^*\}$ is convex.
The Proposition 2.3.2 can be proved starting from the conditions on $\succsim_i$ and A_i and applying the fixed point theorem 2.3.1. This prove will be omitted.
Note that Preposition 2.3.2 does not hold for games with finite actions, since a finite A_i cannot be convex.
Using this preposition John Nash proved that:

Proposition 2.3.3. *Every finite strategic game* $G = \langle N, (A_i), (\succsim_i)\rangle$ *has a mixed strategy Nash Equilibrium.*

Proof. If we call m_i the cardinality of A_i we can identify the set $\Delta(A_i)$ (see Definition 2.2.4) with the set of vectors $(p_1, ..., p_{m_i})$ for which $p_k \geq 0$ for all k and $\sum_{k=1}^{m_i} p_k = 1$ (p_k being the probability with which player i uses his k_{th} pure strategy). This set is nonempty, convex, compact. Since expected payoff is linear in the probabilities (see 2.2 in Section 2.2.2), each U_i in the mixed extension of G is both quasi-concave in his own strategy and continuous. Thus the mixed extension of G satisfies all the requirements of Proposition 2.3.2. □

2.3.4 Symmetric games

A game in normal form is *symmetric* if all agents have the same strategy set, and the payoff to playing a given strategy depends only on the strategies being played, not on who plays them. Many well-known games are symmetric, the Prisoners' Dilemma described in Table 2.1 is an example. A formal definition follows:

Definition 2.3.4. A game in strategic form $G = \langle N, (A_i), (u_i)\rangle$ is *symmetric* if the players have identical strategy spaces ($A_1 = A_2 = ... = A_N$) and $u_i(a_i, a_{-i}) = u_j(a_j, a_{-j})$, for $a_i = a_j$ and $a_{-i} = a_{-j}$ for all $i, j \in \{1, ..., N\}$.

Thus we can write $u(t, a)$ for the utility to any player playing strategy t in profile a.

Definition 2.3.5. An action profile $a = (a_1, a_2, ..., a_N)$ in a symmetric game is a *symmetric profile* if all players play the same strategy ($a_1 = a_2 = ... = a_N$).
If such a profile is a Nash Equilibrium we call it *symmetric equilibrium.*

It can be proved that every symmetric game in two strategies has a pure strategy Nash Equilibrium. Furthermore it was proved by John Nash in 1951 [17] that:

Theorem 2.3.4. *every finite symmetric game has a symmetric mixed strategy Nash Equilibrium.*

2.4 Learning rules

In this section I will introduce a new concept of game. So far we thought about a game as a one-shot process. What the concept of Nash Equilibrium exactly mean if we consider an isolated play of the game? How can we apply this concept to real life situations? It is unnatural to consider a Nash Equilibrium as the outcome of a single play of a game. Instead, it should be thought as the limiting outcome of a learning or evolutionary process that unfolds over time. This concept is the rationale of *learning game theory*.
Learning game theory abandons the demanding assumptions of classical game theory on players' rationality and beliefs, and assumes instead that players learn over time about the game and about the behavior of others. The process of learning can take many different forms, depending on the available information, the available feedback, and *the way these are used to modify behavior*. The assumptions made in these regards give rise to different models of learning. In most models players use the history of the game to decide what action to take.
In this section I will present some of these models and I will discuss, using the results of some related works, whether or not these processes surely reach an equilibrium. Depending on the way these processes approach equilibria, learning models can suggest useful ways to modify and differentiate the traditional equilibrium definitions. Furthermore they can suggest an algorithmic way to find equilibria of complex games.

At first I will introduce the fundamental concept of *improvement path* and present some fundamental models of learning known in literature. The convergence properties of these algorithms to the equilibrium will be discussed afterwards through the presentation of some of the most important results concerning some classes of games: *Potential games*, *Congestion games* and *S-modular games*.

2.4.1 Better and Best reply dynamics

In these dynamics each player knows the actions that every player selected in the immediate past. When making his decision, each player assumes that every other player will behave as in the last play of the game. Thus he will choose an action that would lead to an improvement of his current payoff. The *best reply* dynamic differs from the *better reply* just in the fact that the choice of the action is made in order to get the highest possible payoff and not only a higher one.
In these models players respond to the environment in a *myopic* fashion, ignoring the implications of each choice on future choices and payoffs.
I will now provide formal definitions of these dynamics, considering a game in strategic form $G = \langle N, (A_i), (u_i) \rangle$. The following definitions were introduced in [15] and [13].

Definition 2.4.1. A *path* in A is a sequence $\gamma = (a_0, a_1, ...)$ such that for every $k \geq 1$ there exists a unique player i, such that $a_k = (a_{k-1}^{-i}, x)$ for some $x \neq a_{k-1}^i$ in A_i.
A path $\gamma = (a_0, a_1, ...)$ is an *improvement path* with respect to G if $\forall\ k \geq 1$ $u_i(a_k) > u_i(a_{k-1})$, where i is the unique deviator at step k. The *improvement path* defined above is a path generated by the *better reply dynamic*.

Definition 2.4.2. A path $\gamma = (a_0, a_1, ...)$ where $\forall\ k \geq 1$, a_k^i is a best response to a_{k-1}^{-i}, where i is the unique deviator at step k, is a path generated by the *best reply dynamic*.

Note that a best response path is a particular case of an improvement path. We are particularly interested in games where all the improvement paths are finite, since it is intuitive that in that case, for finite games, every maximal improvement path must terminate in an equilibrium point. Thus, the myopic learning process based on the one-sided better reply dynamic converges to the equilibrium set.
Formally we say that a game G has the *finite improvement property* (FIP) [15] (or is *1-acyclic* [13]) if every improvement path is finite.

In the sections that follow I will, amongst other things, present some classes of games that have been showed to have the finite improvement property and thus to converge to the equilibrium set.

2.4.2 Fictitious play

In this learning model each player assumes that every opponent is playing a certain mixed strategy. Every player builds an estimation of this mixed strategy, that we call *belief*, taking into account not only the actions in the last play, but the whole history of the game. He plays a myopic pure best response against the empirical strategy distribution of his opponents. This concept was originally introduced by Brown [3]. In order to formally define this process we need first to define a best response to the mixed strategy profile α_{-i}:

Definition 2.4.3. In a game $G = \langle N, (A_i), (u_i) \rangle$, we call $BR_i(\alpha_{-i})$ the pure best response to the mixed profile α_{-i}, according to the extended utility function $U_i : \times_{j \in N} \Delta(A_j) \to \mathbb{R}$ defined in (2.2).

We can now go on and define the first learning model, the *simultaneous fictitious play* (the following is an extension for N players of the definitions given by Berger in [2]):

Definition 2.4.4. For the game $G = \langle N, (A_i), (u_i) \rangle$, the sequence $(a_1^t, a_2^t, ..., a_N^t)_{t \in \mathbb{N}}$ is a *simultaneous fictitious play process* if $(a_1^1, a_2^1, ..., a_N^1) \in A$ and for all $t \in \mathbb{N}$ and for all $i \in N$,

$$a_i^{t+1} \in BR_i(\alpha_{-i}^t) \tag{2.6}$$

where the *beliefs* α_i^t are given by

$$\alpha_i^t = \left(\frac{t-1}{t}\right) \alpha_i^{t-1} + \left(\frac{1}{t}\right) a_i^t \tag{2.7}$$

Note that in this process every player updates his belief simultaneously with the other players. We can now provide the definition of the process

originally described by Brown in [3], where players update their beliefs *alternatingly*. I only report the definition of a game with two players, in order to discuss the results showed by Berger in [2]:

Definition 2.4.5. For the game $G = \langle N, (A_i), (u_i) \rangle$, where $N = \{1, 2\}$, $|A_1| = n$ and $|A_2| = m$ the sequence $(a_1^t, a_2^t)_{t \in \mathbb{N}}$ is an *alternating fictitious play process* (AFP) if $(a_1^1) \in A_1$ and for all $t \in \mathbb{N}$,

$$(a_1^{t+1}) \in BR_1(\alpha_2^t) \text{ and } (a_2^t) \in BR_2(\alpha_1^t)$$

where the *beliefs* $\alpha_i(t)$ are given by

$$\alpha_i^{t+1} = \left(\frac{t}{t+1}\right) \alpha_i^t + \left(\frac{1}{t+1}\right) a_i^{t+1}$$

First of all note that if a fictitious play process converges, it must be constant from some stage on, implying that the limit is a pure Nash equilibrium. Even if the process does not converge, it is easily established that if the beliefs converges then the limit must be a (non necessarily pure) Nash equilibrium.
Before introducing the work done by Berger in [2] we need to define the class of non-degenerate games, since many results are shown to hold only for this class. We only need to deal with two players games:

Definition 2.4.6. We call a game $G = \langle N, (A_i), (u_i) \rangle$, where $N = \{1, 2\}$, *degenerate*, if for some $a_1, a_1^{'} \in A_1$, with $a_1 \neq a_1^{'}$, there exists $a_2 \in A_2$ with $u_1(a_1, a_2) = u_1(a_1^{'}, a_2)$ or if for some $a_2, a_2^{'} \in A_2$, with $a_2 \neq a_2^{'}$, there exists $a_1 \in A_1$ with $u_1(a_1, a_2) = u_1(a_1, a_2^{'})$. Otherwise the game is said to be *non-degenerate*.

Berger proved the following [2]:

Theorem 2.4.1. *If an AFP process for a non-degenerate game contains in the sequence* $(a_1^t, a_2^t), (a_1^{t+1}, a_2^{t+1})$ *at generic time t, where* $a_1^t \neq a_1^{t+1}$ *or* $a_2^t \neq a_2^{t+1}$*, then there is an improvement path from* (a_1^t, a_2^t) *to* (a_1^{t+1}, a_2^{t+1}).

Barely speaking AFP processes essentially follow improvement paths in non-degenerate games. So if a game has the FIP, then the AFP cannot involve cycles, and must terminate in a pure Nash equilibrium.
Note that the same does not hold for the SFP process, since if both players switch at the same time, there need not be an improvement path from the old to the new pure-strategy profile.
It is now useful to formalize the convergence property of the fictitious play process in a game. We call it *FP property* from [13].

Definition 2.4.7. A FP process $(a_1^t, a_2^t, ..., a_N^t)_{t\in\mathbb{N}}$ *approaches equilibrium*, if for every $\varepsilon > 0$ there exists $t_0 > 0$, such that for every $t \geq t_0$, there exists a mixed equilibrium profile $p \in \times_{j\in N}\Delta(A_j)$, such that

$$|(\alpha_1^t, \alpha_2^t, ..., \alpha_N^t) - p| < \varepsilon$$

Thus we say that a game has the *FP property*, if every FP process, independently of initial actions and beliefs, approaches equilibrium.

We know from the literature of some class of games with the FP property:

- Zero-sum games, i.e. where $N = \{1, 2\}$ and $u_1(a) = -u_2(a)$ for every $a \in A$ (Robinson [21]).
- Games with identical payoff functions, i.e. $u_i(a) = u_j(a)$ for every $i, j \in N$ (Monderer and Shapley [14]).
- Games which are dominance solvable (Milgrom and Roberts [12])
- Games with strategic complementarities and diminishing returns (Krishna [9])
- 2x2 non-degenerate games (Monderer and Shapley [14])
- 2x3 non-degenerate games (Monderer and Sela [13]). In this case only the Continuous FP property has been proved. Although it is not proved, *it seems* that FP paths behave similarly to the continuous ones.

In the following sections I will discuss about other particular classes of games which have been showed to have the FP or the FIP property.

2.4.3 Adaptive play

This dynamic has been described for the first time by Young in [25], where he focused on the unfeasibility of the infinite history assumption in fictitious play. He depicted a model where each player knows only a small portion of the history, and she bases her action on a sample of plays from recent time periods. Young called this process *Adaptive play* and found some convergence results related to some class of games (i.e *weakly acyclic games*) under certain assumptions. In his analysis he also assumed that players can make mistakes when choosing their response, and investigated about how this assumption affect convergence.
What we need on this thesis is his formulation of *Adaptive play* for a game $G = \langle N, (A_i), (u_i) \rangle$:

Definition 2.4.8. We call $s(t) = (s_1(t), s_2(t), ..., s_n(t))$, which is a strategy tuple where $s_i(t) \in A_i$ and t is a time instant, a *play at time t*.
Then the *history* of plays up to time t is $h(t) = (s(1), s(2), ..., s(t))$.

Each player of an *Adaptive play*, in period $t + 1$, samples k plays drawn without replacement from the most recent m periods, where $1 \leq k \leq m$. Note that the sequence of plays containing the most recent m periods is $l(t) = (s(t-m+1), ..., s(t-1), s(t))$. Note furthermore that $l(t)$ fully describes the state of the process at time t.
Once having his k samples, each player plays a *best-reply* at time $t + 1$.

Definition 2.4.9. The process descibed above is an *Adaptive play with memory m and sample size k*.

Young does not specify any sampling probability, his only assumption is that, on every iteration, every subset of k elements from $l(t)$ has a positive probability of being player i's information $\forall i \in N$.
He assumes though that the sampling process begins in period $t = m + 1$, with some initial arbitrary sequence of m plays $h(m) = (s(1), s(2), ..., s(m))$.

Young observes moreover that a state l is an absorbing state of the process if and only if consists of a pure strategy Nash Equilibrium played m times. Formally:

Theorem 2.4.2. *$l = (s^1, s^2, ..., s^m)$ is an absorbing state if and only if $s^j = s$ $\forall j \in (1, 2, ..., m)$ and s is a pure strategy Nash Equilibrium.*

I will use this condition while writing the *non-cooperative game simulator* in order to understand when the *Adaptive play* reaches and equilibrium and thus when to stop the process.

2.4.4 Potential games

I will now define a new class of games called *Potential games*. This type of games has shown various properties related to the FP process and the best reply dynamics. I will provide some definition of different types of *potentials* and I will present the main results concerning the convergence of the learning processes introduced above. I will also depict a way to algorithmically detect if a game has a potential. The following definitions are reported in [15]:

Definition 2.4.10. A function $P : A \to \mathbb{R}$ is an *ordinal potential* for the game $G = \langle N, (A_i), (u_i) \rangle$, if for every $i \in N$ and for every $a_{-i} \in A_{-i}$:

$$u_i(a_{-i}, a_i^{'}) - u_i(a_{-i}, a_i^{''}) > 0 \Leftrightarrow P(a_{-i}, a_i^{'}) - P(a_{-i}, a_i^{''}) > 0$$

for every $a_i^{'}, a_i^{''} \in A_i$.

G is called an *ordinal potential game* if it admits an ordinal potential. Clearly, the pure strategy equilibrium set of the game coincides with the pure-strategy equilibrium set of the game in which every player utility function is given by P. If we want to extend this concept to the mixed strategy equilibrium set we need a stronger definition of potential:

Definition 2.4.11. A function $P : A \to \mathbb{R}$ is a *w-potential* for the game G, if for every $i \in N$ and for every $a_{-i} \in A_{-i}$:

$$u_i(a_{-i}, a_i^{'}) - u_i(a_{-i}, a_i^{''}) = w_i \left(P(a_{-i}, a_i^{'}) - P(a_{-i}, a_i^{''}) \right)$$

for every $a_i^{'}, a_i^{''} \in A_i$. Where $w = (w_i)$ is a vector of positive numbers called *weights*.

Coherently G is called a *w-potential game* if it admits a w-potential. Furthermore G is called an *exact potential game* (or simply a *potential game*) if it admits an *exact potential*, that is a w-potential for G with $w_i = 1$ for every $i \in N$. In the next example, also present in [15], a potential function P is shown :

Example 2.4.1. *Given the following Prisoners' Dilemma game G, the function P is a potential for G:*

$G =$

	C	D
C	1,1	9,0
D	0,9	6,6

P=

	C	D
C	4	3
D	3	0

If P is an ordinal potential function for $G = \langle N, (A_i), (u_i) \rangle$, then the equilibrium set of G coincides with the equilibrium set of $G = \langle N, (A_i), P \rangle$. That is, $a \in A$ is an equilibrium point for G if and only if for every $i \in N$

$$P(a) \geq P(a_{-i}, a_i^{'}) \text{ for every } a_i^{'} \in A_i. \tag{2.8}$$

Consequently, if P admits a maximal value in A, then G possesses a pure-strategy equilibrium. Thus, every finite ordinal potential game possesses a pure-strategy equilibrium. In [15] Monderer and Shapley proved that

Theorem 2.4.3. *Every finite ordinal potential game has the FIP.*

To prove the opposite we need a further assumption on the game utility function:

Theorem 2.4.4. *Let G be a finite game with the FIP. Suppose in addition that for every $i \in N$ and for every $a_{-i} \in A_{-i}$*

$$u_i(a_{-i}, a_i^{'}) \neq u_i(a_{-i}, a_i^{''}) \text{ for every } x \neq z \in A_i$$

then G has an ordinal potential

Note that the further assumption is just the non-degeneration of the game G.
From Theorem 2.4.3 and from the conclusions we drew when we talked about the better reply dynamic, we can now state that *the myopic learning process based on the one-sided better reply dynamic always converges to the equilibrium set in finite ordinal potential games.* According to the Fictitious Play learning process, Monderer and Shapley proved in [14] the following:

Theorem 2.4.5. *Every finite weighted potential game has the Fictitious Play property*

So this class of games can be added to the list we formulate at the end of Section 2.4.2. Another result concerning fictitious play and non-degenerate ordinal potential games was proved by Berger in [2]:

Theorem 2.4.6. *Let $G = \langle N, (A_i), (u_i)\rangle$, where $N = \{1, 2\}$, be a non-degenerate ordinal potential game. Then every AFP process converges to a pure Nash equilibrium.*

Note that this result holds just for the alternate fictitious play process, so we cannot say anything about the FP property of this class of games.
Now I will report the method depicted by Monderer and Shapley in [15] for detecting potential games. The prove is present in the related article. I will omit the intuitive definitions of *closed path*, *simple closed path* and *length of a path*.
For a finite path $\gamma = (a_0, a_1, ..., a_N)$ and for a vector $\upsilon = (\upsilon_1, \upsilon_2, ..., \upsilon_n)$ of functions $\upsilon_i : A \to \mathbb{R}$, we define

$$I(\gamma, \upsilon) = \sum_{k=1}^{n} [\upsilon_{i_k}(a_k) - \upsilon_{i_k}(a_{k-1})]$$

where i_k is the unique deviator at step k. If we consider a game $G = \langle N, (A_i), (u_i)\rangle$ and we compute $I(\gamma, u)$ we obtain the sum of the improvements of the utility values of all the unique deviators in the path.
Now I can report the method to establish if a game has a potential:

Theorem 2.4.7. *Let G be a game in a strategic form. Then the following claims are equivalent:*

1. *G is a potential game.*
2. *$I(\gamma, u) = 0$ for every finite closed path γ.*
3. *$I(\gamma, u) = 0$ for every simple finite closed path γ.*
4. *$I(\gamma, u) = 0$ for every finite closed path γ of length 4.*

So in order to understand if a game has a potential we need to compute $I(\gamma, u)$ for every closed path of length 4.

2.4.5 Congestion games

In this section I present another class of games and I show, following [15], that this class coincides with the class of potential games I just presented. Intuitively, a congestion game takes place when some agents have to choose from a finite set of alternatives, and the payoff of a player depends on the number of players choosing each alternative. Formally a congestion game is defined as follows:

Definition 2.4.12. A *congestion model* is a tuple $C = \langle N, M, (A_i)_{i \in N}, (c_j)_{j \in M} \rangle$ where:

- N denotes the set of players $\{1, 2, ..., n\}$
- M denotes the set of facilities $\{1, 2, ..., m\}$
- A_i for $i \in N$ denotes the set of strategies of player i, where each $a_i \in A_i$ is a nonempty subset of facilities.
- $c_j \in \mathbb{R}^{\{1,2,...,n\}}$ for $j \in M$ denotes the vector of payoffs, where $c_j(k)$ denotes the payoff to each user of facility j, if there are exactly k users.

The *congestion game* associated with the congestion model is the game in strategic form $G = \langle N, (A_i), (u_i) \rangle$ with u_i defined as follows:
For every action profile $a \in A$, where as usual $A = \times_{j \in N} A_j$, and for every $j \in M$ let $n_j(a)$ be the number of users of facility j. That is:

$$n_j(a) = \#\{i \in N : j \in a_i\}, \text{ where } a = (a_1, a_2, ..., a_n).$$

Define $u_i : A \to \mathbb{R}$ as

$$u_i(a) = \sum_{j \in a_i} c_j(n_j(a)). \tag{2.9}$$

I can provide an example of game which fits the previous definition of congestion game.

Example 2.4.2. *There are two drivers ($N = \{1, 2\}$) who have to go from a point, A, to another, B, using 5 different road segments ($M = \{1, 2, 3, 4, 5\}$). Every road segment has a cost for the users who decide to use it, which grows with the number of users actually using it. So the payoff of every road segment can be defined as the negative of this cost. In this case $c_1(2)$ denotes for example the payoff a user would get using the road segment 1, assuming that this segment is used by two users (him and another one). A strategy a_i in the set A_i of strategies of player i, is a subset of the road segments which allows player i to go from A to B. The utility function is calculated as in (2.9) and represents the sum of the payoffs associated with the segments that player i is choosing in the action profile a.*

Monderer and Shapley deduce in [15] the following two theorems from Rosenthal [22]:

Theorem 2.4.8. *Every congestion game is a potential game.*

Theorem. *Every finite potential game is isomorphic to a congestion game.*

Note that every result regarding the convergence of learning processes in finite potential games I presented in the sections above also holds for

congestion games. Showing that the class of congestion games coincides (up to an isomorphism) with the class of finite potential games, Monderer and Shapley in [15] proved Theorem 2.4.8 providing also the equation of a potential for a congestion game:
Let $C = \langle N, M, (A_i)_{i \in N}, (c_j)_{j \in M} \rangle$ be a congestion game, for each $a \in A$ define

$$P(a) = \sum_{j \in \cup_{i=1}^{n} a_i} \left(\sum_{l=1}^{n_j(a)} c_j(l) \right) \tag{2.10}$$

P is a potential for G.
The previous equation can be very useful when searching for equilibrium points of a congestion game.

Following the previous definition we can define a more general class of congestion games, called *player specific congestion games*, simply stating that in this class of games every players has her own vector of payoffs $c_j^i \in \mathbb{R}^{\{1,2,...,n\}}$,where $c_j^i(k)$ denotes the payoff to user i using facility j, given that there are exactly k users served from that facility.
The utility equation reported in (2.9), becomes:

$$u_i(a) = \sum_{j \in a_i} c_j^i(n_j(a)). \tag{2.11}$$

Important results were found also examining the so called *matroid congestion games* [1], where the strategy space of each player consists of the bases of a matroid on the set of facilities. These two classes of games have particular importance for this thesis, since I will show they are equivalent to particular subclasses of the *Replication games* class I will meanly deal with in the following sections.
I will now provide the formal definition of matroid and matroid congestion game and present the main results related to these classes.

Definition 2.4.13. A tuple $\mathcal{M} = (M, l)$ is a *matroid* if $M = \{1, ..., m\}$ is a finite set of resources and l is a non-empty family subsets of M such that:

- if $I \in l$ and $J \subseteq I$, then $J \in l$, and

- if $I, J \in l$ and $|J| < |I|$, then there exists an $i \in I \setminus J$ with $J \cup \{i\} \in l$.

If $I \subseteq M$ and $I \in l$, then I is an independent set. A maximal independent set is called a basis of $\mathcal{M}$.
Now we have all the required definitions to define the related game:

Definition 2.4.14. A congestion game $C = \langle N, M, (A_i)_{i \in N}, (c_j)_{j \in M} \rangle$ is a *matroid congestion game* if for every player $i \in N$, $\mathcal{M}_i := (M, l_i)$ with $l_i = \{I \subseteq S | S \in A_i\}$ is a matroid and A_i is the set of bases of $\mathcal{M}_i$

These definition will be very useful later, while examining the properties of *Replication games*. Right now I will present the known main results related to these two class of games.

In 1996 Milchtaich [11] showed the results of his study about player-specific congestion games introducing however two assumptions:

- each player does not choose a nonempty subset of M, but only one facility
- the payoff function of a player choosing j is non-increasing with the number of players selecting the same facility $n_j(a)$.

Under these assumption he proved the following:

- Every game in that class possesses a Nash equilibrium in pure strategies.
- Every game in that class involving only two strategies (i.e $m = 2$) possesses the finite improvement property (FIP, see Section 2.4.1).
- Every game in that class involving only two players possesses the finite best-reply improvement property (FBRP, i.e no cycles in paths generated by a best-reply dynamic).
- Every game in that class is *weakly acyclic* (i.e every strategy tuple is connected to a pure NE by a best-reply path)

The work of Milchtaich has been extended by Ackerman et al. [1]. They removed the constraint on the single facility and they introduced a more relaxing requirement (the matroid requirement) on the strategy space (see Definition 2.4.14). They proved the following:

- Every player-specific matroid congestion game with non-increasing pay-off functions possesses at least one Nash equilibrium in pure strategies.
- Every game in the same class above is *weakly acyclic in better responses* (i.e every strategy tuple is connected to a pure NE by a better-reply path).

I will further refer to the results listed above in Chapter 3.

Chapter 3

Replication Games

In this chapter I present a new class of games called *Replication games*. I also describes the formal analysis I performed on them, and the results I achieved.

In a replication game players are arranged on a social graph. Thus each player has a set of neighbors. Each player has a private storage which she can use to replicate some objects (or parts of them) from a common set. Each player can also access objects located in the storage of her neighbors. Accessing an object from her own storage, from a neighbor's storage or from the common set has different costs.

This class of games, whose formal definition will be introduced in a few lines, is particularly feasible to model network cooperation among Internet Service Providers or caching proxies. Here players generally need to decide which objects they should put in their cache in order to minimize the overall cost of their network traffic. In [5] Dán provides an example of such games, together with their definition and an important result over the existence of a pure strategy Nash Equilibrium. Here the game concerns a group of ISPs performing cooperative caching of peer-to-peer content. The goal of the analysis is to understand if it is worth for the players to cooperate, only accordingly to their selfish interests and hence their utility.

3.1 Formal definition

Formally a replication game is defined by a tuple $C = \langle N, (\mathcal{R}_i), (u_i) \rangle$ where, as usual, N is the set of players and u_i is the utility function of player i. $\mathcal{R}_i$ represents the action set of player i previously called A_i.
Given a common set of objects O of size m, in a replication game we represent the action of player i, and thus the set of objects she replicates, using a vector $r_i = (r_i^1, ..., r_i^m)$ of size m, whose component $r_i^o \in \{0, 1\}$ is set equal to 1 if object o has been replicated.
In order to enumerate the possible actions of a player, and thus to build her action set, we need to introduce a capacity constraint that limits the players' storage. If we represent with S^o the size of object o and with K_i the storage size of player i, the action set will be $\mathcal{R}_i = \{r_i | \sum_{o \in O} r_i^o S^o \leq K_i\}$.
We can now express the utility u_i as the sum of the cost savings U_i^o that player i achieves replicating the objects in her storage according to r_i, given the strategies of the other players r_{-i}:

$$u_i(r_i, r_{-i}) = \sum_{o \in O} U_i^o(r_i^o, r_{-i}^o) \tag{3.1}$$

where

$$U_i^o(r_i^o, r_{-i}^o) = r_i^o w_i^o \left[\beta_i (1 - \prod_{j \in \mathcal{N}(i)} (1 - r_j^0)) + \gamma_i \prod_{j \in \mathcal{N}(i)} (1 - r_j^0) - \alpha_i \right] \tag{3.2}$$

Note that we express with $\mathcal{N}(i)$ the set of neighbors of player i according to the social graph $G = (N, E)$. Furthermore α_i, β_i and γ_i indicate the cost for player i to access data respectively from her storage, a neighbor's storage and directly from the set of common objects.

3.2 Previous results

We find a detailed study of the *Replication game* class' properties in [6]. Here Dán presents his results about existence of equilibria and convergence

of different dynamics.
At first he proves that every replication game has a pure strategy NE. He then analyzes the behavior of the *better reply* dynamic in the synchronous, asynchronous and plesiochronous case.
Within the plesiochronous dynamics, he allows a player to make an improvement step only if no neighboring player makes a step at the same time. This is allowed because the actions of a player in a *Replication game* depends, within a step, only on his neighborhood.
Introducing the concept of *finite lazy improvement step* he proves that for $\beta_i \leq \alpha_i$ every asynchronous and plesiochronous replication game possesses the *lazy finite improvement property* (FLIP).
The *finite lazy improvement step* is an improvement step as we defined in the above sections, but the response of each player is such that it introduces the minimum amount of changes from the previous action profile.
The FLIP for games with $\beta_i > \alpha_i$, where the retrieval of an object from a neighbor's cache is more costly then the retrieval for one's own cache, is still an open issue. For this reason in this dissertation I will mostly assume $\beta_i > \alpha_i$.

3.3 Equivalence to congestion games

Now I will list some properties of some particular subsets of this class by showing their equivalence with some subclasses of the congestion games. In order to better clarify the meaning of equation (3.2) and to show the equivalence of a particular subclass of these games, we deeper analyze the utility function equations, (3.1) and (3.2). Note that in (3.2) the results of the two equal inner products can be equal only to 0 or 1. Thus the multiplicative factors of β_i and γ_i are complementary.
We can distinguish two cases:

- Object o is not replicated by any neighbor of player i:

$$U_i^o(r_i^o, r_{-i}^o) = r_i^o w_i^o \left[\gamma_i - \alpha_i\right]$$

- Object o is replicated by at least one neighbor of i:

$$U_i^o(r_i^o, r_{-i}^o) = r_i^o w_i^o \left[\beta_i - \alpha_i\right]$$

If player i does not replicate object o (i.e $r_i^o = 0$), the cost saving (3.2) related to that object is 0. Furthermore note that the utility of player i is only affected by the actions of the players in her neighborhood $\mathcal{N}(i)$.
Following these considerations we can rewrite the equation of the utility function (3.1). First of all we express the strategy r_i of player i as the set of objects she replicates: $s_i = \{o | r_i^o = 1\}$. The utility becomes:

$$u_i(s_i, s_{-i}) = \sum_{o \in s_i} U_i^o(s_{-i}) \tag{3.3}$$

where

$$U_i^o(s_{-i}) = \begin{cases} w_i^o \left[\gamma_i - \alpha_i\right] & \text{if } o \notin \bigcup_{j \in \mathcal{N}(i)} s_j \\ w_i^o \left[\beta_i - \alpha_i\right] & \text{if } o \in \bigcup_{j \in \mathcal{N}(i)} s_j \end{cases} \tag{3.4}$$

We will now introduce a particular assumption that will allow us to show the equivalence between the resulting class of replication games and congestion games.
We assume from now on that the social graph underlying the replication game is (*undirected* and) *complete*. The effect of this constraint is a simplification of the utility function. Each player's cost saving depends now on the set of object replicated by all the other players, not only by the neighborhood.

Theorem 3.3.1. *A replication game played on a complete graph is a player-specific congestion game.*

Proof. We call S_{-i} the set of replicated objects among all the players but i, defined as $S_{-i} = \bigcup_{j \in N \setminus \{i\}} s_i$. The conditions in the cost saving expression become respectively $o \notin S_{-i}$ and $o \in S_{-i}$. We now call n_o the number of players that replicate object o. Observe that the two conditions in the expression of the cost saving (3.4) can be rewritten as, respectively, $n_o = 1$ and $n_o > 1$. The first represents the case where no other player except i is

replicating object o, the latter represent all the cases where somebody else, other than player i, is replicating the object.
Here follows the final expression of the utility function:

$$U_i^o(n_o) = \begin{cases} w_i^o [\gamma_i - \alpha_i] & \text{if } n_o = 1 \\ w_i^o [\beta_i - \alpha_i] & \text{if } n_o > 1 \end{cases}$$

The game is now expressed as a player-specific congestion game (see Section 2.4.5) $C = \langle N, M, (A_i)_{i \in N}, (c_j^i)_{i \in N, j \in M} \rangle$ where:

- $M = O$;
- $A_i = \{I \subseteq O | \sum_{o \in I} S^o \leq K_i\}\}$
- The vector of payoffs is:

$$c_j^i(n_j(a)) = \begin{cases} w_i^j [\gamma_i - \alpha_i] & \text{if } n_j(a) = 1 \\ w_i^j [\beta_i - \alpha_i] & \text{if } n_j(a) > 1 \end{cases} \tag{3.5}$$

□

Note that the each component of the vector of payoffs c_j^i is a step and, assuming that all the weights are positive ($w_i^o > 0$, $\forall i \in N, o \in O$), we can easily impose that each component is a non-increasing function of $n_j(a)$:

$$w_i^j(\beta_i - \alpha_i) \leq w_i^j(\gamma_i - \alpha_i) \Rightarrow \beta_i \leq \gamma_i \tag{3.6}$$

This constraint is particularly feasible for a replication game, since it states that player i spends less to access an object replicated in a neighbor's cache rather than accessing it directly form the common set of objects.
If we introduce another constraint on the replication game class we can prove its equivalence with the matroid congestion games introduced by Ackermann [1] and presented in Definition 2.4.14.

Theorem 3.3.2. *A replication game played on a complete graph where players always fill up their caches and $S^o = c$, $\forall o \in O$, is a player-specific matroid congestion game.*

Proof. We already showed the equivalence of a replication game on a complete graph with a player-specific congestion game. We still need to show that the strategy space of each player consists of the bases of a matroid on the set of facilities.
We consider that the action set of player i, $\mathcal{S}_i$, is a family of the subsets of O which fulfill the cache capacity constraint.
Formally we need to show that, for every player $i \in N$, $\mathcal{M}_i := (O, l_i)$ with $l_i = \{I \subseteq S | S \in \mathcal{S}_i\}$ is a matroid and $\mathcal{S}_i$ is the set of bases of $\mathcal{M}_i$.
Since each player always fills up her cache, and every object has the same size c, $\mathcal{S}_i$ is the family of the subsets of O with same size: $\mathcal{S}_i = \{S \subseteq O$ such that $|S| = \lfloor \frac{K_i}{c} \rfloor\}$.
Consider now the following construction of a *uniform matroid*: Let O be a finite set, and $n \in \mathbb{N}$ (in our case $n = \lfloor \frac{K_i}{c} \rfloor$). One may define a matroid on O by taking every n-element subset of E to be a basis.
This builds up a uniform matroid that agrees with our definition of $\mathcal{M}_i$. □

According to the just presented results, given the right assumptions, many theorems presented in Section 2.4.5 also hold for Replication games. Here follows the list of results and specific assumptions, reformulated for Replication games. Keep in mind that the full graph assumption must hold in every case:

- If $c_j^i(n_j(a)) = c_j(n_j(a))$ (i.e not player-specific games), the game possess at least a pure Nash equilibrium, and is a potential game, thus has the FIP (Rosenthal [22]).

In this case we can also provide the potential function. We already reported the expression of the potential for congestion games in (2.10). Note that the value of $c_j(l)$ is expressed in (3.5): if we take into account the hypotesis of not player-specificity, it becomes:

$$c_j(l = 1) = w_1^j [\gamma_1 - \alpha_1] = w_2^j [\gamma_2 - \alpha_2] = .. = E^j \tag{3.7}$$

$$c_j(l > 1) = w_1^j [\beta_1 - \beta_1] = w_2^j [\beta_2 - \beta_2] = .. = F^j \tag{3.8}$$

we can rewrite the expression of the potential as:

$$P(a) = \sum_{j \in \bigcup_{i=1}^{n} a_i} \Big(E^j + F^j \cdot (n_j(a) - 1) \Big) \tag{3.9}$$

All the following statements hold for games with non-increasing payoff functions (see condition (3.6)) and where each player can choose just one object $o \in O$ to replicate:

- They possess at least one pure Nash equilibrium.
- They are *weakly acyclic in best responses*
- If $|O| = 2$, they possess the finite improvement property (FIP).
- if $|N| = 2$, they possess the finite best-reply improvement property (FBRP).

It is also worth a mention that the improvement cycle showed by Milchtaich in [11] on player-specific congestion games with non-increasing payoff functions, with cache limited to one object, and $|O| > 2$, is not possible to recreate in correspondent Replication games, because of they particular utility. Thus is not yet proved that they actually do not have the FIP.
Instead, the following hold for games with non-increasing payoff functions which are also matroid replication games (see Theorem 3.3.2), thus we relax the constraint of the replication of at most one object:

- They possess at least one pure strategies Nash equilibrium.
- They are *weakly acyclic in better responses.*

Note that the weakly acyclicity allows to state that if the players play *best* (or *better* in the matroid case) responses in a random order, then the expected number of iterations needed to reach an equilibrium is finite.

3.4 Further results

In order to introduce further results related to replication games I need to make some remarks about the way the utility function of a player changes in a better-reply process.
Right before i put some constraints on α_i, β_i and γ_i. Since Dán in [6] proved his results under the assumptions of $\beta_i \leq \alpha_i$ (see Section 3.2), I only consider the complementary case where $\beta_i > \alpha_i$.
Note that when $\gamma_i \geq \beta_i$, and thus the payoff function is non-decreasing (3.6), $\delta_i = \frac{\gamma_i - \alpha_i}{\beta_i - \alpha_i}$ is alway greater than 1. All the considerations that follow, are made under the assumption of non-decreasing payoff function (3.6), particularly feasible for a replication game.
Note the following remarks about the best-reply process:

1. If player i earns a higher payoff by replacing and object o with a object m, given that object o is replicated in her neighborhood but object m is not, we can write:

$$w_i^m [\gamma_i - \alpha_i] > w_i^o [\beta_i - \alpha_i] \Rightarrow w_i^m > \frac{w_i^o}{\delta_i} \tag{3.10}$$

2. if player i is better off by replacing object o with m, assuming this time that m is replicated in her neighborhood but o is not, then

$$w_i^m [\beta_i - \alpha_i] > w_i^o [\gamma_i - \alpha_i] \Rightarrow \frac{w_i^m}{\delta_i} > w_i^o \tag{3.11}$$

3. in any other case (o and m both in the neighborhood, or both not), if player i is better off by replacing object o with object m:

$$w_i^m [(\beta_i \vee \gamma_i) - \alpha_i] > w_i^o [(\beta_i \vee \gamma_i) - \alpha_i] \Rightarrow w_i^m > w_i^o \tag{3.12}$$

The last remark I make concerns the way a utility of a player decreases. In a better reply process played on a replication game on a full graph there is only one way a utility of a player i can decrease: another player, say j, starts replicating a object o which was only replicated by player i.

Definition 3.4.1. I will call an object o, *free for player* i if not replicated by anyone in the neighborhood of i ($o \in \bigcup_{j \in \mathcal{N}(i)} s_j$). It will call it *busy for player* i instead ($o \notin \bigcup_{j \in \mathcal{N}(i)} s_j$).

According to the last definition, we can reformulate the condition for a utility to decrease in the following way:

Lemma 3.4.1. *In a better reply dynamic the payoff of a player* i *can decrease only if another player, say* j*, starts replicating a object* $o \in s_i$*, busy for him, but free for player* i*.*

Proof. The cost saving expression is a step (see equation (3.4)), and the lower value of it is reached only when $n_j(a) > 1$. □

Chapter 4

Non-cooperative Games Simulator

In this chapter I describe the main tool I developed during this thesis. I designed and implemented a *simulator of non cooperative games* in C++. So far the simulator can play, following different learning rules, every game $G = \langle N, (A_i), (u_i) \rangle$ where A_i is a non-empty finite set. Non-finite action sets are not yet implemented, though the program structure supports them, and the development of such capability would remain totally transparent for the rest of the application.

The simulator has been written following a strict object-oriented design, and realized to exploit a high level of parallelism. All the objects and functions are realized keeping in mind that one goal of the software is to achieve a high level of efficiency and a low complexity, thus low level functions and efficient algorithms are always used when possible. The data structures provided by C++ have been carefully selected for each different necessity, taking into account the complexity of the operations the application needs to perform on them.

The most important features and design choices will be discussed more deeply in this section. Now follows a high level description of the functionality of the simulation software:

In order to use the tool, the user provides a class describing the game she wants to analyze. This class must inherit from a virtual game class and implements few important functions. Once compiled, the simulator can perform simulations of different learning rules played on the user's game. Every parameter of a learning rule can be customized by the user modifying some configuration files. The user has also at her disposal a smart way to dynamically customize some game-specific parameters, providing an appropriate configuration file. The program's output is fully customizable at different levels of verbosity.

4.1 High level description

The structure of the application is summarized in Figure 4.1, omitting on purpose many additional functionalities. The NCGSIM component (namely the Non Cooperative Game Simulator foremost class) receives an instance of the virtual class representing the user's game. It initializes it using the *game's configuration file* and creates an instance of the learning process the user wish to play, obtaining the type and the parameters from the *learning configuration file*. It finally starts the simulation, performing a number of simulations and using the starting profiles which are both defined in the *multi process configuration file*.

The log files are filled according to the settings the user specifies using both

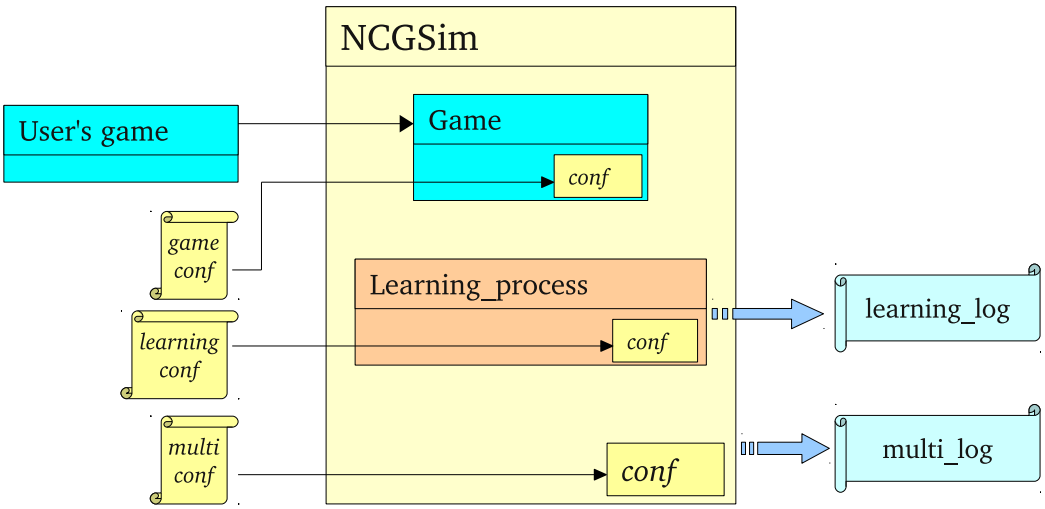


Figure 4.1: Application structure

the *learning* and the *multi process* configuration files.

Every paragraph that follows describes, in more detail, the various capabilities of the application.

4.2 The game description

The explanation of how a game is represented in the program will be particularly detailed since the user of the simulator is asked to understand it perfectly and to provide a description of her game, strictly following the chosen representation.
In the simulator a game is represented using a virtual class (see Figure 4.2). Note that in all the UML diagrams which follow, virtual functions are in *italic* and pure virtual functions are indicated with the "=0" token.

The Game component provides all the functions the application needs in

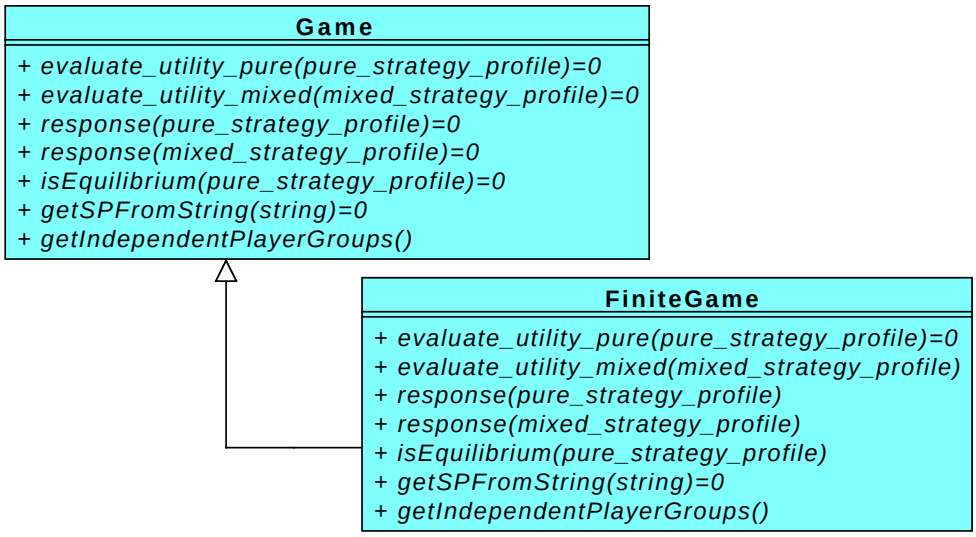


Figure 4.2: Game and FiniteGame UML diagrams

order to play the learning processes. Some of these are purely virtual and strictly depending on the game.
A good example is the function `evaluate_utility_pure()` which is in charge of returning the utility value, given a pure strategy profile. This function constitutes, together with the players set and the actions sets, the complete representation of a game $G = \langle N, (A_i), (u_i) \rangle$ and thus, is the minimum amount of formal information a user needs to provide.

If we assume that the game has finite action sets (the non-finite action set is not implemented yet), then we can use the `evaluate_utility_pure()` function provided by the user in order to implement, following their formal definition, the functions for the evaluation of the utility of a mixed profile, the `response()` functions and the function to check if a profile is a Nash equilibrium.
The simulator provides an implementation of each of these functions, but they are kept virtual since they can be much more efficient if written taking into account each specific game, and not only their formal definition.

Additionally, the user is expected to provide a description of an action of

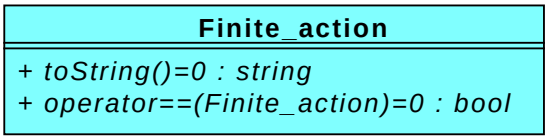


Figure 4.3: FiniteAction UML diagram

the game. Inheriting the right abstract class (see Figure 4.3), it is possible to provide a method, `toString()`, that returns the representation of the action given its value, which better reflects the user needs in terms of notation: the simulator will in fact call the member function `toString()` each time it needs to print an action, providing the user with a consistent representation. This function will be only used to build the output of the program and it does not affect the core algorithms. It does not alter the formal behavior of each simulation.

The user is finally required to provide a function that performs the parsing of a string which represents a strategy profile in user's notation. The expected output is the object corresponding to that string. This can be considered as the inverse function of `toString()`, and it is a good practice, expected from the user, to make the output profile printed by the program compatible with this parsing function.

Apart from detailed requirements we will take later into consideration, the ones listed above are the main requisites asked to the user. Once specified, the simulator can be compiled and is ready to execute with different parameters.

4.3 The learning processes

Once the user defined game class is ready, the simulator can compile. From now on the customization and the tuning of the simulations and of the game itself can be done using three configuration files.
The settings of the learning process must be included in the *learning configuration file* (see Figure 4.1) which allows the user to choose between three different learning rules:

- Simple dynamic
- Fictitious play
- Adaptive play

For a formal description of the three dynamics see Section 2.4. Note that, with "Simple dynamic", I mean a generic *better/best reply dynamic*. The choice between best or better is made by the user when he provides her game class. The function `response()` in fact, does not specify which of the two dynamics is implemented. It simply requires to return a strategy, and the underlying algorithm is up to the user. The simulator helps the user also in this occasion, providing a best response function implementation which only uses the `evaluate_utility_pure()` function.
Independently of the chosen learning rule, the user can specify the maximum number of iterations the simulator will play at each simulation. After this number of plays, the process will stop, even if it does not meet any stopping requirements (see Section 4.5).
Furthermore the user can choose whether to update the players' strategies in a *asynchronous* or *synchronous* manner: keep in mind that every player calculates a response according to the beliefs he has on the play of the other players. In terms of the simulator, these beliefs are encapsulated in the strategy profile the function `response()` receives as input. Briefly, in case of synchronous play, in each iteration, each player plays a response to the beliefs he

had at the end of the last iteration. In case of asynchronous play, on each iteration each player plays a response to his beliefs, and the beliefs are updated before the next player calculates her response. Formally, this distinction was made when talking about SFP and AFP, in Definition 2.4.4 and 2.4.5.
Note that, at the state of the art, while playing asynchronously, the simulator never changes the order in which the players play their action: thay are picked up sequentially, according to their ID. This behavior is restrictive, but if we assume a random picking of players we introduce issues concerning the sopping conditions (see Section 4.5). We will further discuss this matter in Section 6.1.3.
Finally, the user can specify the logging options and the learning rule's specific parameters, like the history length and the sampling size of the Adaptive play.

Design

The simulator design allows the development of further learning rules. What is strictly required from a learning rule is in fact described in an abstract class which can be inherited and implemented (see Figure 4.4). The purely virtual function `start_play()` is actually called by the program to start a simulation from a given pure starting profile.

The NCGSim component performs the following actions in a loop, thus a learning rule must guarantee consistency at every simulation:

1. creates and prepares a pure strategy profile, to use as starting profile
2. calls `start_play()` with the starting profile
3. analyzes the outcome and eventually logs the results
4. calls `reset_process()`
5. destroys the starting profile.

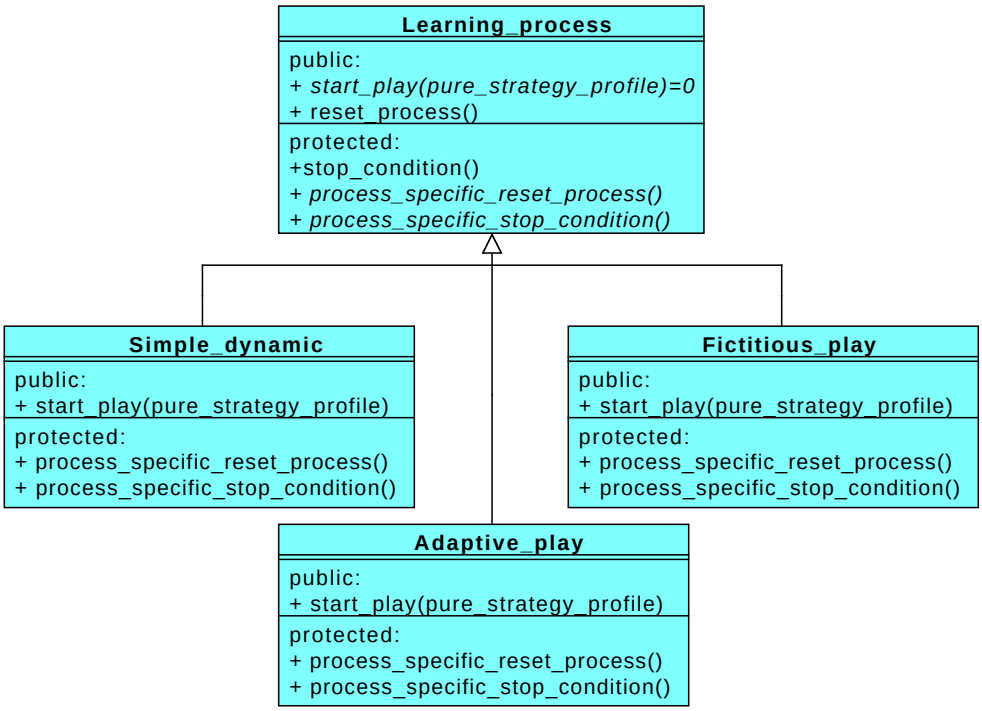


Figure 4.4: Learning_process UML diagram

Not that at the end of each simulation, none of the data structures created by the learning process in the two called functions should be still in memory. Because of its particular structure, it is crucial that the program does not leak memory and an interesting design choice has been made while examining this aspect: the `reset_process()` function destroys the common objects and reinitializes all the variables of the `Learning_process` class and then calls its virtual process-specific counterpart, which is in charge of completing the same tasks in the environment of the inherited class.
The same design choice characterizes the `stop_condition()` function which only stops the process if the maximum number of iterations have been performed in the simulation, but calls its counterpart for the more complex process specific analysis.

4.4 Parallelism exploitation

The simulator has been realized with a special focus on performance. It has been designed to exploit a high level of parallelism performing a multi-threaded execution. Some learning processes have an intrinsic parallel characterization and thus they offer the opportunity to share the workload among different threads. The three learning rules that I implemented in the simulator support different levels of threading execution, depending on the number of available cores and the effective parallelism which can be extracted from each formal definition. For example a synchronous dynamic offers the opportunity to calculate in parallel the responses of the players during each iteration: a player response does not depend in fact on the actions of the others during the same iteration.
An asynchronous process does however not allow the same parallel calculation. Every player in fact needs the response of the previous player in order to decide her play. Though, there is some part of the execution that can be parallelized anyway. In the following lines I will give an account on the level of parallelism reached in every process. Before I briefly explain how the simulator shares the workload among threads.

Sharing the workload

As soon as a simulation is started, the application creates a number of worker threads equal to the number of the available cores. Each worker can execute different jobs specific for every learning process. It waits that the main thread sets the required parameters of a job, and as soon as it gets the start signal, it performs the job and signals back the main thread. This behavior is achieved through the use of condition variables and mutexes.
The workload is shared trying to involve the highest possible number of workers, given a set of independent jobs to perform.
Note that in some games it is possible that even if the dynamic is asynchronous, some players' responses do not depend on the response of other

players. Thus the set of players can be divided in independent groups, which allow some sort of parallelism in the execution. The simulator allows the user to specify this independent grouping of players, re-implementing the virtual function `getIndependentPlayerGroups()`, which in the default implementation, assumes no independence among players. Specifying a "plesiochronous" execution in the *learning configuration* file, the user enables the simulator to use this grouping to achieve more parallelism, fairly assigning to different workers, players which are specified as independent.
As introduced before, each learning rule allowed me to exploit the parallelism in different ways. Here I briefly report what executes in parallel in each case.

- **Simple dynamic**:
  The execution is parallel during the calculation of responses, of course only in case of *synchronous* or *plesiochronous* dynamic. This behavior is common to all the learning rules, hence I will not repeat it in the following cases, even if it is always present.

- **Adaptive play**:
  More parallelism is achieved during the calculation of the players' beliefs. They are computed starting from the samples picked up from the history: the simulator must update the beliefs according to the last picked up sample, which is a pure strategy profile. To do so it must, for every strategy in every sample, retrieve the index of that strategy from the action set (to represent a mixed strategy the simulator uses a ordered tuple of probabilities). Depending on the size of the action space, this action can take a significant amount of resource, but it is independent from everything in the learning process, thus can be parallelized.

- **Fictitious play**:
  In this learning rule, we have one more possibility of parallelization in every iteration and for each player. In case of *synchronous* or *plesiochronous* dynamic in fact, the update of each player's beliefs can be

parallelized as well.

Furthermore, note that the fictitious play needs the pure starting profile expressed as a mixed strategy profile. Also this operation is performed concurrently by the workers threads.

4.5 Stopping conditions

In the design of the simulator I needed to address the problem of when to stop a simulation. Each learning rule in fact requires different stop conditions, depending on his formal properties. In general a process should stop when it converges or when it starts cycling. How to detect these conditions largely varies from rule to rule, and it can be not trivial in some cases. In this section I describe how I addressed and, in some cases, solved the problem on each learning rule. Note that all the algorithms which follows are implemented in the `process_specific_stop_condition()` function in each learning process (see Figure 4.4). They are all realized under the assumption that the `response()` function provided by the user has a deterministic behavior.

Simple dynamic

To solve the problem of convergence and cycles in this case is trivial. The state of this process at time $t > 0$ can be represented with one pure strategy profile s_{t-1}, which represent the last game played, and thus the beliefs the players need in order to decide their new strategy. It is easy to see that a process converges when $s_{t-1} = s_t$.
The cycle detection in the simulator has been implemented by saving a hash of each strategy profile played so far. I have chosen to save a hash and not another representation of the profile, since the latter can vary from game to game and can have an arbitrary size. Each time a new response is calculated at time $t = h$, the simulator checks that the same hash is not present in the history. In case it is present, say at time $t = k < h$, a cycle might have occurred (if not a seldom case of hash collision occurred) and the application checks replaying the process from two possible starting points, choosing the most convenient:

- if $k \leq h - k$: it replays from the starting profile $t = 0$ to the first colliding hash ($t = k$). Once arrived to s_k, it can check if it is actually

the same as s_h, in which case the cycle is confirmed.

- if $k > h-k$: it plays the same dynamic using as starting profile $s_0^{'} = s_h$ for $h-k$ iterations, and checks if $s_{h-k}^{'} = s_h$. In such a case the cycle is confirmed.

Adaptive play

With this learning rule a process converges at time k only if the elements in the history $s_i(k) \in h(k)$ are all the same and equal to s_k (see Theorem 2.4.2). The simulator performs this check in order to decide on the convergence of the process. In order to improve the efficiency of this check, I started the comparison from the first history element, which is the last one which changes in case of convergence.
Cycle control is not implemented in Adaptive play, since in this thesis I do not require an analysis of cycles with this learning rule. However it is possible to check a process for cycles similarly to simple dynamic. Instead of saving the hash of the response at each iteration, we need now to save the hash of the response and of the entire history, thus we can implement the control as in simple dynamic. Note that Adaptive play has an uncertain behavior picking up the samples from history at each iteration, thus it is not possible to simply play the game again and expect the same flow. In order to fix this issue I equipped the simulator with a random number generator that supports multiple random flows and the possibility to set the random seed. Saving the random seed at the beginning of each Adaptive play, makes now possible to implement the cycle checking as in the simple dynamic.

Fictitious play

The convergence condition of a Fictitious play is formally given in Definition 2.4.7. This condition is impossible to check in practice, since we cannot play the game for "every $t \geq t_0$ ". In the simulator I decided to perform the following check, which turned out to be very efficient in terms of perfor-

mance:
If the strategy profiles played in two subsequent iterations are the same, say s_k, it means the beliefs of each player are increasing in the direction of the just played strategy profile and it might be that they are starting to converge. In this case I check whether s_k is a pure strategy Nash equilibrium, using the function `isEquilibrium()`, and in that case I assume the game is converging to it. This assumption is not rigorous, since it might be that after some iterations where the beliefs are approaching the pure strategy NE, one or more players deviate from their old strategy. This heuristic has however practical value, since it allows to discover different equilibria of a game, and it gives hints about how likely is the equilibrium where the process will converge.
The check of cycles has not been implemented for fictitious play. It is possible however, to realize it proceeding as indicated in the Adaptive play section, without worrying this time about uncertain behaviors. Note though that AFP is very unlikely to have cycling beliefs.

4.6 Multiple simulations

In this section I will give an account of the capabilities of the NCGSim component. Its task is to provide the required settings to each learning rule and to launch every simulation. This component can be configured by the user who can choose the number of simulations to be performed. Each simulation starts from a strategy profile and the user can specify which starting policy the simulator must follow. There are three available options:

1. *RANDOM*: The starting profiles are randomly picked up.
2. *SPECIFIED*: The user explicitly provides the starting profiles.
3. *EXHAUSTIVE*: The application enumerates all the possible starting profiles according to the action spaces of the players, and starts the simulations from them.

In order to specify the starting profiles, the user can list them in the configuration file using the notation she prefers. The NCGSim component will call the `getSPFromString(string)` function on each string representing a starting profile, thus the user must write the function consistently with the used notation.
Take into account that, if the user specifies a certain number of simulations, together with the *SPECIFIED* option, the simulator will start picking up random starting profiles in case the user did not provide enough. Note as well that the *EXHAUSTIVE* option (so far implemented only for finite action sets, consistently with the rest of the application) can list a huge quantity of starting profiles, depending on the cardinality of the action sets. This option is useful for small games though, and offers an instrument to completely analyze the learning rule's behavior.

4.7 Dynamic game customization

While designing the simulator I realized the lack of possibility for a user, to customize the parameters of her game. My simulator in fact, did not provide the possibility to easily change the parameters of the game after its compilation. One possible way for the user was to write, inside the game class's constructor, her own parser, which could perform the initialization of the game's parameters from a file. The matter was that the simulator itself did not provide a standard way to customize these parameters, and the entire task was left to the user. In order to implement such a feature, I needed to find a way to initialize game specific variables from a file, without knowing their name or type (The game implementation is in fact left to the user).
The solution I decided to adopt consists in providing the user with a data structure she can use in the game's constructor to initialize some custom parameters. The three basic steps the user must perform are the following:

1. prepare the data structure (namely `Params_description`) describing the content of the game's configuration file
2. initialize the data structure from the configuration file, using the provided function `initialize_from_file(file_path)`
3. use the data structure to initialize the game specific variables

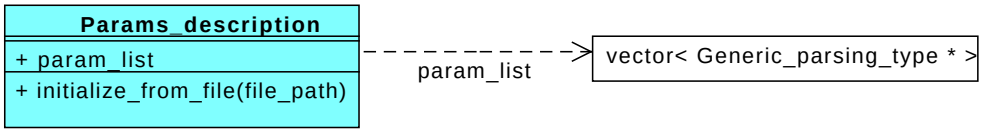


Figure 4.5: Params_description UML diagram

In this way, each time the game's constructor is called, the game configuration file is automatically parsed in order to extract the required parameters. The user has at her disposal a set of different data structures (called parsing

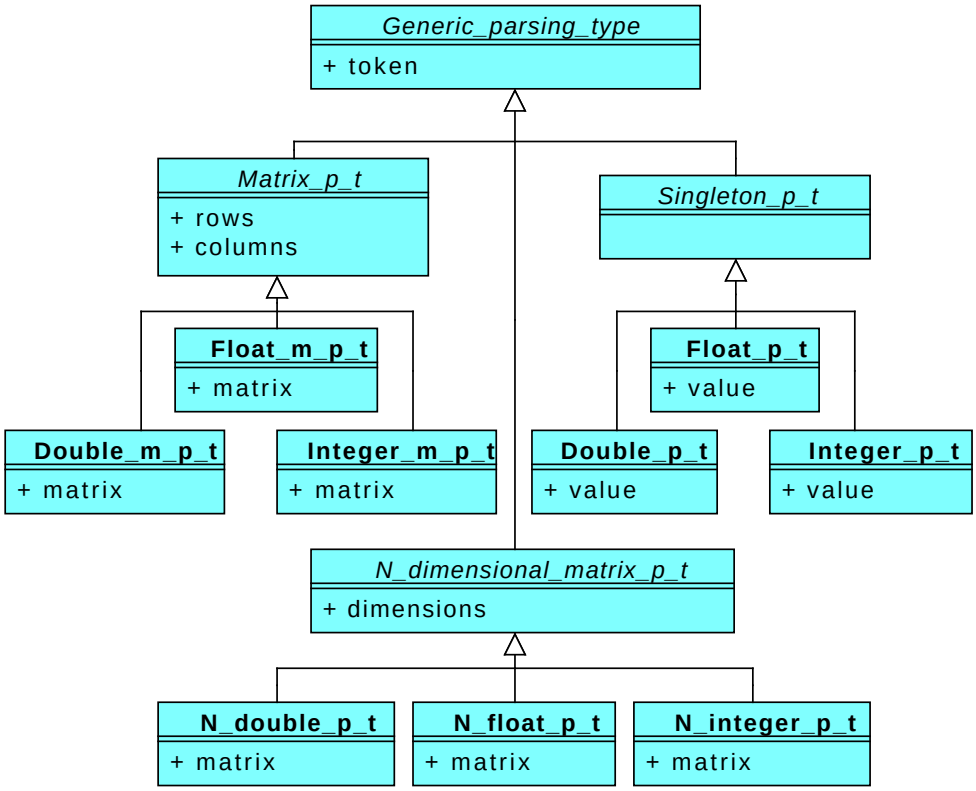


Figure 4.6: Parsing types UML diagram

types) that allow her to describe the content of the file. Her task is to insert in the `Params_description` the appropriate `parsing_type` for each parameter she wishes to initialize. In Figure 4.5 and Figure 4.6 I show respectively the structure of `Params_description` and the different parsing types the user has in hand. All the classes which inherit from `Matrix_parsing_type` allow to initialize two-dimensional matrixes. Even if this can be achieved using the more generic `N_dimensional_matrix_parsing_type`, I decided to provide a more simple way to represent two-dimensional matrixes, since they are very common in games' formulations. The `Singleton_parsing_type` permits to initialize simple parameters like integers, floats or doubles.

Here follows a basic example on how to use these data types:
If, for example, the user needs to initialize a two-dimensional matrix of integers, which in the configuration file is called "values" , she has to insert a `Integer_matrix_parsing_type`, initialized with token="values", in the `Params_description` structure.
The user's listing will be close to the following:

```
Users_game::Users_game(string file_path){
        int n=2,m=3;
        values=new int*[n];
        Params_description description;

        Integer_matrix_parsing_type * values_p=new
            Integer_matrix_parsing_type("values");

        description.param_list.push_back(values_p);
        description.initialize_from_file(file_path);

        for(int i=0;i<n;i++){
            values[i]=new int[m];
            for(int j=0;j<m;j++){
                values[i][j]=values_p->matrix[i][j];
            }
}
```

The configuration file should then look like the following:

```
#game's configuration file
values: 2x3
1 2 3
4 5 6
```

Note that the dimensions of the *values* matrix are expressed immediately after its name, in the configuration file. Note furthermore that it is possible to specify custom dimensions simply parsing them as singleton integers, in such a way to have consistency in the code dynamically assigning the right values to the variables m and n.

There are no limits on the number of the dimensions, the user can use the n-dimensional matrix parsing types to initialize completely generic matrixes.

Chapter 5

Performance Analysis of Replication Games

THE purpose of this chapter is to present the results I obtained using the *non-cooperative game simulator* I just described in order to analyze the *Replication games* class defined in Chapter 3. I mainly performed sensitivity analysis with the ambition to understand to what extent different game parameters affect the performance of different learning rules and how.

I focused on *Simple dynamic* and *Fictitious play* described in Section 2.4, and I performed the simulations using a computer with 8 cores, which allows to test the concurrent execution of the simulator and to evaluate the effective level of parallelism reachable with a Replication game $C = \langle N, (\mathcal{R}_i), (u_i) \rangle$.

I will start giving an account of the Replication game class in the simulator and I will go on showing the most relevant results coming form different groups of simulations.

5.1 Replication game in the simulator

At first I had to provide the simulator with a `Game` class describing my game, following the rules I exhaustively expressed in Section 4.2. I inherited from the `Finite_action` class which allowed me to describe an action of a player as the choice of the content of her cache. Afterwards I implemented the compulsory functions of the `Finite_game` class. Note that in order to express the finite action set of a player i, the game needs to compute all the possible cache configurations given the cache size (K_i) and the number of objects $m = |O|$. The resulting action set is the set of all the possible K_i-*combinations* of the m-element set O, and thus its size is:

$$\binom{m}{K_i} = \frac{m!}{K_i!(m - K_i)!} \tag{5.1}$$

This calculation is made in the game constructor, as soon as the simulator knows the cache sizes of the players and the cardinality of the objects set. The complexity of this calculation, together with the required memory space, grows linearly with the number of objects. Note however that the action set's size in (5.1) has its maximum when the cache size K_i is equal to the half of m and grows exponentially with the cache size K_i.

The calculation of the utility function given a pure strategy profile is made according to (3.3). More than the compulsory functions, I have re-implemented the `Response(pure_strategy_profile)` function in order to increment its performance: instead of computing the utilities of all the actions of the player and pick up the best, this function first orders the objects according to their actual contribution to the player (see equation (3.4)). It then picks up the first K_i objects of the resulting list and uses them to populate the cache.

The computation of the independent players groups is made according to the graph underlying the game: if two players are not connected then they can be in the same independent set since their best reply is not reciprocally affected (see Section 3.2). This computation is made through an implementation of the *Welsh-Powell* greedy heuristic.

The game parameters can be dynamically customized using the game config-

uration file. I achieved this behavior using the simulator's feature described in Section 4.7. A very simple configuration file for this game looks like the following:

```
#Replication game configuration file
Num_players:3
Num_objects:5
Cache_sizes: 1x3
  3 3 3
Alpha: 1x3
  1 1 1
Beta: 1x3
  2 2 2
Gamma: 1x3
  3 3 3
Weights: 3x5
  4 1 3 7 15 3 7 15 31 63 1 3 7 2 5
Graph: 3x3
  1 1 0
  1 1 1
  0 1 1
```

5.2 Sensitivity analysis

I performed different groups of simulations, each one with the goal to discover if the learning rule played on the game was affected by the change of one of the following parameters:

- cache sizes (K_i)
- Erdős-Rényi graph edge probability (p)
- weights of the objects (w_i^o)

Every simulation which follows has been performed using α, β and γ equal for every player, and respectively set to $\alpha = 1$, $\beta = 2$ and $\gamma = 3$. Note that using these parameters we always fulfill the condition of non-increasing payoff function defined in equation (3.6). Note furthermore that $\delta_i = 2$ for every player of the game.
In order to generate the graph underlying the game I used a $G(n, p)$ Erdős-Rényi model [7], where each edge is included in the graph with probability p, with the presence or absence of any two distinct edges in the graph being independent. In the second group of simulations I analyzed how different probabilities p affect the number of iterations necessary for each learning rule to converge.
The choice of the weights is always random, but once is made, the weights' set is kept equal for all the simulations performed on a game. This is done in order to avoid that they alter a sensitivity analysis performed on different parameters. Apart from the sensitivity analysis especially made to understand the effect of different weights' sets (Section 5.2.3), every game I analyze keeps always the same set of weights.
I performed every investigation on two learning rules: *best-reply* and *fictitious play*. Both of them have been always played following an asynchronous dynamic that exploits the internal parallelism of the *Replication game*: I used the plesiochronous feature of the simulator and the graph-coloring heuristic

described in Section 5.1 to achieve the high level of concurrency already discussed in Section 4.4.

5.2.1 Sensitivity to cache sizes

The aim of this section is to understand how much the convergence performance of the best-reply dynamic and fictitious play on a Replication game is affected by the cache capacity.
I tuned the simulator to log the number of iterations of different plays of the same game but with different cache sizes. Note that even if the cache capacity is different among games, I keep it equal among the players playing the same game ($K_i = K \ \forall i \in N$).

Best-reply dynamic

I performed the analysis of the best-reply dynamic playing the following games and varying their cache sizes:

1. $|N| = 5$, number of objects $m = 25$, and Erdős-Rényi $p = 0.5$
2. $|N| = 10$, number of objects $m = 20$, and Erdős-Rényi $p = \{0.3, 0.5\}$
3. $|N| = 20$, number of objects $m = 30$, and Erdős-Rényi $p = \{0.3, 0.5\}$
4. $|N| = 40$, number of objects $m = 60$, and Erdős-Rényi $p = \{0.3, 0.5\}$

Note that, as I specified in the previous paragraph, the size of the action set of every player grows exponentially with the cache size. Unfortunately we have a limit on the choice of the parameters since, in case of best-reply dynamic, memory is the bottleneck of the simulator's performance. Consider in fact the Game 4 with 40 players: I call the memory size necessary to store the representation of an object in cache as l. Note then that the size of a player i's action is lK_i, where K_i is the cache capacity. I played this game with 3 different cache sizes, namely 2, 3 and 4.

cache sizes	**2**	**3**	**4**	**5**
action	$2l$	$3l$	$4l$	$5l$
action set	$1770 \cdot 2l$	$34220 \cdot 3l$	$487635 \cdot 4l$	$5461512 \cdot 5l$
total size	$1.41 \cdot 10^5 \cdot l$	$4.11 \cdot 10^6 \cdot l$	$7.80 \cdot 10^7 \cdot l$	$1.09 \cdot 10^9 \cdot l$

Table 5.1: Memory requirements for action sets in Game 4

Table 5.1 shows the memory requirements as multiples of l. I could not increase the cache size over 4 because of memory shortage. It can be possible however to represent the action set of a player in a different way, without enumerating all the possible cache configurations and thus saving memory. Unfortunately this approach does not fit with the simulator architecture, which, being not game specific, does not allow to change the action set representation. I will further discuss the possibility to add this feature in Section 6.1.1.

The results of the simulations showed that, for the best-reply dynamic, the number of iterations needed to reach a Nash Equilibrium increases in general with the cache size. Note this trend in Figure 5.1, which represents the averages of the number of iterations over 80 simulations using Game 1.

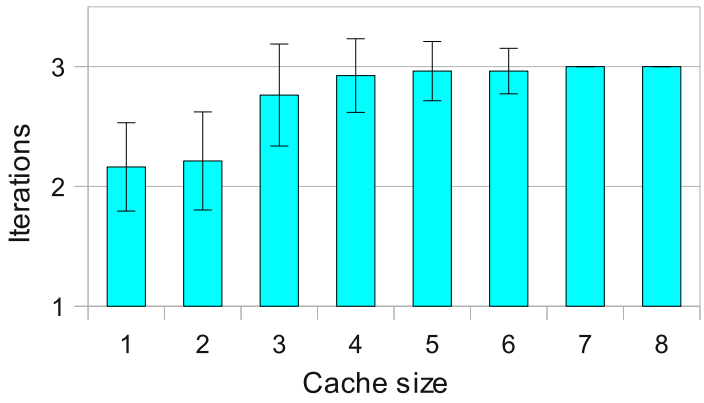


Figure 5.1: Game 1

Note furthermore that the number of iterations levels out with higher sizes of cache. I experienced the same behavior with Game 2 and edge probability $p = 0.3$: the results are shown in Figure 5.2. The standard deviation of each set of simulations is also shown in black in each figure, on the top of each bar.

It was unfortunately impossible for me to check if the same trend holds in games with a higher number of players, since I could not increase the parameters enough to eventually observe the same curve, because of memory shortage.

I observed however another behavior in the case of Game 2 and $p = 0.5$: increasing the probability p the number of iterations decreases and in some cases falls for higher cache sizes. Note Figures 5.3 and 5.5: The amount of iterations has, at first, the same increasing behavior we already saw, but then it decreases for higher caches capacity.

Going further with the analysis of the game I realized that this behavior is

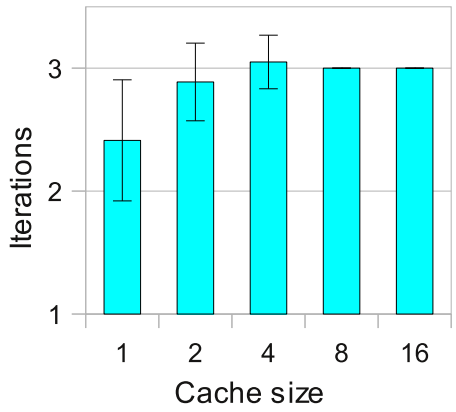


Figure 5.2: Game 2, $p = 0.3$

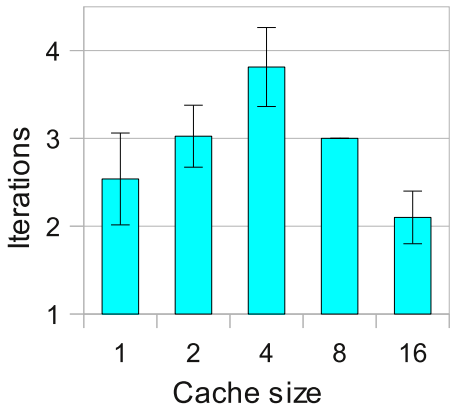


Figure 5.3: Game 2, $p = 0.5$

due to a shortage of objects. When the graph starts to be more connected, the number of objects which are not replicated in the neighborhood is smaller. This happens simply because every player i has more neighbors and thus more objects $o \in \bigcup_{j \in \mathcal{N}(i)} s_j$ (see notation in Section 3).

If the number of objects is too small, it happens that all the most valuable

objects of a player are already replicated by one or more other players. If a player starts to replicate one of them, the possibility to affect the response of a neighbor is small and thus is very likely that a best response remains such. The analysis of Game 4 showed the same result I already commented, thus

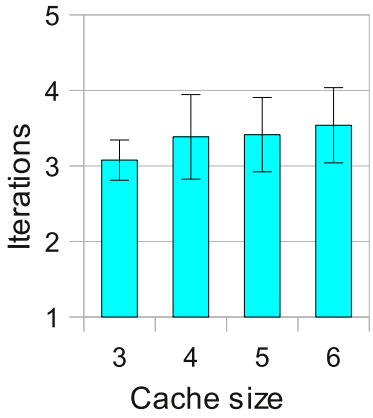


Figure 5.4: Game 3, p=0.3

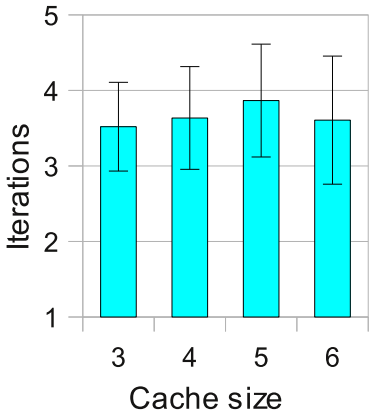


Figure 5.5: Game 3, $p = 0.5$

is omitted in this section.

Fictitious play

I performed the analysis of the fictitious play process playing the following games and varying their cache sizes:

1. $|N| = 4$, number of objects $m = 16$, and Erdős-Rényi $p = 0.5$
2. $|N| = 5$, number of objects $m = 16$, and Erdős-Rényi $p = 0.5$
3. $|N| = 6$, number of objects $m = 18$, and Erdős-Rényi $p = 0.3$
4. $|N| = 6$, number of objects $m = 9$, and Erdős-Rényi $p = 0.5$

Comparing the parameters of this set of games with the parameters used while simulating the best-reply dynamic, one may realize the low number of players and objects used in this case. This is a consequence of the complexity of the fictitious play algorithm. More precisely if the bottleneck of

the best-reply was memory, now we have much higher restrictions because of computational complexity.

In every iteration of fictitious play the simulator needs to compute a best response to a mixed strategy profile, for every player of the game. Formally we expressed the equation of the *expected mixed utility* in 2.2. We need to analyze the complexity of such a calculation thus we report again the equation of the expected payoff to a mixed strategy:

$$U_i(\alpha) = \sum_{a \in A} p(a) u_i(a), \text{ where } p(a) = \prod_{j \in N} \alpha_j(a_j)$$

Since we are calculating the pure best response of player i, we need to compute the value of U_i for all the strategy profiles (a_i, α_{-i}) such that $a_i \in A_i$. Then we can pick up the value of a_i correspondent to the maximum U_i.

Note that to evaluate every single $U_i(\alpha)$ we need to sum $|A|$ elements. According to equation 5.1, the size of set A will be:

$$|A| = \prod_{i \in N} \binom{m}{K_i} = \prod_{i \in N} \left(\frac{m!}{K_i!(m - K_i)!} \right) \tag{5.2}$$

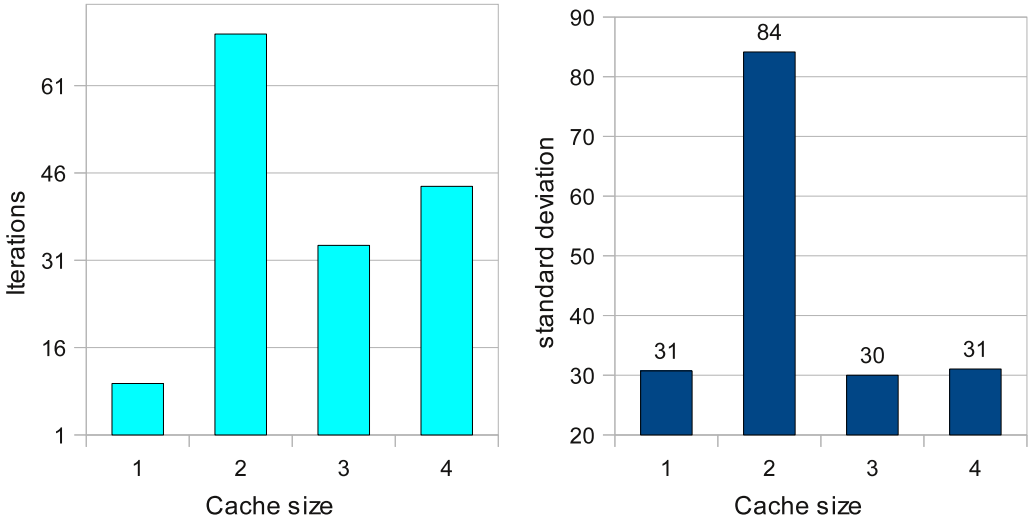
Taking into account the assumption we made at the beginning of this chapter ($K_i = K \ \forall i \in N$), we can rewrite 5.2 as:

$$|A| = \prod_{i \in N} \binom{m}{K} = \binom{m}{K}^n = \left(\frac{m!}{K!(m - K)!} \right)^n \tag{5.3}$$

This number would be completely unfeasible even for very small *Replication games*. Fortunately we can actually remove all the cases where $p(a) = 0$. Since we are calculating a pure response for player i, $\alpha_i(a_i)$ will be 1 in correspondence to the action we are evaluating, and 0 for all the other actions. Hence we are actually calculating $\binom{m}{K}^{n-1}$ utilities.

There is more: in fictitious play there will be many actions which will never be played, thus their belief (in this notation $\alpha_j(.)$) will always be 0. The simulator has been optimized to exclude them from the calculation. This optimization made possible to run fictitious play on the simulator, but unfortunately the constraints on the game's parameters are still very strict.

Unfortunately the trends of the simulations performed during the cache sensitivity analysis do not seem to follow a regular pattern. The number of iterations does not in general increase with the cache size, and probably depends on more complex aspects. What comes out from every simulation however, is that a small variation of the cache size noticeably affects the convergence performance. Keeping the other parameters constant, we often noticed a high gap in the number of iterations. Note for example Figure 5.6a and 5.6b: we represented the number of iterations and their standard deviation needed to reach the end of 50 fictitious play simulations using Game 2. The number of iterations does not increase with the cache size, and we have



(a) Game 2, iterations

(b) Game2, standard deviation

Figure 5.6: Average of iterations and relative standard deviation for Game 2

a peak when the cache has size 2.

Notice furthermore the huge values of the standard deviation. For fictitious play in fact, the iterations before the convergence highly depend on the starting profile. It may happen that the dynamic reaches immediately an equilibrium in the first iterations; if not fictitious play needs a lot more iterations to switch from a best response to another one. This is mainly be-

cause the process has an infinite history, thus the more it goes on, the higher becomes the number of plays needed to significantly affect the beliefs.

A trend we distinguished, while analyzing the outcomes of the simulations, concerns this "inertial" behavior. While observing the results of the fictitious play process, we noticed that all the values could be grouped into a few small intervals. To better understand the pattern, the discrete probability distribution of the number of iterations has been calculated and drawn in Figure 5.7. The figures are related to Game 1 and 4.
Observe how the outcomes of the experiments which do not converge before the 10^{th} iteration, are all grouped in small intervals. Note furthermore that these intervals are all multiples of the same quantity: for Game 1 this value is 16 (we have samples only around 16 and 32), and for Game 4 is 40 (samples only around 40, 80 and 120).
This quantity is related to the equation of the mixed utility 2.2, and affects the number of plays required to have a sufficient change on the beliefs in order to switch action. Observe finally how this quantity does not apparently depend on the cache size: both the distributions in Figures 5.7b and 5.7a are related to Game 1, and even if the cache size is different, the samples group around the same values.

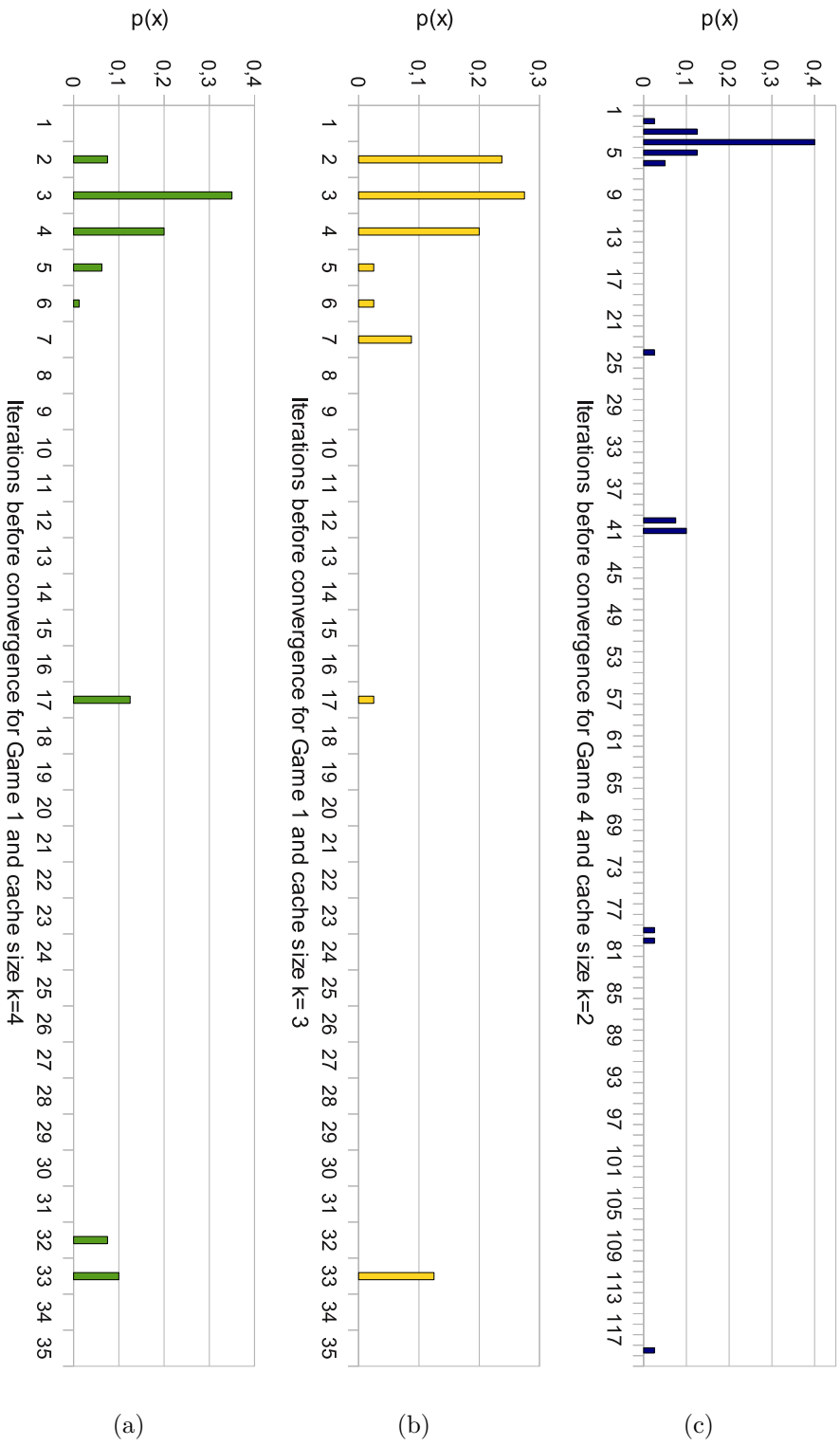


Figure 5.7: Probability distributions calculated on the outcomes

5.2.2 Sensitivity to Erdős-Rényi edge probability

In this section I report the results of a sensitivity analysis performed in order to understand how the Erdős-Rényi edge probability of the underlying graph affects the number of iterations a learning rule needs to converge.
The same analysis has been performed to test both the simple dynamic and the fictitious play. Concerning the choice of the games' parameters, I had to face the problem of memory shortage in case of simple dynamic and computation time in case of fictitious play. I already discussed this issue in the previous section. I have tried however to perform consistent simulations despite of these limits on the adoptable values.
In every simulation I performed for the analysis in this section, I used games where the cache capacity is the same among all the players and always equal to 3. It represents a trade off between performance (memory requirements and computational time) and complexity of the scenarios. Using a smaller cache would have resulted in too simple games, where, during a learning process, every player would have picked up very few items from the total set of available objects.
Every game I analyzed has been played repetitively with 5 different graph topologies. For each game I generated 5 different graphs following the $G(n, p)$ Erdős-Rényi model [7], specifying the probabilities p =0.1, 0.3, 0.5, 0.8 and 1. Obviously the case where $p = 1$ generated fully connected graphs where the set of the neighbors of each player i is the set of all the other players: $\mathcal{N}(i) = N \setminus \{i\}$.
I will now describe the results distinguishing between the two learning rules I analyzed.

Best-reply dynamic

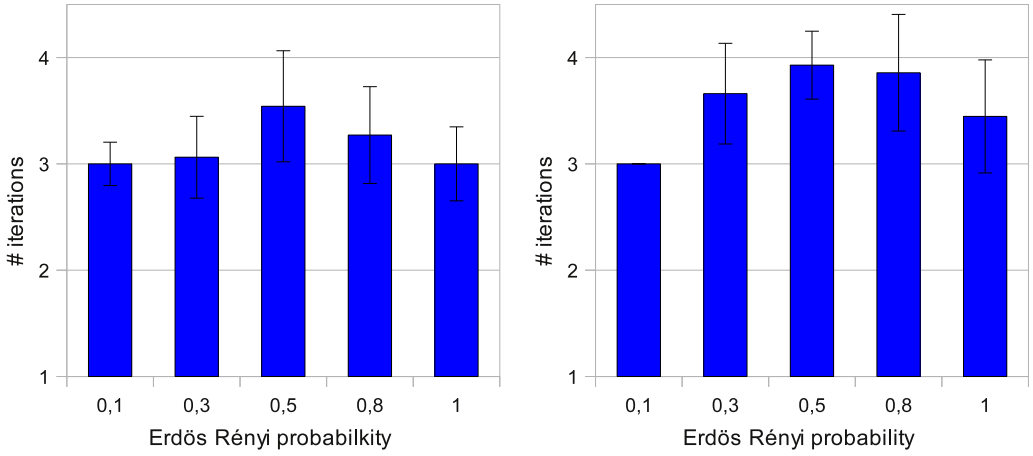
In Table 5.2 I list the games I used to analyze the best-reply dynamic and the number of simulations I completed before collecting the results.
In order to fairly compare the outcomes of the simulations played on different games, I had to find an approach to choose the number of objects in such

Game	$\|N\|$	m	$K_i = K$	**# simulations**
1	5	8	3	56
2	10	15	3	72
3	20	30	3	48
4	40	60	3	56
5	80	120	3	56

Table 5.2: Games used in the simulations and parameters

a way that the behavior of the simulations would not have been affected by a too different objects' availability. Roughly speaking, I wanted to guarantee the players with the same supply of objects in order to assure that their choices would have been made under the same conditions. Hence I picked up the number of objects according to the following rule: $m = \lceil |N| \cdot K \cdot 0.5 \rceil$. Note that a player must pick up $K = 3$ objects to fill up her cache. If the game is played on a fully connected graph, we would need $|N| \cdot K$ different objects to give the possibility to each player to pick up a free object. I multiplied the resulting value by 0.5 in order to prevent too many or too few available objects: we have already seen in the previous paragraph that with a too few number of objects the amount of iterations needed to converge decreases. On the other hand, with too many objects we encounter the opposite situation: the possibility that an object, valuable for a player, has already been replicated is low, and it gets even lower if the player has few neighbors (in our case if the Erdős Rényi edge probability is little).

I observed this phenomenon in all the simulations I performed. The conclusion is that the number of objects in a Replication game is a decisive factor which notably affects the amount of iterations required to the convergence.
I will report now some figures that depict the outcome of the analysis about the variation of the Erdős-Rényi edge probability. The results of the simulations on Game 3 and Game 4 are represented in Figures 5.8a and 5.8b.
In all of the simulations we note a simple regularity on the trends. Every



(a) Game 3, $(20, 30)$ (b) Game 4, $(40, 60)$

Figure 5.8: Average of iterations for Game 3 and Game 4

chart in fact has his maximum around its center, in correspondence to Erdős-Rényi edge probability $p = 0.5$ or $p = 0.8$ in case of 5 players (see Figure 5.9).

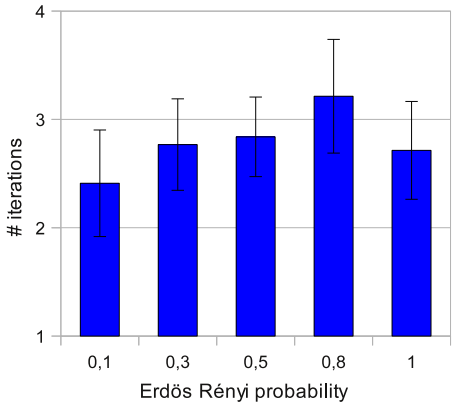
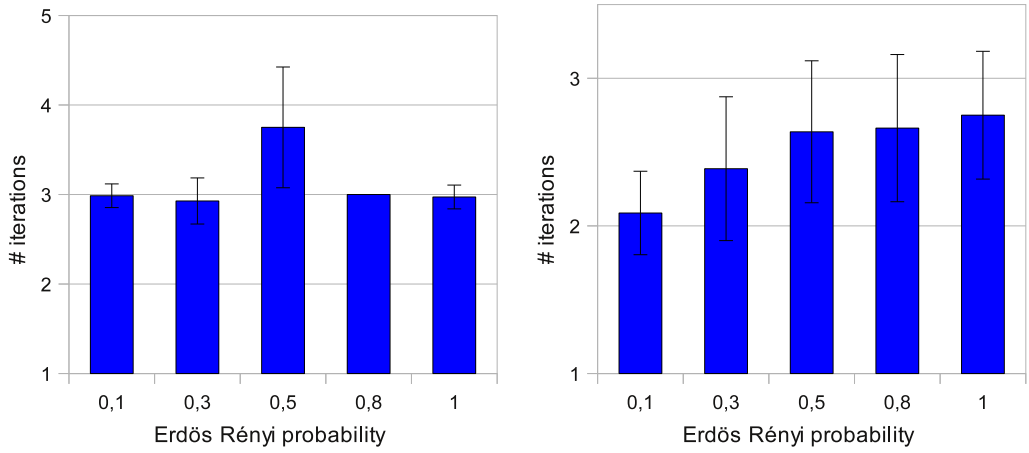


Figure 5.9: Game 1, $(5, 8)$

Very often the minimum corresponds to the outermost Erdős-Rényi probabilities, namely $p = 0.1$ and $p = 1$. This reflects the considerations I just made about the number of objects.
When the behavior is not exactly the expected one, we can probably find the reason in the choice of the graph edges: since the number of the players is not high, we cannot expect the effective probability between the edges to be exactly the one specified. We probably need to run the simulations on a bigger set of players and objects to ensure an exact peak at $p = 0.5$.

However, in order to show how the Erdős-Rényi edge probability affects

the number of iterations without the limit given by the number of objects, I decided to perform a simulation modifying Game 2 by significantly raising the number of objects from 15 to 120. I could use such a high amount of objects because the number of players is still feasible for the available memory. The results of the normal formulation of Game 2 together with the modified one, are shown in Figures 5.10a and 5.10b.



(a) Game 2,$(10, 15)$ (b) Game 2, $(10, 120)$

Figure 5.10: Analysis of Game 2, different amount of objects

We can observe the expected maximum in $p = 0.5$ in the $(10, 15)$ case, but what we need to focus on is the $(10, 120)$ case: note how the amount of iterations continues to increase.
This behavior was expected, since we removed the limit of the available objects. In such an environment, raising the Erdős-Rényi edge probability leads to a higher number of possible conflicts among players, and thus to a higher number of iterations before the convergence.

Before proceeding with the fictitious play analysis, I used all the experiments in this section to briefly analyze the trend of the iterations depending on the number of the players.

Figure 5.11 has been made calculating the average of the number of iterations of each game, including the experiments with every Erdős-Rényi edge probability. We can easily see that the number of iterations rises with the number of players, and thus with the game complexity and possible conflicts.

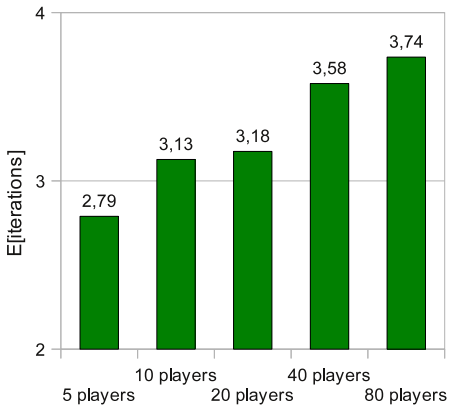


Figure 5.11: $|N|$ sensitivity analysis

Fictitious play

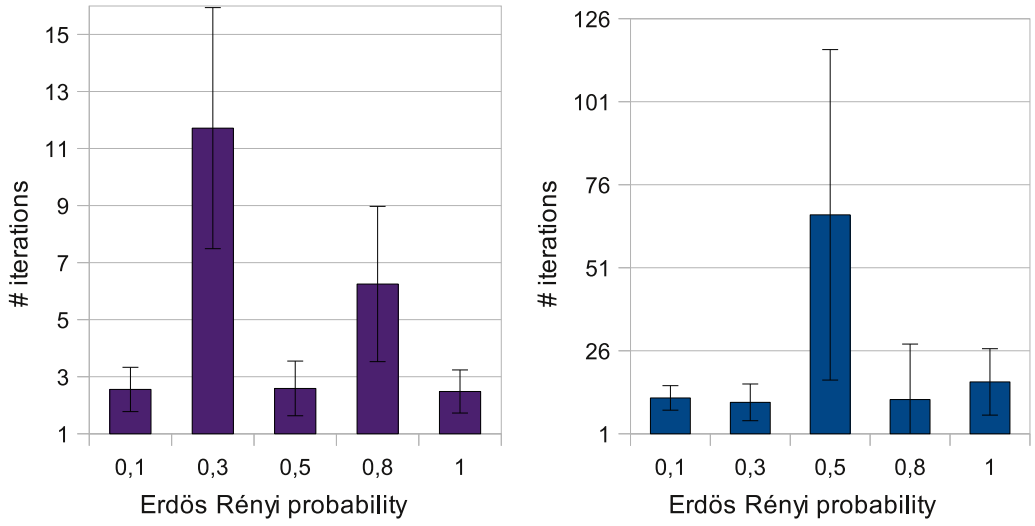
I performed a preliminary analysis of fictitious play using the following set of games, all with $K = 3$:

1. $|N| = 4$ players, number of objects $m = 6$
2. $|N| = 5$ players, number of objects $m = 8$
3. $|N| = 6$ players, number of objects $m = 9$
4. $|N| = 7$ players, number of objects $m = 11$
5. $|N| = 8$ players, number of objects $m = 12$

Unfortunately I could not use a higher number of players because of the computation constraints. The number of objects has been chosen using the already explained rule: $m = \lceil |N| \cdot K \cdot 0.5 \rceil$.
The first experiments were performed as usual by randomly choosing the weights for each game, and keeping them equal for all the simulations performed on that game. The results of this phase do not apparently show any regularity apart from the grouping of the samples around intervals which are multiple of the same quantity, as we pointed out in the previous section. The outcomes of the simulations on Game 1 and 5 are reported in Figures 5.12a

and 5.12b.

With a deeper examination of the results, comparing them with the topology

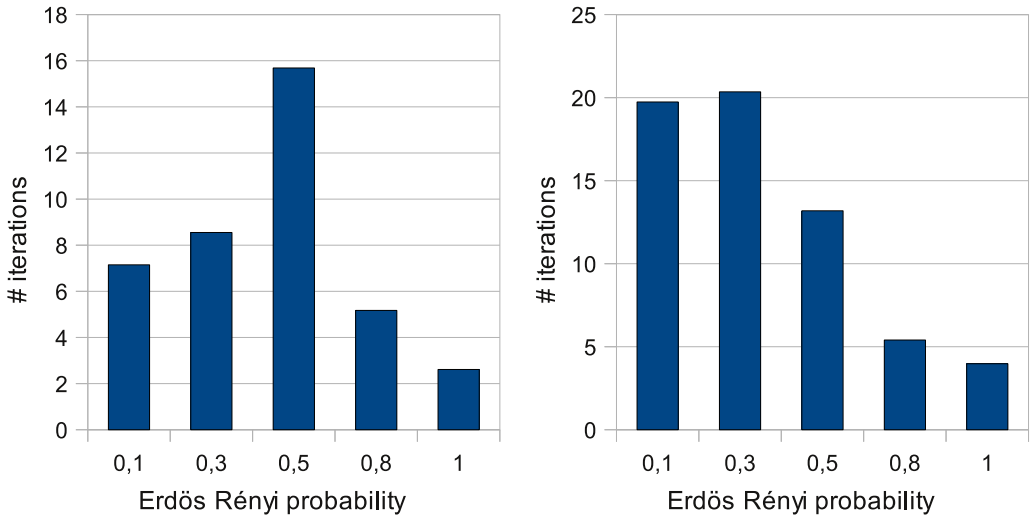


(a) Game 1 (b) Game 5

Figure 5.12: Analysis of Game 1 and 5

of each underlying Erdős-Rényi graph, one can figure out a great influence of the graph topology together with the objects' weights. Unfortunately, since the players set is small, an edge can make the difference: there may be only few players with similar objects' weights, who would actually compete for the same objects (thus increasing the number of iterations to converge), and whether they are connected or not really makes the difference.

For this reason we performed the same simulations using the same objects' weights for all the players. Note that in this case, according to the results in Section 3.3, we are simulating a *Congestion game*. The outcomes of the learning processes finally showed a common pattern: as depicted in Figure 5.13a and 5.13b , all the graphs show an increasing trend until their only peak, after which the number of iterations goes on decreasing.



(a) Game 2

(b) Game 4

Figure 5.13: Analysis of Game 2 and 4

5.2.3 Sensitivity to objects' weights

The last group of simulations aims to determine if the convergence properties of a Replication Game are affected by the choice of the object's weights. The results are totally different between *simple dynamic* and *fictitious play*. In order to analyze the variations of the number of iterations playing a best reply, I picked 10 different sets of weights and performed 40 simulations for each set, starting each time from a random starting profile. I carried out the same analysis on two different Replication games with the following parameters:

1. $|N| = 20$, number of objects $m = 30$, cache size $K_i = 3\ \forall i \in N$, and Erdős-Rényi $p = 0.5$

2. $|N| = 40$, number of objects $m = 60$, cache size $K_i = 3\ \forall i \in N$, and Erdős-Rényi $p = 0.5$

The results show that the best reply dynamic is not really influenced by the choice of weights: in both the cases I calculated the 10 averages over the 40

plays, and subsequently I computed mean and variance over the 10 averages.

	Best-reply		**Fictitious play**	
Weight set	$\|N\| = 20$	$\|N\| = 40$	$\|N\| = 6$	
1	4.35	4.6	12.18	18/40
2	4.48	4.4	16.20	22/40
3	4.40	4.625	8.93	9/40
4	4.45	4.45	8.38	11/40
5	4.25	4.7	5.05	2/40
6	4.63	4.55	8.48	13/40
7	4.35	4.45	10.55	4/40
8	4.60	4.55	7.43	3/40
9	3.85	4.775	9.73	6/40
10	4.25	4.6	5.30	0/40
$\sum$	4.36	4.57	9.62	
σ^2	0.04	0.01	11.52	

Table 5.3: Results of weights sensitivity analysis

The results are listed in Table 5.3: the two replication games, played following a best reply dynamic, converges on average in 4.36 and 5.57 iterations. The small variance, in both cases, shows that the number of iterations before the convergence is not influenced on the choice of weights.
I experimented a different behavior while performing the same sensitivity analysis with fictitious play. The game I used to perform the tests is the following:

- $|N| = 6$, number of objects $m = 9$, cache size $K_i = 3\ \forall i \in N$, and Erdős-Rényi $p = 0.5$

In Table 5.3 I show the average of the number of iterations and the number of plays that exceeded 10 iterations to converge, for fictitious play. I decided

to add this measure because I noticed that the mean and the variance are not enough to understand the behavior of fictitious play. I noticed some regularities in the behavior of fictitious play: usually if the game does not converge during the first 10 iterations, then it needs many more. This happens because a player, in order to change his best response, needs a significant change on the beliefs of the other players. Since fictitious play has infinite memory, the more iterations have been played, the more iterations are needed to affect the beliefs.

The results clearly show that this learning rule is much more affected than the best-reply dynamic by the weights choice. The games with weight set 5 and 10 for example, converge almost always (39 cases over 40) in less than 10 iterations. The behavior of the games with weight sets 1 and 2 is completely different, they often (22 cases over 40) need more than 10 iterations to converge.

Chapter 6

Conclusions

This dissertation has focused first on the theoretic analysis performed on the class of *Replication games* and presented the related results. We proved the equivalence of some particular subsets of the main class to the *Player-Specific Congestion games* and the *Matroid Congestion games* classes. We provided furthermore some formal definitions that may help with the comprehension of the main game's structures and a possible future investigation of their properties.
The equivalence to the two classes of games already studied in literature allowed us to perform a study on their properties and decide if the previous results on these classes also hold for *Replication games*. In this way we showed that, under particular assumptions, this class of games possesses the FIP or is *weakly acyclic*.
The formal analysis suggested also the absence of cycles in some other subclasses of the *Replication games* class, but this has not been proved yet.

Afterwards we described the design choices that brought to the realization of the general purpose *simulator for non-cooperative games*: a particular focus on performance and parallelism allowed the implementation of a particularly efficient application, capable of exploiting concurrent calculations intrinsic in games.

Its modular structure allows future users or developers to implement custom checks on the learning process, permitting to verify generic properties coming from theoretic results.

Finally we used this simulator to perform a sensitivity analysis on the parameters of the *Replication game*, and to describe how they affect the performance of two common learning rules: the *best-reply* dynamic and the *fictitious play*.
We showed that, even though the way some parameters affect the first learning rule is quite straightforward, the analysis of their effects on fictitious play is considerably tricky.
The main results achieved in this phase can be summarized as follows:

- Increasing the cache size does not in general reflects to a higher number of iterations before convergence. Depending on the number of available objects and on the underlying topology, it may even decrease the convergence time.

- How the weights are chosen does not significantly affects the performance of the best-reply dynamic, though considerably influences fictitious play.

- The number of edges of the underlying graph, and the way the players are linked is a primarily factor affecting both the learning rules. Higher edge probability does not mean in general higher number of iterations. An influencing factor for fictitious play, more than the edge probability is how the players are linked: if there are a few players competing for the same set of objects, whether they are connected or not really makes the difference.

- The number of iterations of fictitious play follows a regular pattern: the learning processes tend to converge in a number of iterations that is very often a multiple of a game specific quantity. This value, which showed to be not dependent on the cache size, depends on the amount

of plays required in order to have a significant change on the beliefs which allows to switch action.

6.1 Future development

In this section we describe the main limitations of the simulator and the missing functionalities that would have made easier our analysis. While performing the simulations in fact, we drew out a list of the desirable additional features, mainly concerning the performance of the application.

6.1.1 Implicit representation of action sets

One aspect that immediately comes in mind when looking at the constraints to meet when tuning the parameters of a simulation, is the memory requirement while playing a best-reply dynamic. The real bottleneck of such a simulation is in fact the dimension of the action set of each player. We already reported at the beginning of Chapter 5 that their size explosion is due to the generic representation in the simulator. We needed in fact to enumerate all the possible cache configurations, and to store them in the action set structure.
It would really improve the simulator's performance, as well as the range of possible runnable scenarios, to have a different representation to store the set of the possible choices a player can chose among, without enumerating them. In case of *Replication games* or *Congestion games* it would be enough to simply store the set of objects a player can chose, together with a constraint on size, or other general game related constraints. Such an implicit representation of a set would have size proportional to the objects set's cardinality. It follows immediately that the representation of a single action can remain unchanged. What will change is the `response()` function that, at the state of the art uses the current representation of action set.
Such a memory saving technique would allow to simulate scenarios that are really far from the current possibilities of the simulator.

6.1.2 Infinite action sets

The simulator still lacks the support for non-finite action sets. The main problems related to this feature concern the way to calculate a best response or how to represent a mixed strategy. Without knowing the expression of the utility function, the program would need to use numerical methods to calculate its maximum and thus find the best response (see for example *Cournot's oligopoly* in Section 2.3.1).
Furthermore a mixed strategy, in the case of non-finite action sets, would be a probability distribution, and thus the application must be able to calculate a best response given that. Of course these tasks can be left to the user, since, according to the `Game` abstract class, these all are pure virtual functions. It still remains however the problem of how to represent a probability distribution in the best way to ease the user's tasks and calculations. Certainly the implementation of a customized game class will not be as straightforward as it is now.
Even more challenging will be the derivation of the beliefs in Fictitious play: the program would in fact need to deduce a probability distribution from a finite set of values (the history of game plays), and this can be achieved for instance using statistics. To perform a maximum likelihood estimation for example, we would need to assume a certain probability distribution and then estimate its parameters, such as mean or standard deviation.

6.1.3 Non-deterministic response and learning capabilities

As we already pointed out in Sections 4.3 and 4.5, the simulator always has a deterministic behavior at the state of the art. Introduction of indeterminism can affect two parts of the application:

1. The `response()` function can provide different responses to the same action profile.

2. The order in which the players play asynchronously may vary.

Both the scenarios would break the current stopping conditions: as we described in Section 4.5, the entire learning process can be replayed to confirm a cycle detection. In order to guarantee the same behavior in the first scenario though, we need to assure that the `response()` function returns the same responses provided during the first play. The consequences are analogous for the second scenario: in this case we need to guarantee that the simulator would make the same choices when picking the players.
Both the issues can be solved by reading and saving the simulator seed at the beginning of each learning process. We already provided, included in the simulator, an implementation of the random number generator written by L'Ecuyer [10], which allows multiple random streams.
Note though that dealing with both the scenarios at the same time may be tricky, since we need to use two independent streams, one used by the user in the `response()` and another one for the learning process. Note furthermore that we cannot guarantee that the user generates only one random number per function (`response()`) call and this may introduce problems in the implementation.

Bibliography

[1] H. Ackermann, H. Röglin, and B. Vöcking. Pure Nash equilibria in player-specific and weighted congestion games. *Internet and Network Economics*, pages 50–61, 2006.

[2] U. Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135(1):572–578, 2007.

[3] G.W. Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13:374–376, 1951.

[4] A.A. Cournot. *Recherches sur les principes mathématiques de la théorie des richesses/par Augustin Cournot*. L. Hachette, 1838.

[5] G. Dán. Cache-to-Cache: Could ISPs Cooperate to Decrease Peer-to-peer Content Distribution Costs?

[6] G. Dán. Existence and convergence to equilibrium in replication games. *KTH Tech. Rep.*, TRITA-EE 2010:020, 2010.

[7] P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6(26):290–297, 1959.

[8] S. Kakutani. A generalization of brouwer's fixed point theorem. *Duke Math. J.*, 1941.

[9] V. Krishna. Learning in games with strategic complementarities. *Preprint*.

[10] P. L'ecuyer, R. Simard, E.J. Chen, and W.D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075, 2002.

[11] I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13(1):111–124, 1996.

[12] P. Milgrom and J. Roberts. Adaptive and sophisticated learning in normal form games* 1. *Games and economic Behavior*, 3(1):82–100, 1991.

[13] D. Monderer and A. Sela. Fictitious play and no-cycling conditions. *Sonderforschungsbereich 504 Publications*, 1997.

[14] D. Monderer and L.S. Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68(1):258–265, 1996.

[15] D. Monderer and L.S. Shapley. Potential games. *Games and economic behavior*, 14:124–143, 1996.

[16] J.F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1950.

[17] J.F. Nash. Non-cooperative games. *Annals of mathematics*, 1951.

[18] J. Neumann and O. Morgenstern. Theory of games and economic behavior, 1944.

[19] M.J. Osborne. *An introduction to game theory*. Oxford University Press New York, NY, 2004.

[20] M.J. Osborne and A. Rubinstein. *A course in game theory*. The MIT press, 1994.

[21] J. Robinson. An iterative method of solving a game. *Annals of mathematics*, 54(2):296–301, 1951.

[22] R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[23] J.J. Rousseau. Discourses on the Origins of Inequality, ed. *G. Cole, London: Dent*, 1754.

[24] L.J. Savage. *The foundations of statistics*. Dover Pubns, 1972.

[25] H.P. Young. The evolution of conventions. *Econometrica: Journal of the Econometric Society*, 61(1):57–84, 1993.

List of Figures

List of Tables