Libera Università Internazionale degli Studi Sociali "Guido Carli"

Department of Economics and Finance
Chair of Games and Strategies

The buck-passing game: simulation of adapting learning dynamics

SUPERVISOR

Professor Roberto Cominetti

CANDIDATE

Filippo Allegri
Matr. 215331

ACADEMIC YEAR 2018/2019

Contents

1 Introduction **3**

2 The Buck-Passing Game **5**
2.1 The deterministic buck-passing game 6
2.2 The stochastic buck-passing game 7
2.3 Nash equilibria 9
2.3.1 Deterministic buck-passing game 9
2.3.2 Stochastic buck-passing game 9
2.4 Fairness of equilibria 10

3 Players: intelligence and behaviour **11**
3.1 The "learn and decide framework" 12
3.2 The learning process 13
3.2.1 Alpha (α) 14
3.2.2 Arithmetic average 14
3.3 The decision-making process 15
3.3.1 Deterministic game: the "chosen neighbour" 17
3.3.2 Stochastic game: Beta (β) 18

4 Simulation **20**
4.1 Computer programming: Python 21

5 The 3-player Network **22**
5.1 Theory 23
5.2 Simulation - Deterministic 26
5.3 Simulation - Stochastic 28
5.3.1 Beta (β) and Equilibria 31
5.3.2 Stable states switching 35

6 Cycle Network **37**
6.1 Nash Equilibria 38
6.1.1 Fairness of equilibria 39
6.2 Simulation - Deterministic 40
6.3 Simulation - Stochastic 42

7 Complete Network **45**
7.1 Nash Equilibria 46
7.2 Simulation - Deterministic 49

8 Conclusions **52**

9 Glossary **54**

10 References **57**

11 Appendix **58**
11.1 Python codes: Cycle Network and 3-player Network 58
11.2 Python codes: Complete Network 62

1 Introduction

During everyday life, humans face tasks that vary in complexity and length. Sometimes we are willing to perform them, other times we try to get rid of them. The main focus of this thesis will not be on problems that, even if challenging, we are willing to solve, but instead on the ones we would like to avoid.

Under an economic perspective, these can be described as tasks in which the energy required to reach a solution does not counterbalance the utility obtained from it. According to economic theory, any rational agent who faces a (non-mandatory) task which provides a negative overall utility would be happy to give it up. However, this is not always possible. For example, suppose to be a doctor and to have a patient affected by a very strong form of hypochondria. After many analyses, you find out that the patient is in very good health, but she is still suspicious and does not believe in your results. Now, you can either carry out new investigations (which will cost you a considerable amount of time and may not be accepted) or suggest your patient to appoint another doctor, a "specialist". If you consider the case to be a negative utility task, the second solution is the best. However, delegating the case cannot be considered a final solution. In fact, the specialist, if unable to satisfy the patient, may consider sending the case back to you or to a third doctor (an "expert") which may eventually send it back to you. Figure 1 shows the situation under a graphical viewpoint. If both, "specialist" and "expert", are not able to find any remedies, the patient will be sent back to you, thus replicating the initial situation.

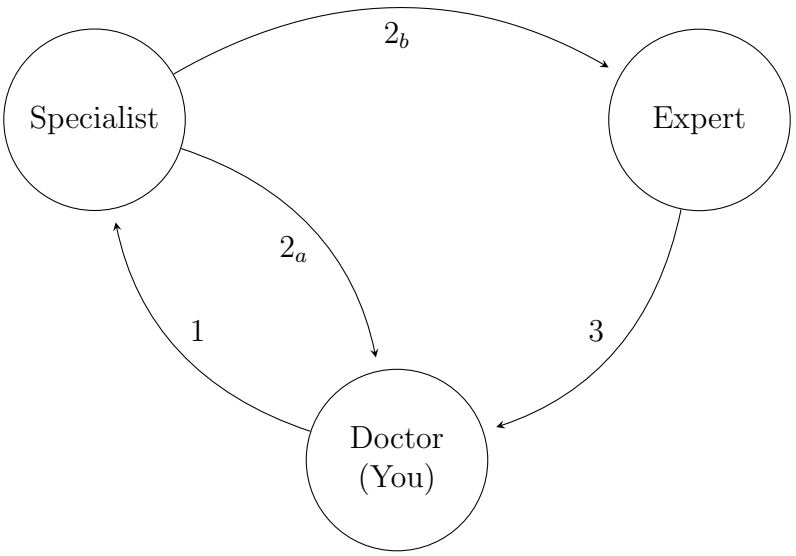


Figure 1: Graphical representation of the "Doctor/Specialist/Expert" situation

In the case in which all the agents in the network believe that performing the task brings negative utility, you should figure out that the patient is very likely to be sent back to you in the future. Therefore, your target should switch from getting rid of the patient to receiving the patient back as rarely as possible. For example, if you think that the expert and the specialist will act as follows:

$$\begin{cases} \text{Specialist} \Rightarrow \text{pass the patient to the expert} \\ \text{Expert} \Rightarrow \text{pass the patient to you,} \end{cases} \tag{1.1}$$

to whom should you pass the patient? As shown in figure 2, if you "pass" the patient to the expert, you will receive her back straight after, while if you "pass" the patient to the specialist, you will receive her back after a longer amount of time. Your best strategy would be to "pass" the patient to the specialist.

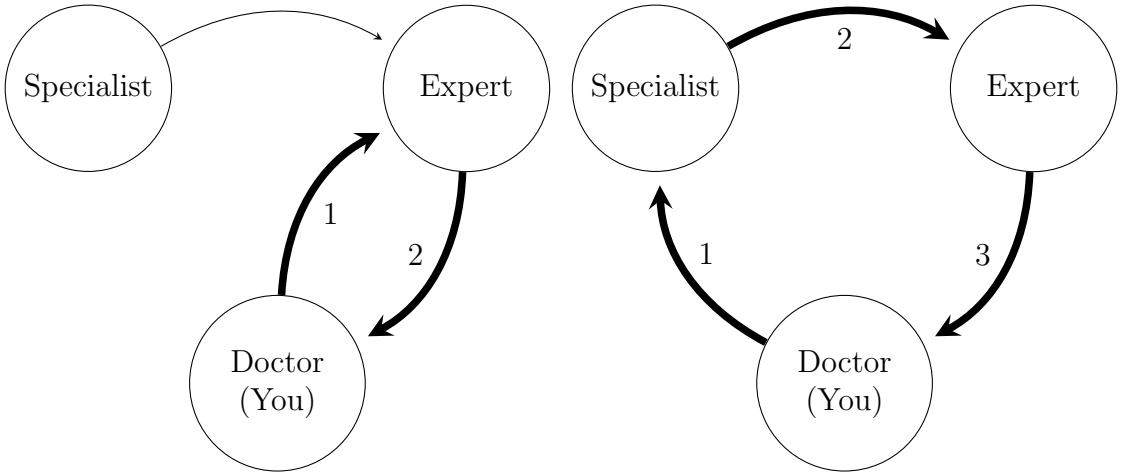


Figure 2: Graphical representation of the two strategies and outcomes of the doctor in the "Doctor/Specialist/Expert" situation

In real life, the total number of possible doctors, specialists, and experts to which the case can be "passed" is much higher than three. However, assuming that no one will be able to satisfy the patient and assuming infinite medical visits, you are very likely to receive the patient back at a certain point in time.

Many real-life situations follow a structure similar to the one described above and can all be represented by a unique model: the buck-passing game.

2 The Buck-Passing Game

The Buck-Passing Game is a network game in which agents want to transfer the responsibility of performing a task to one of their neighbors in a social network. The game has been defined for the first time in August 2018 by Roberto Cominetti, Matteo Quattropani, and Marco Scarsini in the paper "The Buck-Passing Game[1]".

The name of the game comes from the English language expression "Passing the Buck". The phrase became popular in the United States in the 1950s when President Harry S. Truman used to keep a sign with the sentence "The buck stops here" on his desk in the Oval Office. However, the expression has originated from the game of poker during the American Frontier era. At the time, knives with a buckhorn handle (called "bucks") were common and were often used as counters to recognise the person whose turn it was to deal. If a player did not want to take the responsibility to bet, she had to pass the buck to the next player. Since then, the expression "passing the buck" indicates the act of assigning personal responsibility to another person[2].

The general model of the Buck-Passing game consists of a finite number of players represented as vertices of a directed graph. The game begins by randomly assigning the buck to one of the players, following an established distribution. Each player keeps the buck during the period in which it is received and passes it in the following period. The agents pass the buck to each other for an infinite amount of periods (the game has no end). The agents aim to receive the buck back as rarely as possible or, more formally, to minimise personal costs. The costs that a player incurs in are represented by the expected long term frequency of periods in which the player has the buck.

The buck-passing game can be analysed under two viewpoints: the deterministic one and the stochastic one.

[1]Cominetti, Roberto; Quattropani, Matteo; and Scarsini, Marco "The Buck-Passing Game" Research Paper. Accessed May, 2019.

[2]Mitford M. Mathews, ed., A Dictionary of Americanisms on Historical Principles (Chicago, University of Chicago Press, 1951), I, pages 198-199.

2.1 The deterministic buck-passing game

In the deterministic version of the buck-passing game, every player chooses a neighbour. Whenever a player gets the buck, she will pass it to the "chosen neighbour".

Formally, a deterministic buck-passing game Ψ is defined in the form $\Psi = (\mathscr{G}, \mu)$. $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is a directed graph in which $\mathscr{V} = \{1, ..., n\}$ is the set of vertices and $\mathscr{E}$ is the set of edges. μ is a probability measure of $\mathscr{V}$. In the game, $\mathscr{V}$ is the set of players. For $i \in \mathscr{V}$, call $\mathscr{N}_i^+$ the set of out-neighbours of player i in $\mathscr{G}$ such that $\mathscr{N}_i^+ = \{k \mid k \in \mathscr{V} : \ (i,k) \in \mathscr{E}\}$. The set of strategies of player i is $S_i = \mathscr{N}_i^+$. $S = \times_{i \in \mathscr{V}} S_i$ is the set of strategy profiles. At time $t = 0$ one vertex $i \in \mathscr{V}$ is drawn at random according to μ. Once that all the players have set their strategies and that Nature has drawn one player according to μ, the selected player receives the buck. Every time player $i \in \mathscr{V}$ receives the buck, she will pass it to player $j \in \mathscr{V} \setminus i$ where player j is the "chosen neighbour" stated in the strategy of player i. Given a strategy profile $\boldsymbol{s}$, the random variable $\Theta_{i,t}$ is defined as follows:

$$\Theta_{i,t}(\boldsymbol{s}) = \begin{cases} 1 & \text{if at time } t \text{ Player } i \text{ has the buck} \\ 0 & \text{otherwise,} \end{cases} \tag{2.1}$$

with

$$\mu_i = \mathbb{P}(\Theta_{i,0}(\boldsymbol{s}) = i). \tag{2.2}$$

For $i \in \mathscr{V}$ and given a strategy profile $\boldsymbol{s}$, the cost function $c_i : S \to \mathbb{R}$ is defined as follows:

$$c_i(\boldsymbol{s}) = \mathbb{E}\left[\lim_{T \to \infty} \left[\frac{1}{T} \sum_{t=0}^{T} \Theta_{i,t}(\boldsymbol{s}) \right] \right], \tag{2.3}$$

where the expectation is calculated with respect to the initial measure μ.

The profile of cost functions is denoted by $\boldsymbol{c} = (c_1, ..., c_n)$. Moreover, the costs of all players add up to 1 and thus the buck-passing game is equivalent to a zero-sum game.

2.2 The stochastic buck-passing game

In the stochastic version of the buck-passing game, players randomise the choice between neighbours. Formally, a stochastic buck-passing game Υ is defined in the form $\Upsilon = \{\mathscr{G}, S, \mu\}$. As for the deterministic buck-passing game, $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is a directed graph in which $\mathscr{V} = \{1, ..., n\}$ is the set of vertices, $\mathscr{E}$ is the set of edges and μ is a probability measure of $\mathscr{V}$. $S = \times_{i \in \mathscr{V}} S_i$ is the set of strategy profiles.

In a stochastic buck-passing game, the strategy selected by player i is identified by the n-probability vector $\boldsymbol{\phi}_i$:

$$\boldsymbol{\phi}_i = \begin{bmatrix} \phi_{i,1} & \phi_{i,2} & \dots & \phi_{i,j} & \dots & \phi_{i,n} \end{bmatrix}, \tag{2.4}$$

in which $\phi_{i,k} \geq 0$ with:

$$\sum_{k=1}^{n} \phi_{i,k} = 1, \tag{2.5}$$

and $\phi_{i,k} = 0$ if $(i,k) \notin \mathscr{E}$.

Therefore, the strategy profile Φ can be identified with a transition matrix whose rows are the n-probability vectors of each player ($\boldsymbol{\phi}_1$, ..., $\boldsymbol{\phi}_n$) for which the same symbol is used:

$$\Phi = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \dots & \phi_{1,j} & \dots & \phi_{1,n} \\ \phi_{2,1} & \phi_{2,2} & \dots & \phi_{2,j} & \dots & \phi_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{i,1} & \phi_{i,2} & \dots & \phi_{i,j} & \dots & \phi_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_{n,1} & \phi_{n,2} & \dots & \phi_{n,j} & \dots & \phi_{n,n} \end{bmatrix}, \tag{2.6}$$

in which:

$$\phi_{i,j} = \mathbb{P}(\Theta_{j,t+1} = 1 | \Theta_{i,t} = 1). \tag{2.7}$$

For $i \in \mathscr{V}$, the set of strategies of player i is S_i, which is the set of all the possible n-probability vectors $\boldsymbol{\phi}_i$.

At time $t = 0$, Nature draws one player at random according to the measure μ. Whenever player $i \in \mathscr{V}$ receives the buck, she passes it to a random neighbour decided accordingly to $\boldsymbol{\phi}_i \in S_i$. The process is represented by the **Markov chain** $(X_t^{\Phi})_{t\geq 0}$ with transition matrix Φ and initial measure μ.

> **Markov chain**: A discrete stochastic process in which the probabilities of occurrence of various future states depend only on the present state of the system or on the immediately preceding state and not on the path by which the present state was achieved.[3]

For $i \in \mathscr{V}$ and given a strategy profile Φ, the cost function $c_i : S \to \mathbb{R}$ is defined as follows:

$$c_i(\Phi) = \mathbb{E}\left[\lim_{T\to\infty}\left[\frac{1}{T}\sum_{t=0}^{T}\Theta_{i,t}(\Phi)\right]\right], \tag{2.8}$$

where the expectation is calculated with respect to both the initial measure μ and the transition matrix Φ.

The profile of cost functions is denoted by $\boldsymbol{c} = (c_1, ..., c_n)$. Moreover, the stochastic buck-passing game, as the deterministic one, is equivalent to a zero-sum game as the costs of all players add up to 1.

[3] "Markov Chain." *Dictionary by Merriam-Webster*, Merriam-Webster, 2019, www.merriam-webster.com/dictionary/Markov%20chain.

2.3 Nash equilibria

A Nash Equilibrium is defined as a stable state of a system involving the interaction of different participants, in which no participant can gain by a unilateral change of strategy if the strategies of the others remain unchanged.[4]

2.3.1 Deterministic buck-passing game

A strategy profile $\boldsymbol{s}^*$ is a Nash equilibrium (NE) if for all players $i \in \mathscr{V}$ and for all strategies $s_i \in S_i$ the following inequality holds:

$$c_i(\boldsymbol{s}^*) \leq c_i(s_i, \boldsymbol{s}^*_{-i}), \tag{2.9}$$

where $\boldsymbol{s}^*_{-i}$ is the sub-profile of strategies of all players different from i.

2.3.2 Stochastic buck-passing game

A strategy profile Φ^* is a Nash equilibrium (NE) if for all players $i \in \mathscr{V}$ and for all strategies $\boldsymbol{\phi}_i \in S_i$ the following inequality holds:

$$c_i(\Phi^*) \leq c_i(\boldsymbol{\phi}_i, \Phi^*_{-i}), \tag{2.10}$$

where Φ^*_{-i} is the sub-profile of strategies of all players different from i.

[4] "Nash Equilibrium" *Oxford Reference*, www.oxfordreference.com, 2019, www.oxfordreference.com/view/10.1093/oi/authority.20110803100223327

2.4 Fairness of equilibria

In the buck-passing game, different equilibria may occur. In some, the costs are almost evenly spread among all the players, in others, just a small portion of the players incurs in all the costs. The fairness (or efficiency) of an equilibrium depends on how the costs are spread.

To represent social fairness, the social cost (SC) of a strategy profile $\boldsymbol{s}$ (for a deterministic buck-passing game) or Φ (for a stochastic buck-passing game) is defined as equal to the highest personal cost incurred by any of the players:

$$\mathrm{SC}(\boldsymbol{s}) = \max_{i \in V} c_i(\boldsymbol{s}); \qquad \mathrm{SC}(\Phi) = \max_{i \in V} c_i(\Phi). \tag{2.11}$$

Fairness is measured through the price of anarchy (PoA) and the price of stability (PoS):

$$\mathrm{PoA} = \frac{\max_{\boldsymbol{s} \in \mathrm{NE}} \mathrm{SC}(\boldsymbol{s})}{\min_{\boldsymbol{s} \in S} \mathrm{SC}(\boldsymbol{s})}; \qquad \mathrm{PoA} = \frac{\max_{\Phi \in \mathrm{NE}} \mathrm{SC}(\Phi)}{\min_{\Phi \in S} \mathrm{SC}(\Phi)} \tag{2.12}$$

$$\mathrm{PoS} = \frac{\min_{\boldsymbol{s} \in \mathrm{NE}} \mathrm{SC}(\boldsymbol{s})}{\min_{\boldsymbol{s} \in S} \mathrm{SC}(\boldsymbol{s})}; \qquad \mathrm{PoS} = \frac{\min_{\Phi \in \mathrm{NE}} \mathrm{SC}(\Phi)}{\min_{\Phi \in S} \mathrm{SC}(\Phi)}. \tag{2.13}$$

3 Players: intelligence and behaviour

Intelligence is defined by the Cambridge Dictionary as:

> *"The ability to learn, understand, and make judgments or have opinions that are based on reason"*[5]

The *conditio sine qua non* of intelligence is learning from experiences. It would be impossible to learn or understand something which has not been experienced. It follows that there is a strong correlation between experience, intelligence, and good performances.

Experience has a relevant role in every strategic game. Usually, players with a greater amount of experience tend to perform better than beginners. The story of Daniel Negreanu, one of the best poker players in the world, represents a significant example. In the circuit, he is known as "the mind-reader" for his exceptional ability in predicting his opponents' actions. Probably he does not have any super-powers, but surely he has a lot of experience. Throughout the years he has been able to develop the ability to read other players' moves and understand what poker hands they might hold. Clearly, Daniel Negreanu is a better player now than he was when he played his first game and, like him, poker players usually improve over time. This concept applies to the buck-passing game players as well.

In the buck-passing game, the aim of every player is receiving the "buck" back as rarely as possible, thus minimising personal costs c_i. Assuming that all the players are intelligent and behave rationally, it would be reasonable to expect them to adapt their strategy during the game pursuing lower costs. More precisely:

- In the deterministic version: player $i \in \mathscr{V}$ is allowed to substitute the "chosen neighbor" from S_i.
- In the stochastic version: player $i \in \mathscr{V}$ is allowed to modify the probabilities in the row vector $\boldsymbol{\phi}_i$ (according to conditions (2.4) and (2.5))

Given that players can modify their strategies during the game, when and how should they do that? A clear answer to this question is vital in order to properly model the real-life buck-passing game. To do so, I developed the "learn and decide framework".

[5] "Intelligence" *Cambridge Dictionary*, Dictionary.cambridge.org, 2019, dictionary.cambridge.org/it/dizionario/inglese/intelligence.

3.1 The "learn and decide framework"

Intelligence is not only about learning from experiences, but it is also about making judgments and taking decisions (as stated by its definition). The "learn and decide framework" combines these two aspects. In this thesis, players' intelligence is assumed to be structured as shown in Figure 3:

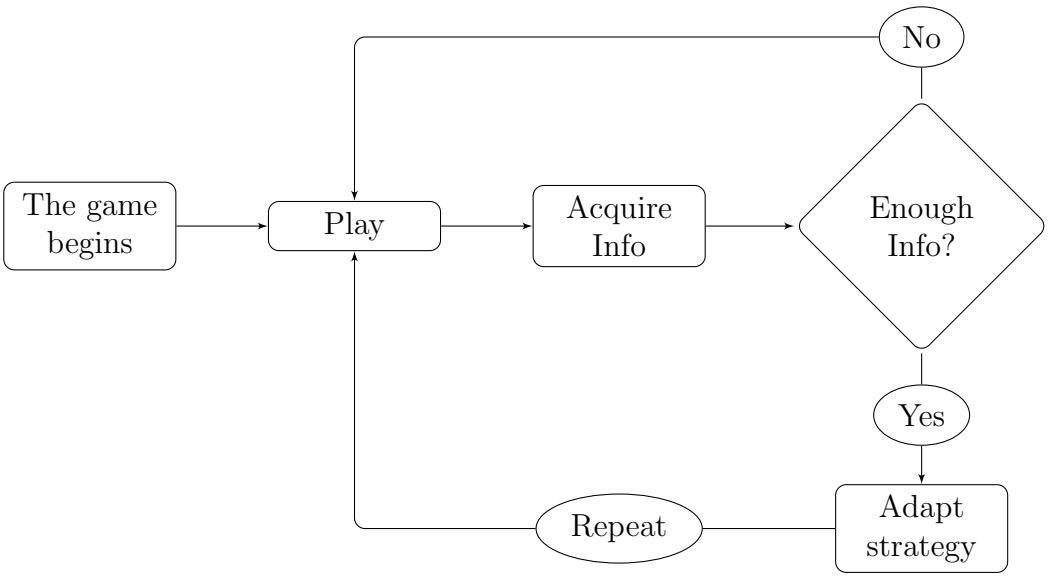


Figure 3: Flow chart representing the "learn and decide framework"

in which a player has "Enough Info" if she has already passed the buck at least once to any available neighbour.

I decided to do not let players adjust their strategies before having tested all their passing possibilities in order to limit as much as possible situations in which an initial misjudgment of the player leads to an unprofitable strategy (wrongly believed to be the best by the player). The diagram shows that once that a player has collected information about all neighbours, the strategy is adapted continuously. Throughout the game, more and more information is acquired allowing players to have an increasing amount of data to rely on.

The whole intelligence process may be divided into two parts: the learning process ("Acquire Info" on the diagram) and the decision-making process ("Adapt strategy" on the diagram)

3.2 The learning process

The learning process represents the ground on which the decision-making process is built. In this stage, players transform information about the game into knowledge. The "Forecast matrix", Ω, represents the forecasts of the players regarding the number of periods the buck will take to come back if passed to each of the neighbours:

$$\Omega = \begin{bmatrix} \omega_{1,1} & \omega_{1,2} & \dots & \omega_{1,j} & \dots & \omega_{1,n} \\ \omega_{2,1} & \omega_{2,2} & \dots & \omega_{2,j} & \dots & \omega_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega_{i,1} & \omega_{i,2} & \dots & \omega_{i,j} & \dots & \omega_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega_{n,1} & \omega_{n,2} & \dots & \omega_{n,j} & \dots & \omega_{n,n} \end{bmatrix}, \tag{3.1}$$

in which $\omega_{i,j}$ is equal to the forecast of player i about the number of periods the buck will take to come back if passed to player j.

The buck-passing game does not allow the players to pass the buck to themselves. Therefore, for any $k \in \mathscr{V}$, the values of $\omega_{k,k}$ are not defined:

$$\Omega = \begin{bmatrix} - & \omega_{1,2} & \dots & \omega_{1,j} & \dots & \omega_{1,n} \\ \omega_{2,1} & - & \dots & \omega_{2,j} & \dots & \omega_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega_{i,1} & \omega_{i,2} & \dots & \omega_{i,j} & \dots & \omega_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \omega_{n,1} & \omega_{n,2} & \dots & \omega_{n,j} & \dots & - \end{bmatrix}. \tag{3.2}$$

The set of forecasts of player i is represented by the row vector $\boldsymbol{\omega}_i$:

$$\boldsymbol{\omega}_i = \begin{bmatrix} \omega_{i,1} & \omega_{i,2} & \dots & \omega_{i,j} & \dots & \omega_{i,n} \end{bmatrix}. \tag{3.3}$$

The matrix and the vectors do not represent the real expectations but just the forecasts of the players. These forecasts are based on information acquired while experiencing the game. For example, if the first time that player 1 passed the buck to player 2, it took three periods to come back, it seems reasonable to think that player 1 expectations at that point were probably $\omega_{1,2} = 3$.

Establishing a clear model representing how players' expectations change over time is important. In this thesis, two methods will be used: a moving average and an arithmetic average. The moving average model uses a parameter to represent the memory of the players: α.

3.2.1 Alpha (α)

Alpha is a coefficient included between 0 and 1 which represents players' memory. How do players update their forecasts as the game goes on and more information is obtained? They do so according to their α. A low α is attributed to players which tend to give more importance to older observations, while a high α indicates that the player will give more importance to the most recent observations, forgetting the past easily.

Given that ι is the new piece of information obtained (which is an observation of the time required for the buck to come back) and t is the period, player i updates her forecasts about passing the buck to player j as follows:

$$\omega_{i,j}(t=k) = \Big[\iota \times \alpha\Big] + \Big[\omega_{i,j}(t=k-1) \times (1-\alpha)\Big]. \tag{3.4}$$

Note that equation (3.4) is a weighted moving average between forecast and new observation in which the new observation has a weight of α.

3.2.2 Arithmetic average

When the arithmetic average model is adopted, players' forecasts are calculated according to the following formula:

$$\omega_{i,j}(t=k) = \Big[\iota \times \frac{1}{h_{i,j}(t=k)}\Big] + \Big[\omega_{i,j}(t=k-1) \times \Big(1 - \frac{1}{h_{i,j}(t=k)}\Big)\Big], \tag{3.5}$$

in which $h_{i,j}(t=k)$ is equal to the number of times that player i has passed the buck to player j at time $t=k$.

3.3 The decision-making process

In this stage, the players adapt their strategies according to their knowledge. The "Forecast matrix" (Ω), described in the previous section, is used to derive the "Probability matrix", P:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots & p_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,j} & \cdots & p_{n,n} \end{bmatrix}, \tag{3.6}$$

in which, for $i, j \in \mathscr{V}$:

$$p_{i,j} = \mathbb{P}(\Theta_{j,t+1} = 1 | \Theta_{i,t} = 1), \tag{3.7}$$

i.e. the probability that, once that player i has received the buck, she will pass it to player j.

The buck-passing game does not allow the players to pass the buck to themselves. Therefore, for any $k \in \mathscr{V}$, $p_{k,k} = 0$:

$$P = \begin{bmatrix} 0 & p_{1,2} & \cdots & p_{1,j} & \cdots & p_{1,n} \\ p_{2,1} & 0 & \cdots & p_{2,j} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots & p_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,j} & \cdots & 0 \end{bmatrix}. \tag{3.8}$$

The strategy of player i is represented by the vector $\boldsymbol{p}_i$:

$$\boldsymbol{p}_i = \begin{bmatrix} p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots & p_{i,n} \end{bmatrix}, \tag{3.9}$$

which is equal to row i of matrix P.

In both, the deterministic and the stochastic version of the game, the "Probability matrix" is adopted by players. However, there are consistent differences in the way in which the probabilities are calculated.

3.3.1 Deterministic game: the "chosen neighbour"

In the deterministic game, the players chose a neighbour to which always pass the buck. Therefore, for $i \in \mathscr{V}$, the vector $\boldsymbol{p}_i$ is composed of all "0" entries except for one "1" entry, according to the following formula:

$$p_{i,j} = \begin{cases} 1 & \text{if player } j \text{ is the chosen neighbour of player } i \\ 0 & \text{Otherwise} \end{cases} \tag{3.10}$$

As stated at the beginning of section 3, we assume that all the players are intelligent and behave rationally. Therefore, they should choose the "chosen neighbour" as the one which minimizes personal costs c_i (according to the knowledge they have built during the game). Player $i \in \mathscr{V}$ assigns a probability equal to 1 only to the neighbour corresponding to the higher entry in her forecast vector, $\boldsymbol{\omega}_i$. The "Probability matrix" is continuously updated according to the evolution of the "Forecast matrix" as follows:

$$p_{i,j} = \begin{cases} 1 & \text{if } \omega_{i,j} = \max_{k \in \mathscr{V}} \boldsymbol{\omega}_{i,k} \\ 0 & \text{otherwise} \end{cases} . \tag{3.11}$$

Note: In the case in which two or more entries of vector $\boldsymbol{\omega}_i$ are equal to $\max_{k \in \mathscr{V}} \boldsymbol{\omega}_{i,k}$, the "chosen neighbour" of player i is randomly decided among the ones with the highest equal forecasts.

Example 3.1:

Player 1's forecasts are represented by the following vector:

$$\boldsymbol{\omega}_1 = \begin{bmatrix} - & 5 & 3 & 4 \end{bmatrix}.$$

According to ($\boldsymbol{\omega}_1$), the strategy that minimizes c_1 is "pass the buck to player 2". This forecast is reflected in the probability vector ($\boldsymbol{p}_1$) in which the "chosen neighbour" is player 2:

$$\boldsymbol{p}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}.$$

3.3.2 Stochastic game: Beta (β)

In the stochastic game, a player's strategy consists of a set of probabilities assigned. Always assuming the agents to be intelligent and rational, it is reasonable to build a system in which the higher the forecast about the returning time of a player, the higher the probability to pass the buck to that player. In order to respect this principle, I decided to adopt the following method:

$$p_{i,j} = \begin{cases} \frac{e^{\omega_{i,j}}}{\sum_{k \in \mathscr{N}_i^+} e^{\omega_{i,k}}} & \text{if} \quad j \in \mathscr{N}_i^+ \\ \\ 0 & \text{if} \quad j \notin \mathscr{N}_i^+ \end{cases} . \tag{3.12}$$

However, formula (3.12) does not allow to characterize the players with different mindsets. I decided to add a variable to the function in order to make a distinction between players' behaviours. Beta (β) is a coefficient greater or equal to zero. A value close to zero is given to a player which does not commit too much to her knowledge and tends to evenly randomise the choice. A high value corresponds to a player which strongly believes in the data collected and tends to rely on his best strategy giving less space to the exploration of others. For a value of β between 1 and 2, the player's behaviour tends to be more balanced.

In a stochastic buck-passing game, the "Probability matrix" is continuously updated according to the evolution of the "Forecast matrix" and to the following formula in which β is implied:

$$p_{i,j} = \begin{cases} \frac{e^{\beta\omega_{i,j}}}{\sum_{k \in \mathscr{N}_i^+} e^{\beta\omega_{i,k}}} & \text{if} \quad j \in \mathscr{N}_i^+ \\ \\ 0 & \text{if} \quad j \notin \mathscr{N}_i^+ \end{cases} . \tag{3.13}$$

Example 3.2: This example shows how the same forecasts ($\boldsymbol{\omega}_1$) lead to different strategies in the stochastic game.

Player 1's forecasts are represented by the following vector (the same of Example 3.1) and the β coefficient is set:

$$\boldsymbol{\omega}_1 = \begin{bmatrix} - & 5 & 3 & 4 \end{bmatrix}; \quad \beta = 1.$$

The probability vector $\boldsymbol{p}_1$ is calculated according to equation (3.13) as follows:

$$p_{1,2} = \frac{e^{1\times 5}}{\sum_{k\in\mathcal{N}_i^+} e^{1\times\omega_{1,k}}} \approx 0.67;$$

$$p_{1,3} = \frac{e^{1\times 3}}{\sum_{k\in\mathcal{N}_i^+} e^{1\times\omega_{1,k}}} \approx 0.09;$$

$$p_{1,4} = \frac{e^{1\times 4}}{\sum_{k\in\mathcal{N}_i^+} e^{1\times\omega_{1,k}}} \approx 0.24;$$

$$\boldsymbol{p}_1 = \begin{bmatrix} 0 & 0.67 & 0.09 & 0.24 \end{bmatrix} \tag{3.14}$$

In the example, the simple strategy that minimizes c_1 according to $\boldsymbol{\omega}_1$ is "pass the buck to player 2". The calculated probability vector $\boldsymbol{p}_1$ shows that player 1 will most likely pass the buck to player 2. However, passing the buck to player 3 and player 4 are not excluded possibilities as they were in the deterministic version of the game (Example 3.1).

4 Simulation

The major problem of many scientific pieces of research is the lack of testing of the theory developed throughout the paper. This thesis uses experimental activities in order to discuss the theory.

The buck-passing game is complicated but most importantly, it has no end. Predicting the equilibrium strategy profile adopted by the players when $t = \infty$ is impossible, but exploring how it changes and develops through time, up to very large values of t can reveal useful information about the trending strategies. Understanding the trend has particular relevance in calculating costs. In fact, being c_i an expectation of the frequency, its value gets more stable as t gets bigger, approaching asymptotically its final value (for $t = \infty$). For example, given a strategy profile Φ and $T' = 10^{10}$, it is likely that:

$$\frac{1}{T'}\left[\sum_{t=0}^{T'}\Theta_{i,t}(\Phi)\right] = \frac{1}{10^{10}}\left[\sum_{t=0}^{10^{10}}\Theta_{i,t}(\Phi)\right] \approx \mathbb{E}\left[\lim_{T\to\infty}\left[\frac{1}{T}\sum_{t=0}^{T}\Theta_{i,t}(\Phi)\right]\right] \tag{4.1}$$

In light of this, reproducing the buck-passing game through an experiment that involves real people may seem reasonable. However, even if equation (4.1) shows that a deep study of the game does not require it to keep going forever, it also underlines the fact that precise results require a high number of recorded periods. Reaching a reasonable amount of played periods with a real experiment requires many players and hundreds of hours. These conditions make such an experiment infeasible, especially considering that it must be repeated multiple times in order to analyse different aspects.

Simulating the buck-passing game overcomes all these problems. A standard computer is able to process millions of periods in a few seconds. Moreover, the experiment can be repeated easily, allowing us to explore different game conditions.

4.1 Computer programming: Python

Python is one of the most common programming languages. It is simple, easy to code, and widely understood. These qualities make it suitable for the purpose of this thesis. Nevertheless, this paper does not focus on programming but uses it as a tool to test the theory.

I developed different buck-passing game simulators, each suitable for a different type of situation[6]. They take into consideration the characteristics of the network and the version (deterministic/stochastic). The programs require:

1. $\mathscr{G} = (\mathscr{V}, \mathscr{E})$: The directed graph $\mathscr{G}$ is the network on which the game takes place. In the program, it is represented by an **adjacency matrix**.
2. T: The number of periods the simulation will run for.
3. α: The "Alpha coefficient"
4. β: The "Beta coefficient"

Adjacency matrix[7]: A matrix used as a means of representing an adjacency structure, a graph. If A is the adjacency matrix corresponding to a given directed graph $\mathscr{G}$, then:

$$a_{i,j} = \begin{cases} 1 & \text{if there is an edge directed from vertex } i \text{ to vertex } j \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

[6] The transcription of the programs can be found in Appendix

[7] "Adjacency Matrix" Oxford Reference, www.oxfordreference.com, 2019, www.oxfordreference.com/view/10.1093/oi/authority.20110803095351444

5 The 3-player Network

The first network that will be analysed is the 3-player network. It is one of the simplest diagrams on which a buck-passing game can be played. It is made by three players connected bilaterally to each other.

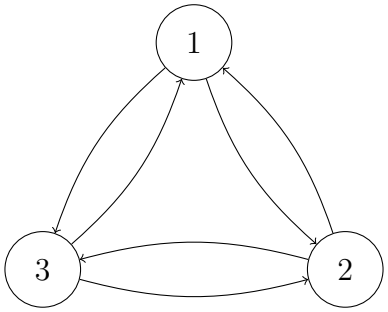


Figure 4: The 3-player Network

The adjacency matrix of the 3-player network is:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}. \tag{5.1}$$

5.1 Theory

A buck-passing developed on this network structure can be transcribed in strategic form as:

player 3 $\Rightarrow P_1$

player 1 \ player 2	$\Rightarrow P_1$	$\Rightarrow P_3$
$\Rightarrow P_2$	$(\frac{1}{2}, \frac{1}{2}, 0)$	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
$\Rightarrow P_3$	$(\frac{1}{2}, 0, \frac{1}{2})$	$(\frac{1}{2}, 0, \frac{1}{2})$

player 3 $\Rightarrow P_2$

player 1 \ player 2	$\Rightarrow P_1$	$\Rightarrow P_3$
$\Rightarrow P_2$	$(\frac{1}{2}, \frac{1}{2}, 0)$	$(0, \frac{1}{2}, \frac{1}{2})$
$\Rightarrow P_3$	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$	$(0, \frac{1}{2}, \frac{1}{2})$

Table 1: The 3-player Network in strategic form

in which "$\Rightarrow P_i$" stands for "Pass the buck to player i".

For this type of network, there are two Nash equilibria in pure strategy (shown in bold in Figure 5):

$$\text{NE} = (\Rightarrow P_2, \Rightarrow P_3, \Rightarrow P_1); (\Rightarrow P_3, \Rightarrow P_1, \Rightarrow P_2) \tag{5.2}$$

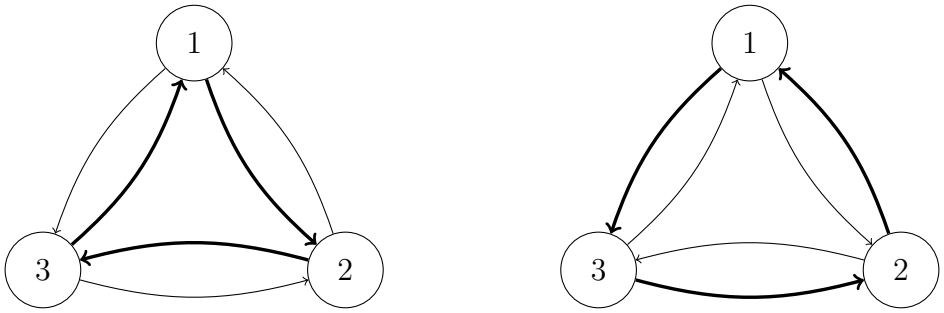


Figure 5: Graphical representation of the Nash equilibria in pure strategy

In both cases, all the strategies of players create a **Hamiltonian cycle**: a particular path of the buck in which each player receives it and passes it exactly once (before starting again the cycle).

The mixed strategy equilibrium can be obtained through "best response" analysis:

$$\begin{cases} \mathbb{E}(\text{player } 1 \Rightarrow P_2) = \mathbb{E}(\text{player } 1 \Rightarrow P_3) \\ \mathbb{E}(\text{player } 2 \Rightarrow P_1) = \mathbb{E}(\text{player } 2 \Rightarrow P_3) \\ \mathbb{E}(\text{player } 3 \Rightarrow P_1) = \mathbb{E}(\text{player } 3 \Rightarrow P_2) \end{cases}, \tag{5.3}$$

in which "$\mathbb{E}(\text{player } i \Rightarrow P_j)$" is the expected number of periods that the buck will take to come back to player i if player i passes it to player j.

The system of equations (5.3) leads to the probabilities adopted by the players in the mixed strategy Nash Equilibrium. There is only one solution to the system and the third Nash Equilibrium is:

$$\text{NE} = \left[\left(\frac{1}{2};\frac{1}{2}\right), \left(\frac{1}{2};\frac{1}{2}\right), \left(\frac{1}{2};\frac{1}{2}\right)\right] \tag{5.4}$$

Note that all the Nash equilibria found in this section are symmetric. In fact, given that:

$$\text{NE} = \left[\left(p_{1,2}; 1-p_{1,2}\right), \left(1-p_{2,3}; p_{2,3}\right), \left(p_{3,1}; 1-p_{3,1}\right)\right], \tag{5.5}$$

a Nash equilibrium occur every time one of the following cases verifies ($p_{i,j}$ eq. (3.7)):

$$a: \;\; p_{1,2} = p_{2,3} = p_{3,1} = 1 \;\; \Rightarrow \;\; \text{NE} = \left[\left(1;0\right), \left(0;1\right), \left(1;0\right)\right] = (\Rightarrow P_2, \Rightarrow P_3, \Rightarrow P_1)$$

$$b: \;\; p_{1,2} = p_{2,3} = p_{3,1} = \frac{1}{2} \;\; \Rightarrow \;\; \text{NE} = \left[\left(\frac{1}{2};\frac{1}{2}\right), \left(\frac{1}{2};\frac{1}{2}\right), \left(\frac{1}{2};\frac{1}{2}\right)\right]$$

$$c: \;\; p_{1,2} = p_{2,3} = p_{3,1} = 0 \;\; \Rightarrow \;\; \text{NE} = \left[\left(0;1\right), \left(1;0\right), \left(0;1\right)\right] = (\Rightarrow P_3, \Rightarrow P_1, \Rightarrow P_2).$$

The following figure shows the three Nash equilibria graphically.

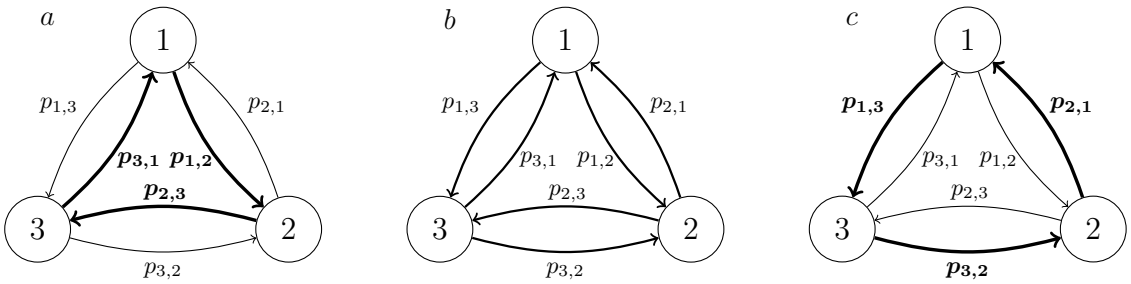


Figure 6: Graphical representation of the Nash equilibria

5.2 Simulation - Deterministic

In a "3-player network" deterministic buck-passing game, the only possible Nash Equilibria are the ones obtained in pure strategy (eq. 5.2). The simulation examined the strategies developed by the players after interacting for 10000 periods testing six possible values of α (1, 0.5, 0.3, 0.2, 0.1, 0.05) and the arithmetic average method.

In order to have a clearer view of the impact on the social efficiency of the different averaging methods, the "Tempo" parameter has been added. It measures the time required to reach a Nash Equilibrium. The data obtained by the simulation are the following:

α	Pure NE	Non-NE	Tempo
1	100%	0%	9.2
0.5	100%	0%	21
0.3	100%	0%	60.4
0.2	78%	22%	96
0.1	81%	19%	260
0.05	80%	20%	739
Ar. Avg	59%	41%	7.7

Table 2: Simulation results. Ratios calculated over a sample of 100 observations per each α. **Ar. Avg**: Arithmetic average. The column "Tempo" reports the average number of periods required by the players before reaching a Nash Equilibrium (the average is calculated among the observations in which a Nash Equilibrium is reached. For example: for $\alpha = 0.2$, Tempo= 96 is the average of the number of periods required to reach a NE only in the 78% of the observations, the ones in which a NE is actually reached

In every simulation, there are two possible outcomes: the strategy profile is a Nash equilibrium or not. The experiment shows that for a value of α greater than or equal to 0.3 the players find the NE $\approx 100\%$ of the times. When $\alpha \leq 0.2$, players find the NE only in $\approx 80\%$ of the cases. This value does not seem to be affected by α (as long as $\alpha \leq 0.2$). In fact, the values obtained when α is equal to 0.2, 0.1, and 0.05 are very close. In the case in which players use the arithmetic average model, the ratio of Nash equilibria is the lowest (59%).

Concerning the "Tempo" parameter, the experiment shows that it is inversely proportional to the value of α (higher the α, lower the "Tempo" and vice versa). However, the lowest measure of "Tempo" is obtained with the arithmetic average method. In this case, new observations strongly affect players' forecasts at the beginning and tend to be meaningless as the game proceeds. This allows the players

to strongly define their strategies but only for a short amount of time, allowing only the early-born Nash equilibria to exist, lowering the "Tempo".

What method (moving or arithmetic average) leads to the most socially efficient outcome? In order to answer this question, the social costs must be calculated. For a strategy profile $\boldsymbol{s}^{PNE}$, which is a pure strategy Nash equilibrium (Hamiltonian cycle), the social cost is:

$$\mathrm{SC}(\boldsymbol{s}^{PNE}) = \frac{1}{3}. \tag{5.6}$$

The social cost obtained when a Hamiltonian cycle occurs is the lowest possible social cost for a 3-player buck-passing game. In fact, if the sum of personal costs (c_i) must add up to 1, the lowest possible $\mathrm{SC}(\boldsymbol{s})$ such that $\mathrm{SC}(\boldsymbol{s}) = \max_{i \in V} c_i(\boldsymbol{s})$ is $\frac{1}{3}$. In the light of this and of the fact that there are no other pure strategies that lead to an even distribution of the costs, it is clear that, for a strategy profile $\boldsymbol{s}^{\not{NE}}$ which is not a pure strategy Nash equilibrium, the social cost is:

$$\mathrm{SC}(\boldsymbol{s}^{\not{NE}}) > \frac{1}{3}. \tag{5.7}$$

From this analysis, it is deducible that a strategy profile which is a pure strategy Nash equilibrium is more socially efficient than one that is not. Thus, the most efficient outcomes are obtained by choosing the moving average method with $\alpha \geq 0.3$.

A further discrimination can be done by considering the "Tempo" parameter. Given that a Hamiltonian cycle is the most efficient situation, the quicker it is reached the better. Thus, between 0.3, 0.5, and 1, $\alpha = 1$ allows a larger amount of periods passed at the most socially efficient stage. Even if the arithmetic average method allows the players to obtain a lower "Tempo" value, it leads to a non-NE in 41% of the simulations and therefore cannot be considered.

In a "3-player network" deterministic buck-passing game, the players obtain the best (socially efficient) result by basing their decisions only on their last experience, easily forgetting the past ($\alpha = 1$). The worst outcomes are obtained by applying a value of α below 0.2 or by adopting an arithmetic average scheme.

5.3 Simulation - Stochastic

In a "3-player network", every player, in turn, faces a choice: passing the buck to the left or to the right. Under a stochastic buck-passing game perspective, the players randomise between these two options. Given that, the sum of each player probabilities must be equal to one, the agents assign probabilities to their options as shown in the following figure.

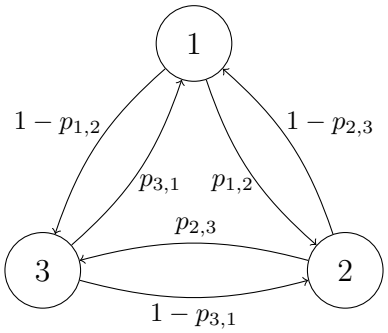
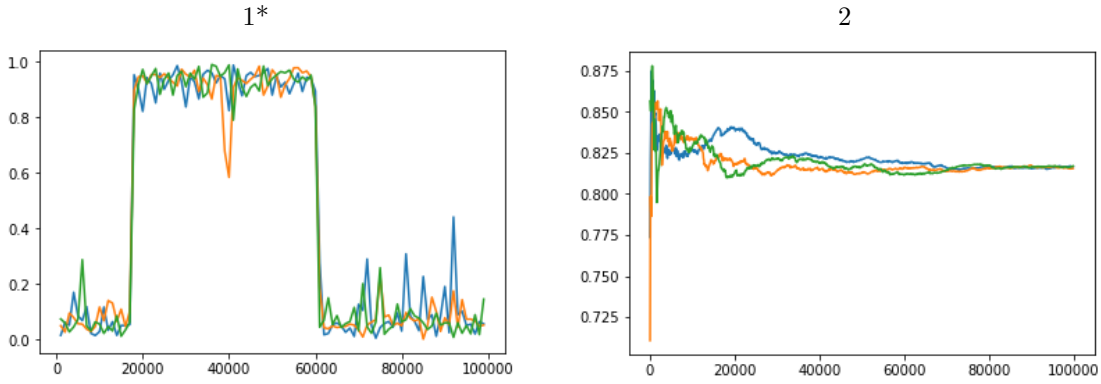


Figure 7: The 3-player Network with probabilities.

After the first simulations, it has been noticed that the probabilities assigned by the players tend to develop simultaneously and in the same way for all players. More precisely, the probabilities $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ (clockwise) tend to be all equal to each other. In the following figure, the graphs of the probabilities trending of two different simulations show the phenomenon.



Plot 1 and Plot 2: Two plots showing the development of $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ over 100000 periods. X-axis: number of periods. Y-axis: probability. Blue line = $p_{1,2}$. Orange line = $p_{2,3}$. Green line = $p_{3,1}$. Simulation parameters plot 1: $\alpha = 0.2$, $\beta = 3$ and Network: 3-player network. Simulation parameters plot 2: Ar. Avg, $\beta = 2$ and Network: 3-player network. *: **further analysis on stable states switching in section 5.3.2**

Given that this phenomenon occurs in every simulation, the variable p is introduced. p is the value that $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ are approaching. When the simulation runs for thousands of periods and $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ reach a stable state, it can be assumed that $p = p_{1,2} = p_{2,3} = p_{3,1}$.

The introduction of p reduces the number of variables, allowing us to express conditional expectations of the players in terms of p only. Let

$$\begin{aligned}\mathscr{T}_{1,P_1 \Rightarrow P_2} &= \mathbb{E}(\text{Return time for player 1} \mid P_1 \Rightarrow P_2) \\ &= 1 + \lambda_{2,1},\end{aligned}$$

in which $\lambda_{2,1}$ is the expected time taken by the buck to go from player 2 to player 1.

Moreover, $\lambda_{2,1}$ and $\lambda_{3,1}$ can be defined as:

$$\begin{cases} \lambda_{2,1} = (1-p)(1) + p(1+\lambda_{3,1}) = 1 + p\lambda_{3,1} \\ \\ \lambda_{3,1} = p(1) + (1-p)(1+\lambda_{2,1}) = 1 + (1-p)\lambda_{2,1}, \end{cases}$$

and thus:

$$\lambda_{2,1} = \frac{1+p}{1-p(1-p)} \tag{5.8}$$

The conditional expected time is then calculated:

$$\mathscr{T}_{1,P_1 \Rightarrow P_2} = 1 + \frac{1+p}{1-p(1-p)} = \frac{2+p^2}{1-p+p^2}\,, \tag{5.9}$$

and, by symmetry:

$$\mathscr{T}_{1,P_1 \Rightarrow P_3} = \frac{2+(1-p)^2}{1-(1-p)+(1-p)^2}\,. \tag{5.10}$$

The two curves $\mathscr{T}_{1,P_1 \Rightarrow P_2}$ and $\mathscr{T}_{1,P_1 \Rightarrow P_3}$ are plotted in the following graph.

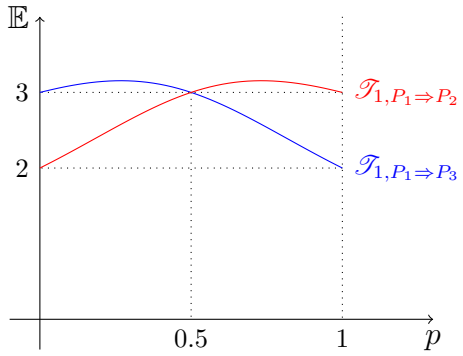


Figure 8: Graphical representation of $\mathscr{T}_{1,P_1 \Rightarrow P_2}$ and $\mathscr{T}_{1,P_1 \Rightarrow P_3}$

The graph shows that, assuming that the value of p is identical for every player, the best response of Player 1 can be described as:

$$\mathrm{BR}_1 = \begin{cases} \Rightarrow P_2 & \text{for } p > 0.5 \quad (\ \mathscr{T}_{1,P_1 \Rightarrow P_2} > \mathscr{T}_{1,P_1 \Rightarrow P_3}\ ) \\ \text{Indifferent} & \text{for } p = 0.5 \quad (\ \mathscr{T}_{1,P_1 \Rightarrow P_2} = \mathscr{T}_{1,P_1 \Rightarrow P_3}\ ) \\ \Rightarrow P_3 & \text{for } p < 0.5 \quad (\ \mathscr{T}_{1,P_1 \Rightarrow P_2} < \mathscr{T}_{1,P_1 \Rightarrow P_3}\ ) \end{cases}$$

This explains the fact that players tend to point in the same direction, moving together towards a clockwise or counter-clockwise cycle. Moreover, the fact that probabilities tend to be symmetrically equal to each other can be explained by best response combined with the way in which these probabilities are decided.

5.3.1 Beta (β) and Equilibria

As shown in section 3, players update their strategies regularly according to the following formula:

$$p_{i,j} = \begin{cases} \frac{e^{\beta\omega_{i,j}}}{\sum_{k\in\mathscr{N}_i^+} e^{\beta\omega_{i,k}}} & \text{if} \quad j \in \mathscr{N}_i^+ \\ \\ 0 & \text{if} \quad j \notin \mathscr{N}_i^+ \end{cases} . \tag{5.11}$$

The term $\omega_{i,k}$ represents player i forecasts about $\mathscr{T}_{i,P_i\Rightarrow P_k}$. Thus, assuming that all players forecasts are correct, equation (5.10) for $p_{1,2}$ can be written as:

$$p_{1,2} = \frac{e^{\beta\mathscr{T}_{1,P_1\Rightarrow P_2}}}{e^{\beta\mathscr{T}_{1,P_1\Rightarrow P_2}} + e^{\beta\mathscr{T}_{1,P_1\Rightarrow P_3}}} .$$

Moreover, by implementing equations (5.8) and (5.9), the following formula is obtained:

$$p_{1,2} = \frac{e^{\beta\left(\frac{2+p^2}{1-p+p^2}\right)}}{e^{\beta\left(\frac{2+p^2}{1-p+p^2}\right)} + e^{\beta\left(\frac{2+(1-p)^2}{1-(1-p)+(1-p)^2}\right)}} . \tag{5.12}$$

At the beginning of the section, it has been shown that in the long run $p_{1,2}$ approaches p. Therefore, at any stable stage, $p_{1,2}$ must be equal to p. The following graph is obtained by plotting the solution to equation (5.12) for $p_{1,2} = p$.

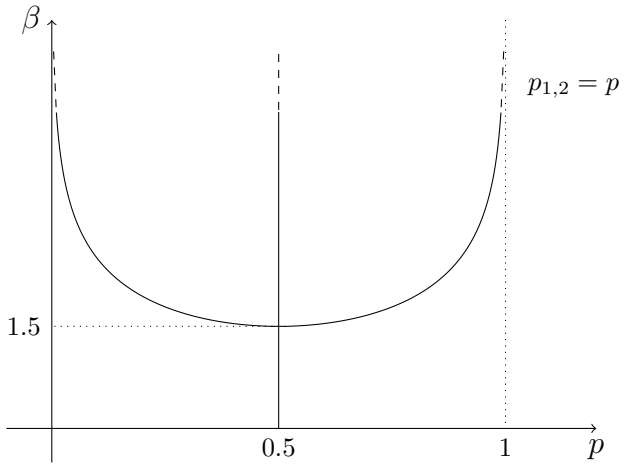
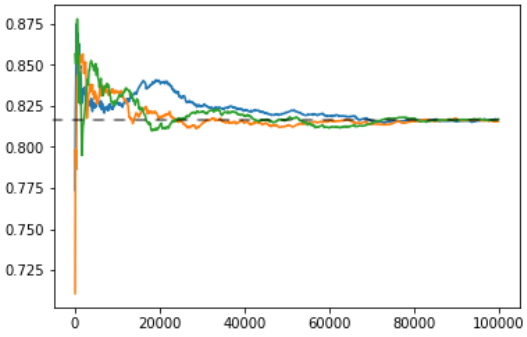


Figure 9: Graph representing the solution to equation (5.12) for $p_{1,2} = p$. Relationship between p and β

The graph suggests that $p = 0.5$ is **always** a stable stage for any value of β. This can be explained by the fact that: whenever players' forecasts are all equal, their strategy will be $p = 0.5$ no matter the value of β. Moreover, if players' strategies follow $p = 0.5$, their forecasts should remain equal, maintaining the stable state.

Whenever players are able to make accurate forecasts, their mixed strategy can be predicted by knowing the β. For example, Plot 2 (at the beginning of section 5.3) represents data from a simulation is which $\beta = 2$ and shows $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ all approaching a common value between 0.800 and 0.825.



Plot 2: X-axis: number of periods. Y-axis: probability. Blue line = $p_{1,2}$. Orange line = $p_{2,3}$. Green line = $p_{3,1}$. Simulation parameters plot 2: Ar. Avg, $\beta = 2$ and Network: 3-player network.

By checking the possible values of p for $\beta = 2$ in Figure 9, three possible results are obtained: $p = 0.1847$, $p = 0.5$, and $p = 0.8153$.

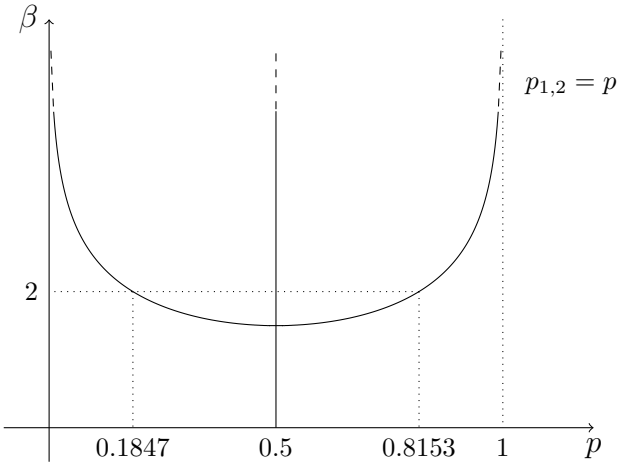
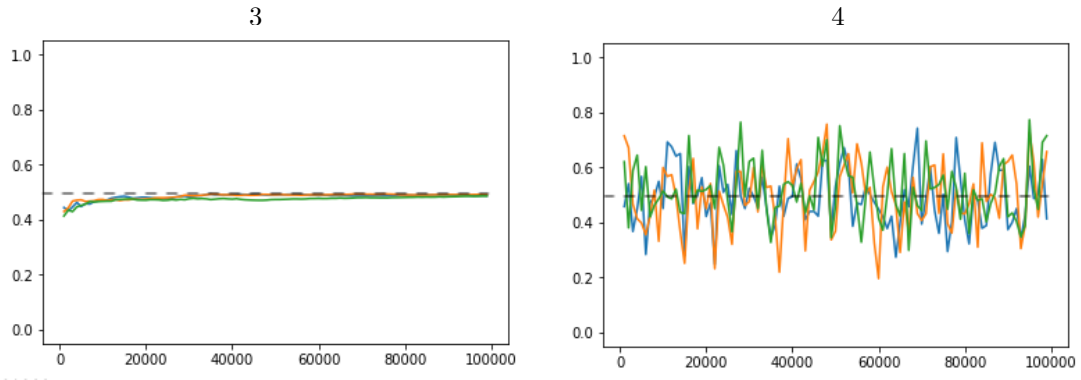


Figure 10: Relationship between p and β. $\beta = 2$

The result shown by Plot 2 is coherent with the value of p given by formula 5.12 and Figure 10. $p = 0.8153$ is very likely the value on which $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ are stabilising in Plot 2.

It can be noticed that the value of β not only affects the values of the stable state's probabilities but also the number of possible stable states. Figure 9 shows that for $\beta \leq 1.5$ there is only one stable state, while for $\beta > 1.5$ the possible states are 3. Moreover, the graph shows that, when β grows, p approaches 0 or 1, and when β falls, p approaches 0.5.

This explains the reason why, for a low β, there is only one stable state. When β is small enough, players are not able to assign very low or very high probabilities to strategies (equation 5.10). Moreover, if each player does not implement a strong (and thus predictable) passing strategy, expectations will be tighter inhibiting players even more to assign low or high probabilities. In the 3-player network, when $\beta \leq 1.5$ players are forced to the Nash Equilibrium NE = $[(\frac{1}{2}; \frac{1}{2}), (\frac{1}{2}; \frac{1}{2}), (\frac{1}{2}; \frac{1}{2})]$. The following graphs confirm these results. They show the evolution over time of $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ in a 3-player network with $\beta = 1$.

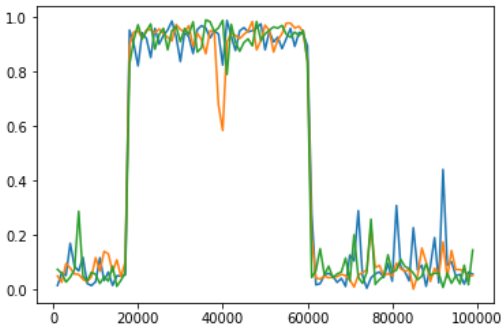


Plot 3 and Plot 4: Two plots showing the development of $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ over 100000 periods. X-axis: number of periods. Y-axis: probability. Blue line = $p_{1,2}$. Orange line = $p_{2,3}$. Green line = $p_{3,1}$. Simulation parameters plot 3: Ar. Avg, $\beta = 1$ and Network: 3-player network. Simulation parameters plot 4: $\alpha = 0.1$, $\beta = 1$ and Network: 3-player network.

In both plots, the values of $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ lie around 0.5. However, in Plot 3 the probabilities seem to converge to 0.5, while in Plot 4, they fluctuate above and below 0.5. These different behaviours are explained by the different averaging methods used in the two experiments. In Plot 3, the arithmetic average method is used. As time passes, the relevance of a singular new observation decreases, allowing $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ to be very stable. In Plot 4, it is used a moving average with $\alpha = 0.1$. In this case, players' forecasts can always be substantially affected by new observations, forcing $p_{1,2}$, $p_{2,3}$, and $p_{3,1}$ to fluctuate around the stability value.

5.3.2 Stable states switching

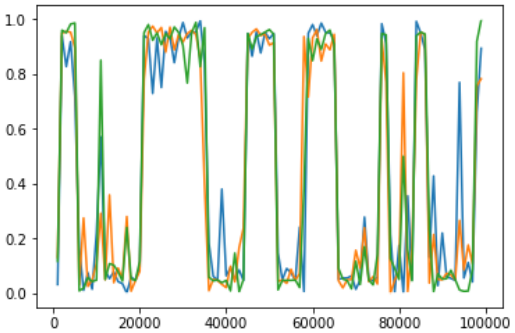
In the previous section it has been shown that, in the case of moving average method, players' strategies do not stabilise on p, but fluctuate around it. If the value of α is high, these fluctuations may be large enough to allow the whole system to switch the stable state.



Plot 1: X-axis: number of periods. Y-axis: probability. Blue line = $p_{1,2}$. Orange line = $p_{2,3}$. Green line = $p_{3,1}$. Simulation parameters plot 1: $\alpha = 0.2$, $\beta = 3$ and Network: 3-player network.

Plot 1 is a great example of this phenomenon. During the simulation, players stabilise on a counterclockwise Hamiltonian cycle at the beginning. At $t \approx 20000$ the state changed to a clockwise Hamiltonian cycle and, at $t \approx 60000$, it switched back to the initial state. How do α and β affect stability?

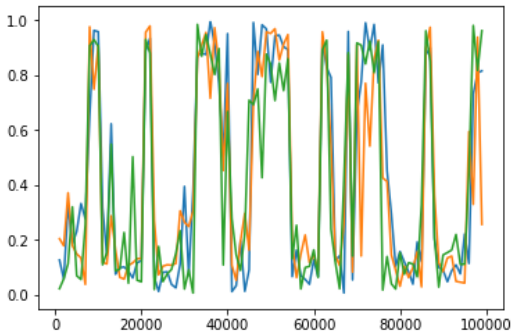
The following plot represents the results of a simulation ran with the same β of plot 1 but with a higher α.



Plot 1A: X-axis: number of periods. Y-axis: probability. Blue line = $p_{1,2}$. Orange line = $p_{2,3}$. Green line = $p_{3,1}$. Simulation parameters plot 1A: $\alpha = 0.3$, $\beta = 3$ and Network: 3-player network.

Plot 1A suggests that, *ceteris paribus*, for a greater value α the stability decreases. This result can be explained by the fact that a greater α leads to sharper fluctuations (section 5.3.1) and thus lower stability.

Concerning β, the following plot represents the results of a simulation ran with the same α of plot 1 but with a lower β.



Plot 1B: X-axis: number of periods. Y-axis: probability. Blue line = $p_{1,2}$. Orange line = $p_{2,3}$. Green line = $p_{3,1}$. Simulation parameters plot 1B: $\alpha = 0.2$, $\beta = 2.5$ and Network: 3-player network.

Plot 1B suggests that, *ceteris paribus*, for a lower value of β the stability decreases. This observation can explained by the fact that a lower β, as shown in figure 9, makes the stable states closer to each other, allowing smaller fluctuations to cause a state switch. Therefore, if β decreases and α stays constant, the stability decreases.

6 Cycle Network

In this network, players are bilaterally connected only to two others. The network is arranged in a ring-shaped structure as shown in Figure 11.

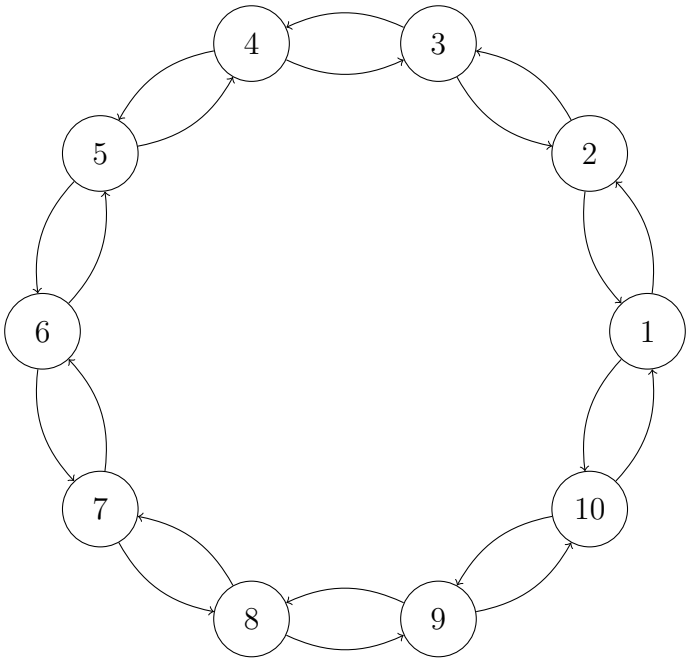


Figure 11: Graphical representation of a cycle graph with 10 players

The adjacency matrix of a 10 players cycle network is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{6.1}$$

6.1 Nash Equilibria

In a cycle network, players face only two pure passing strategies: "passing the buck to the left" and "passing the buck to the right" (Note: the 3-player network is a cycle network composed of only 3 players).
The cycle network allows only two Pure Nash Equilibrium situations: a Hamiltonian cycle (Figure 12,a) or a 2 players loop (Figure 12,b).

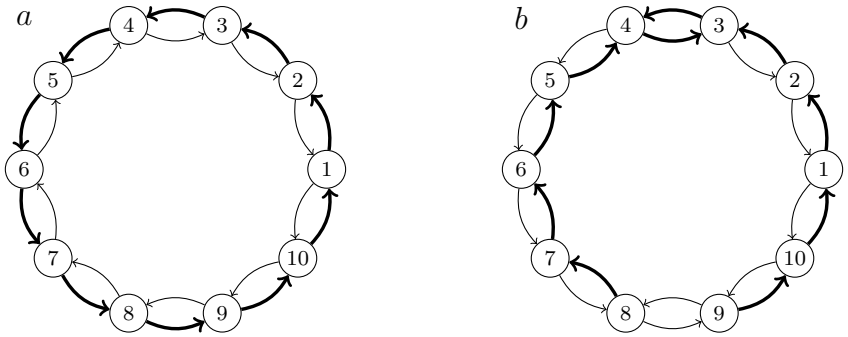


Figure 12: Graphical representation of two possible NE in a cycle graph with 10 players.
a: Hamiltonian Cycle. b: two-player loop

Concerning Hamiltonian cycles, there are 2 Nash Equilibria: clockwise and counter-clockwise. Regarding the two-player loop, there are many possible Nash Equilibria combinations. In order for a two-player loop situation to happen, at least four players are necessary. For example, let us consider the following situation:

Figure 13: Frame of a Cycle network representing a two-player loop situation.

in which personal costs are:

$$c_1 = 0, \;\; c_2 = 0.5, \;\; c_3 = 0.5, \;\; c_4 = 0\,. \tag{6.2}$$

Players 1 and 4 obtain the lowest possible personal costs and thus they have no incentive in changing their strategies. Players 2 and 3 share all the costs, obtaining each half of the total costs. However, they cannot obtain lower personal costs by

changing strategy, as they are "locked-in" by players 1 and 4. If player 2 decides to pass the buck to player 1 (instead of player 3), c_2 would still be equal to 0.5, making this change unprofitable. Since the same reasoning applies to player 3, no player has an incentive to change strategy and thus the two-player loop is a Nash Equilibrium.

6.1.1 Fairness of equilibria

A Hamiltonian cycle is always the most efficient equilibrium. For any cycle network buck-passing game composed of n players, let $\boldsymbol{hc}$ be a strategy profile that leads to a Hamiltonian cycle. For $i \in T$, the personal cost function of i is defined as:

$$c_i(\boldsymbol{hc}) = \frac{1}{n}, \tag{6.3}$$

and thus, the social cost function is defined as:

$$\mathrm{SC}(\boldsymbol{hc}) = \max_{i \in V} c_i(\boldsymbol{hc}) = \frac{1}{n}\,. \tag{6.4}$$

In a two-player loop Nash Equilibria, all the costs are shared by the two players in the loop. For any cycle network buck-passing game composed of n players, let $\boldsymbol{l}$ be a strategy profile which leads to a two-player loop Nash Equilibria. The social cost function is defined as:

$$\mathrm{SC}(\boldsymbol{l}) = \max_{i \in V} c_i(\boldsymbol{l}) = \frac{1}{2}\,. \tag{6.5}$$

This analysis shows that the most efficient equilibrium is the Hamiltonian cycle one. On the other hand, a two-player loop is one of the most inefficient possible situations.

6.2 Simulation - Deterministic

The simulation examined the strategies developed by n players after interacting for 10000 periods testing six possible values of α (1, 0.5, 0.3, 0.2, 0.1, 0.05) and the arithmetic average method for three different values of n (5, 10, 20). The results are reported in the following tables.

Table 3: 5 Players cycle

α	Hamiltonian Cycle	2 players loop	Non-NE
1	100%	0%	0%
0.5	52%	48%	0%
0.3	66%	34%	0%
0.2	63%	37%	0%
0.1	56%	44%	0%
0.05	65%	35%	0%
Ar. Avg	47%	53%	0%

Table 4: 10 Players cycle

α	Hamiltonian Cycle	2 players loop	Non-NE
1	100%	0%	0%
0.5	18%	82%	0%
0.3	12%	88%	0%
0.2	17%	83%	0%
0.1	12%	88%	0%
0.05	17%	83%	0%
Ar. Avg	14%	86%	0%

Table 5: 20 Players cycle

α	Hamiltonian Cycle	2 players loop	Non-NE
1	100%	0%	0%
0.5	0%	100%	0%
0.3	0%	100%	0%
0.2	0%	100%	0%
0.1	0%	100%	0%
0.05	0%	100%	0%
Ar. Avg	0%	100%	0%

Tables 3-4-5: Simulation results. Ratios calculated over a sample of 100 observations per each α and arithmetic average, per each n. Every simulation has been run for 10000 periods.

The data obtained in the simulation disclose many facts and aspects regarding the relationships between parameters, equilibria and players' behaviour:

1. **Always Nash Equilibria**

   The frequency of cases in which players do not reach a Nash Equilibria is approximately 0%. In none of the 2100 (100 per each α and Ar. Avg. [7] per each n [3]) simulations run the players reached a non-Nash Equilibria state. This observation can be explained by the structure of the network. It is very likely for the players to end up in a Hamiltonian cycle or to get stacked in a 2 players loop. There is no other significant stable situation.

2. **Always Hamiltonian cycle when** $\alpha = 1$

   The frequency of cases in which players reach a Hamiltonian cycle Nash Equilibria is approximately 100%. In all the 300 (100 per each n [3]) simulations in which $\alpha = 1$, the players reached a Hamiltonian cycle. This observation can be explained by the fact that, once the players enter in a Hamiltonian cycle, they never leave it. When $\alpha = 1$ players rely only on their previous observation. This allows them to quickly understand to be locked in a two-player loop and force their way out by changing strategy. A single two-player loop is often not sustainable for more than a dozen periods.

3. **Positive relationship between** n **and two-player loop frequency**

   For any $\alpha \neq 1$, the frequency of cases in which players reach a two-player loop increases as n increases. For $n = 5$ the frequency of two-player loop is $\approx 40\%$, for $n = 10$ is $\approx 85\%$, and for $n = 20$ is $\approx 100\%$. This fact can be easily explained by the fact that a bigger network requires a larger number of players to "agree" on the same Hamiltonian cycle. More the players, the higher the probability to get locked in a two-player loop.

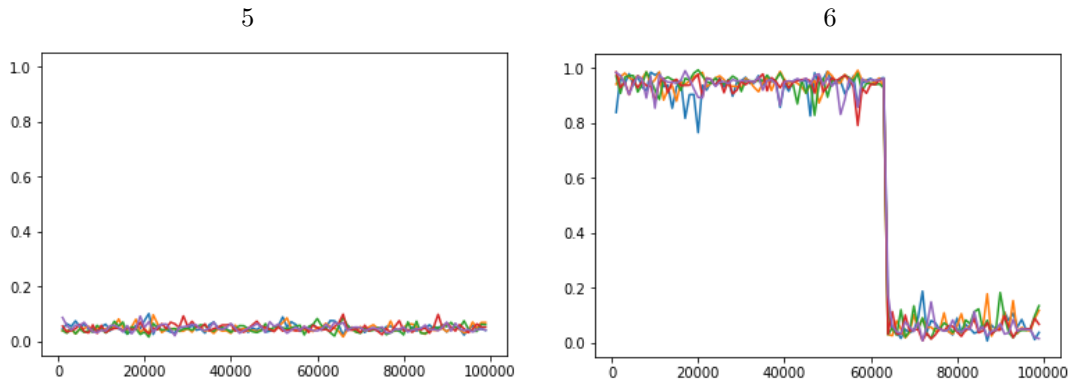
6.3 Simulation - Stochastic

All the data analysed in this section have been obtained from simulations on the 5-player cycle network.

The data obtained in the stochastic simulation bring to light the following aspects about players' behaviour:

1. **Probabilities tend to move together**

   As it has been observed in the 3-player network, in bigger cycle networks probabilities chosen by players tend to be equal to one another following a clockwise or counterclockwise Hamiltonian cycle.



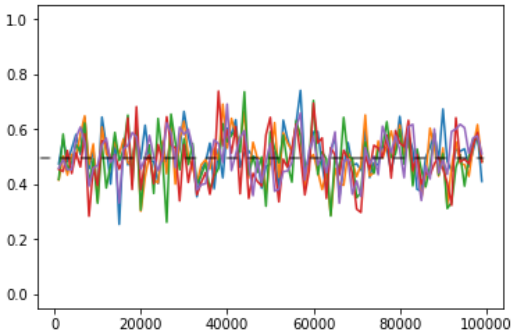
Plot 5 and Plot 6: Two plots showing the development of $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ over 100000 periods. X-axis: number of periods. Y-axis: probability. Blue line $= p_{1,2}$. Orange line $= p_{2,3}$. Green line $= p_{3,4}$. Red line $= p_{4,5}$. Purple line $= p_{5,1}$. Simulation parameters plot 5: $\alpha = 0.1$, $\beta = 1$ and Network: 5-player cycle network. Simulation parameters plot 6: $\alpha = 0.3$, $\beta = 1$ and Network: 5-player cycle network.

Plots 5 and 6 show that, in a 5-player cycle network, $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ tend to be equal to each other. Moreover, plot 6 shows that whenever there is a switch of Hamiltonian cycle (from clockwise to counterclockwise) $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ quickly move together maintaining their bond.

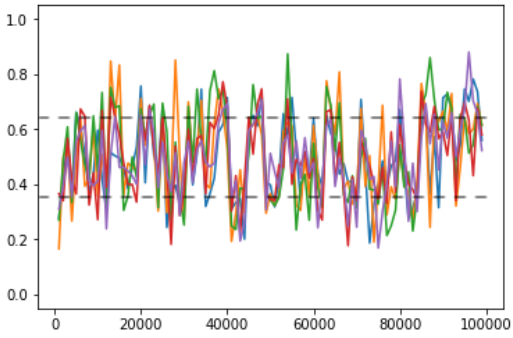
2. β **defines the number of possible stable states**

As it has been observed in the 3-player network, in the 5-player cycle network the value of β defines the number of possible stable states.

7



8



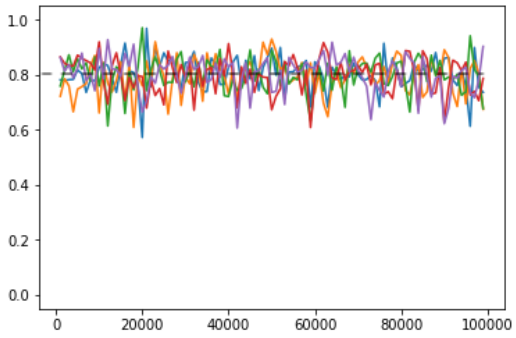
Plot 7 and Plot 8: Two plots showing the development of $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ over 100000 periods. X-axis: number of periods. Y-axis: probability. Blue line $= p_{1,2}$. Orange line $= p_{2,3}$. Green line $= p_{3,4}$. Red line $= p_{4,5}$. Purple line $= p_{5,1}$. Simulation parameters plot 7: $\alpha = 0.1$, $\beta = 0.2$ and Network: 5-player cycle network. Simulation parameters plot 8: $\alpha = 0.1$, $\beta = 0.3$ and Network: 5-player cycle network.

Plot 7 shows that $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ fluctuate around $p = 0.5$. This suggests that for $\beta = 0.2$ there is only one possible stable state: $p_{1,2} = p_{2,3} = p_{3,4} = p_{4,5} = p_{5,1} = 0.5$. Plot 8 shows that $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ fluctuate around two values of p, often switching between the two. It is reasonable to believe that in every cycle network there is a definable relationship between β and the number of stable states as shown for the 3-player network in Figure 9. This reasoning suggests that for a low value of β there is only one stable state, while for a higher value of β the stable states become three.

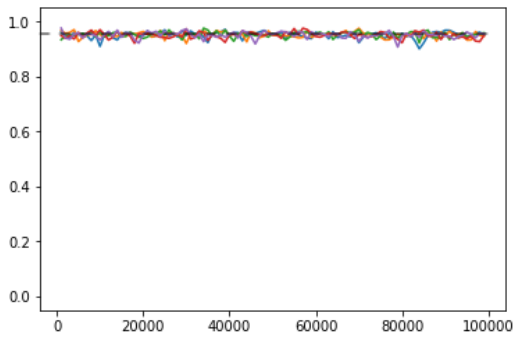
3. **β defines the probability set by players at any stable state**

The value of β seems not only controls the number of possible stable states but also defines them. In section 6.1, it has been stated that two of the pure strategy Nash equilibria are the clockwise and the counterclockwise Hamiltonian cycles. In other words: $p_{1,2} = p_{2,3} = p_{3,4} = p_{4,5} = p_{5,1} = 1$ and $p_{1,2} = p_{2,3} = p_{3,4} = p_{4,5} = p_{5,1} = 0$. However, these values cannot be obtained by this simulation because of the structure of the decision-making process (Formula 3.13). Plots 9 and 10 show that the probabilities selected by the players stabilise on values different from 0 and 1. Probably these values are the maximum possible values allowed by formula 3.13.

9



10



Plot 9 and Plot 10: Two plots showing the development of $p_{1,2}$, $p_{2,3}$, $p_{3,4}$, $p_{4,5}$, and $p_{5,1}$ over 100000 periods. X-axis: number of periods. Y-axis: probability. Blue line $= p_{1,2}$. Orange line $= p_{2,3}$. Green line $= p_{3,4}$. Red line $= p_{4,5}$. Purple line $= p_{5,1}$. Simulation parameters plot 7: $\alpha = 0.1$, $\beta = 0.5$ and Network: 5-player cycle network. Simulation parameters plot 8: $\alpha = 0.1$, $\beta = 1$ and Network: 5-player cycle network.

Plot 9 shows results for $\beta = 0.5$, while Plot 10 shows results for $\beta = 1$. The fact that the stabilising value of Plot 10 is greater than the one of Plot 9 is coherent with formula 3.13. In fact, a higher β allows players to assign a higher probability to the strategy they believe to be the best.

7 Complete Network

Contrary to the cycle graph, which allows only two connections per player, the complete graph permits every possible connection. Each player is connected to any other. This type of network will be analysed only under a deterministic viewpoint.

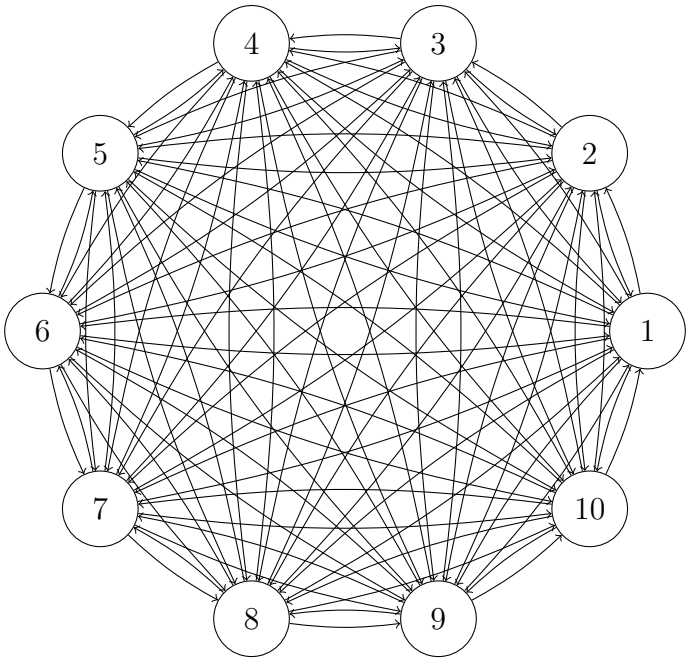


Figure 14: Graphical representation of a complete graph with 10 players

The adjacency matrix of a 10 players complete network is:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}. \tag{7.1}$$

7.1 Nash Equilibria

Because of its structure, every complete graph network allows only one type of pure strategy Nash Equilibria: a Hamiltonian cycle. Every sub-cycle (which does not include every player) can never be considered a Nash Equilibrium in pure strategy.

Proof:

There are two possible scenarios:

1. More sub-cycles:

   In the case in which there are two or more closed sub-cycles, all the players inside of those cycles are adopting strategies that do not allow the buck to get out of the cycle once it gets in. Whenever the buck gets stacked in one cycle, every component of the loop would be better off by passing the buck to any of the players outside of the cycle, lowering personal costs to 0. Therefore, every strategy profile which includes two ore more sub-cycles cannot be considered a Nash Equilibrium.

   Example 7.1

   Let us consider a complete network composed of six players. The strategies chosen by the players lead to 2 sub-cycles made of three players each (Figure 15a). If the buck gets stacked in the sub-cycle composed of players 3, 4 and 5, each of these players will be better off by changing strategy and pass the buck to the other sub-cycle. In the example of Figure 15b, player 5 changes strategy and passes the buck to player 6.

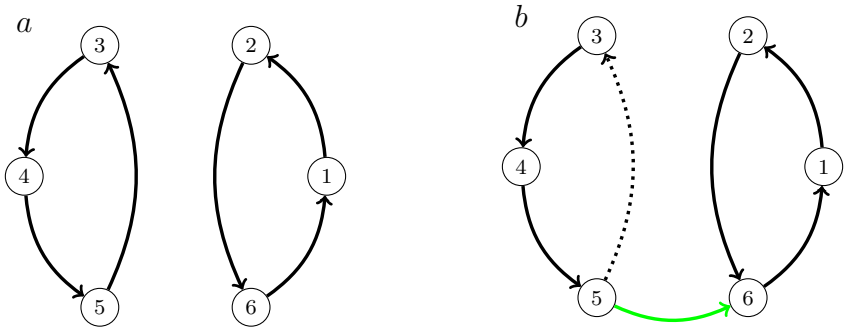


Figure 15: Example of a complete network with two sub-cycles. 15a: Initial situation. 15b: final situation. The green arrow represents one of the possible strategies that will make player 5 better off.

2. Only one sub-cycle

   Let us consider a network composed of n players and a strategy profile $\boldsymbol{s}$ in which $n-k$ of the players pass the buck to each other in a cycle. Given that there is only one sub-cycle, for any k different from 0 there would always be at least 1 player outside this cycle. This player has his own strategy which consists of a "chosen neighbour": j. The personal costs of each player in the cycle are $\frac{1}{n-k}$, while the personal costs of each of the k excluded players are 0. In this situation, there must be at least two players who selected j as "chosen neighbour": one in the cycle and one outside. $\boldsymbol{s}$ may never be a Nash Equilibrium because the player in the cycle who selected j as "chosen neighbour" would be better off by passing the buck to the one outside the cycle which will eventually pass it to j, making the cycle bigger by one unit and lowering personal costs to $\frac{1}{n-k+1}$.

Example 7.2

Let us consider a complete network composed of six players. The strategies chosen by the players lead to one sub-cycle made of four players: 1, 2, 3 and 6. players 4 and 5 are not included in the cycle (Figure 16).

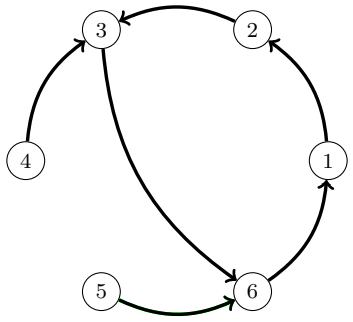


Figure 16: Example of a complete network with one sub-cycle composed of 4 players and 2 players outside of the cycle. $n = 6$, $k = 2$

In this example, two players may change strategy and obtain lower costs. Player 2 should pass the buck to player 4 instead of player 3 (Figure 17a). In this way, the cycle gets bigger and the personal costs are reduced. Following the same reasoning, Player 3 should pass the buck to player 5 instead of player 6 (Figure 17b), reducing costs.

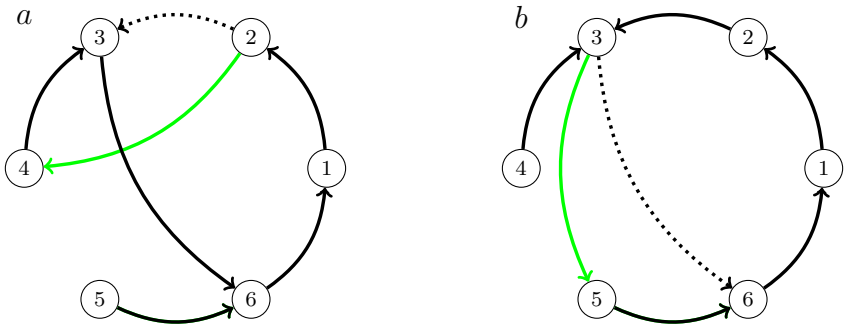


Figure 17: Example of a complete network with one sub-cycle. 17a: Player 2 better strategy. 17b: Player 3 better strategy.

7.2 Simulation - Deterministic

The simulation examined the strategies developed by n players after interacting for 10000 periods testing seven possible values of α (1, 0.9, 0.5, 0.3, 0.2, 0.1, 0.05) and the arithmetic average method for three different values of n (5, 10, 20). The results reported in the following tables represent the relationship between the averaging method and the number of players involved in the passing cycle.

Table 6: 5 Players complete network

α	3	4	5	Avg.
1	16%	79%	5%	3.89
0.9	24%	59%	17%	3.93
0.5	11%	64%	25%	4.14
0.3	9%	64%	27%	4.18
0.2	14%	62%	24%	4.10
0.1	18%	50%	32%	4.14
0.05	13%	55%	32%	4.19
Ar. Avg	12%	53%	35%	4.23

Table 7: 10 Players complete network

α	3-4	5	6	7	8	9-10	Avg.
1	16%	32%	25%	18%	8%	1%	5.73
0.9	8%	34%	34%	20%	2%	2%	5.80
0.5	12%	26%	24%	20%	28%	1%	6.10
0.3	7%	17%	25%	31%	15%	5%	6.47
0.2	10%	10%	20%	35%	19%	6%	6.60
0.1	5%	19%	20%	25%	23%	8%	6.64
0.05	10%	15%	22%	22%	24%	7%	6.56
Ar. Avg	2%	11%	18%	32%	23%	14%	7.06

Table 8: 20 Players complete network

α	3-7	8-9	10-11	12-13	14-15	16-20	Avg.
1	14%	40%	33%	13%	0%	0%	9.44
0.9	9%	41%	39%	11%	0%	0%	9.61
0.5	5%	16%	45%	27%	6%	1%	10.84
0.3	2%	11%	35%	37%	15%	0%	11.52
0.2	0%	8%	34%	41%	16%	1%	11.82
0.1	0%	1%	20%	46%	28%	5%	12.75
0.05	0%	1%	19%	39%	38%	3%	12.93
Ar. Avg	1%	3%	20%	36%	30%	10%	12.98

Tables 6-7-8 (previous page): Simulation results. Ratios calculated over a sample of 100 observations per each α and arithmetic average, per each n. Every simulation has been run for 10000 periods.

The data obtained in the simulation disclose many facts and aspects regarding players' behaviour:

1. **Nash equilibria are rare**

   The frequency of cases in which players reach a Nash equilibrium has a peak of 35% in the most favourable situation (5 players and arithmetic average method). However, in the 10-player complete network, the frequency falls to $\approx 1\%$ and in the 20-player complete network, a Nash equilibrium has been observed in none of the 800 (100 per each α and Ar. Avg [8]) simulations. This observation is explained by the structure of the network. Players have many more possible strategies in a complete network with respect to a cycle network. It is rare that all the players "agree" on the same Hamiltonian cycle. Smaller cycles are more likely

2. **More the players, fewer the Hamiltonian cycle**

   This point follows the same reasoning as in point 1. If the number of players in the network increases, each player faces more possible strategies, making even harder for the players to "agree" on the same Hamiltonian cycle.

3. α **is negatively correlated with the number of players involved in the sub-cycle**

   The data collected show a clear trend between α and the number of players involved in the sub-cycle. It can be noticed that, for the same value of n, the cycles tend to be big when α is small and vice versa. According to these observations, players need to strongly rely on their acquired knowledge and to avoid sudden fluctuations in their expectations.

4. **The arithmetic average method leads to the largest cycles**

   The data collected in the simulation show that, for the three values of n, the largest average number of players included in the cycle is obtained by

using the arithmetic average method. However, for $n = 5$ and $n = 20$, the average number of players in the cycle obtained with the arithmetic average is very close to the one obtained with $\alpha = 0.05$. Only for $n = 10$, the value is significantly larger. Thus, the arithmetic average approach may not be the cause behind this observation. The reason may be that, as after the first hundreds of periods new observations have little impact of forecasts, the arithmetic average method may be compared to a moving average with a low α. Lower the α, the bigger the cycle (point 3).

8 Conclusions

"The buck-passing game: simulation of adapting learning dynamics" has been a successful thesis. By the analysis of players' behaviours in a buck-passing game contest, numerous results have been obtained. Both, how players perceive reality (α) and how they take decisions (β), have an impact on the outcome. With the powerful support of computer programming, I have been able to run thousands of simulations, obtaining accurate output data. However, many questions remain unanswered. The buck-passing game is a wide and challenging topic. In this thesis, I decided to focus on only three networks and two models. Developing a similar analysis on more complex types of networks or codifying agents' interactions in different ways, would for sure bring interesting results.

The experiments started with the 3-player network. Among the three networks that have been analysed, this is not only the simplest but also represents a simplified version of the other two. In fact, the 3-player network is both, a complete network and a cycle network. The most interesting result has been obtained by the analysis of the stochastic buck-passing game played on a 3-player network. Figure 9 in section 5.3 shows that stable states are determined by the value of β. Taking this analysis as a point of reference, further researches may aim to find a general equation for cycle networks of any dimension, which would be a relevant achievement.

In this thesis, β has always been considered as a constant. In order to improve players' decisions, an evolution of β over time might be implemented into the model. Throughout the analysis of the stochastic buck-passing game, it has been observed that, for large values of β, players assign probabilities close to 0 or 1. Allowing β to increase over time can lead to higher stability and to achieve stable states which are Nash equilibria. Decreasing the value of α over time may lead to higher stability as well. However, it has been shown that, by adopting the arithmetic average method (very similar to a model in which α decreases over time), players do not always obtain socially efficient results.

The next step for analysing adapting learning dynamics in the buck-passing game consists in applying the model to different networks. Considering networks created through a preferential attachment system as the Barabási–Albert model may lead to different results. This model allows investigating networks much closer to reality than the ones examined in this paper. Observing players' behaviours and strategies' development in a real-life situation would allow studying the buck-passing game from a different perspective.

In this thesis, computer programming and machine learning are used as tools to explore a new game theory model: the buck-passing game. Throughout several

simulations, many data have been collected, mirroring a clear picture of the game. The obtained results brought to light relevant patterns on players' behaviour in a network game. To conclude, the buck-passing game is still an unexplored topic. Only by deeply focusing on every aspect we can exploit its full potential.

9 Glossary

Glossary arranged by sections. The terms are located in the section in which they first appeared.

Section 2.1:

- $\Psi = (\mathscr{G}, \mu)$: A deterministic buck-passing game.
- $\mathscr{G}(\mathscr{V}, \mathscr{E})$: A directed graph.
- $\mathscr{V}$: The set of vertices.
- $\mathscr{E}$: The set of edges.
- μ: A probability measure of $\mathscr{V}$.
- n: The number of players
- $\mathscr{N}_i^+$: The set of out-neighbours of player i.
- S_i: The set of strategies of Player i.
- S: The set of strategy profiles.
- $\Theta_{i,t}$: The random variable which defines the possession of the buck by player i at time t.
- $\boldsymbol{s}$: A strategy profile in the deterministic buck-passing game.
- c_i: The cost function of Player i.
- $\boldsymbol{c}$: The profile of cost functions.
- T: The number of periods.

Section 2.2:

- $\Upsilon = (\mathscr{G}, S, \mu)$: A stochastic buck-passing game.
- $\boldsymbol{\phi}_i$: The strategy vector of player i.
- $\phi_{i,j}$: $\mathbb{P}(\Theta_{j,t+1} = 1 | \Theta_{i,t} = 1)$.
- Φ: A strategy profile in the stochastic buck-passing game.

- $(X_i^{\Phi})_{t\geq 0}$: The Markov chain which represents the stochastic buck-passing game.

Section 2.3:

- NE: Nash Equilibrium.

Section 2.4:

- SC: Social Cost.
- PoA: Price of anarchy.
- PoS: Price of stability.

Section 3.2:

- Ω: The Forecast Matrix.
- $\omega_{i,j}$: The forecast of player i about the number of periods the "buck" will take to come back if passed to player j.
- $\boldsymbol{\omega}_i$: The forecast vector of Player i.
- α: The "Alpha coefficient". It represents the memory of the players.
- ι: Observation, information acquired.
- $h_{i,j}(t)$: The number of times that player i has passed the buck to player j at time t.

Section 3.3:

- P: The Probability matrix.
- $p_{i,j}$: The probability (calculated by player i) with which player i passes the buck to player j.
- $\boldsymbol{p}_i$: The probability vector of player i.
- β: The "Beta coefficient".

Section 4.1:

- A: The Adjacency matrix.

- $a_{i,j}$: Entry (i, j) of adjacency matrix A.

Section 5.1:

- $\Rightarrow P_i$: "Pass the buck to player i".

Section 5.2:

- Ar. Avg: Arithmetic average.

Section 5.3:

- $\lambda_{i,j}$: The expected time taken by the buck to go from player i to player j.
- $\mathscr{T}_{i,P_i \Rightarrow P_j}$: Conditional expectation. $\mathbb{E}$(Return time for player $i | P_i \Rightarrow P_j$).
- BR_i: Best response of player i.
- p: The value that the probabilities chosen by players tend to approach in a stochastic buck-passing game simulation.

10 References

1. Cominetti, Roberto; Quattropani, Matteo; and Scarsini, Marco "The Buck-Passing Game" Research Paper. Accessed May, 2019.

2. Mitford M. Mathews, ed., A Dictionary of Americanisms on Historical Principles (Chicago, University of Chicago Press, 1951), I, pp. 198–99.

3. "Markov Chain." *Dictionary by Merriam-Webster*, Merriam-Webster, 2019, www.merriam-webster.com/dictionary/Markov%20chain.

4. "Nash Equilibrium" *Oxford Reference*, www.oxfordreference.com, 2019, www.oxfordreference.com/view/10.1093/oi/authority.20110803100223327

5. "Intelligence" *Cambridge Dictionary*, Dictionary.cambridge.org, 2019, dictionary.cambridge.org/it/dizionario/inglese/intelligence.

6. See Appendix

7. "Adjacency Matrix" Oxford Reference, www.oxfordreference.com, 2019, www.oxfordreference.com/view/10.1093/oi/authority.20110803095351444

11 Appendix

11.1 Python codes: Cycle Network and 3-player Network

```
import numpy as np
import random as rm
import math
np.set_printoptions(suppress=True)
import matplotlib.pyplot as plt

def Cycle_graph_Activity(T, n, alpha, beta):

    players = list(range(0, n))

    PlayersCount = np.array([0]*n)

    PlayerPerPlayerCount = np.array([[0]*n]*n)

    LastTimePlayersPassage = np.array([0]*n)

    LastPlayersAction = np.array([-1]*n)

    ExperctedMatrix =np.array([[0.]*n]*n)
    for k in range(0,n):
        ExperctedMatrix.itemset((k, k), -1.)

    ExperctedMatrix =np.array([[-1.]*n]*n)
    for k in range(0,n):
        if k==0:
            ExperctedMatrix.itemset((k, k+1), 0)
            ExperctedMatrix.itemset((k, n-1), 0)
        elif k==(n-1):
            ExperctedMatrix.itemset((k, k-1), 0)
            ExperctedMatrix.itemset((k, 0), 0)
        else:
            ExperctedMatrix.itemset((k, k-1), 0)
            ExperctedMatrix.itemset((k, k+1), 0)


    ProbabilityMatrix =np.array([[0.]*n]*n)
    if beta == -1:
        for j in range(0, n):
            for x in range(0, n):
                if max(ExperctedMatrix[j]) == ExperctedMatrix[j][x]:
                    ProbabilityMatrix.itemset((j, x), 1)

    else:
        for j in range(0, n):
            if j==0:
                Max = max(ExperctedMatrix[j])
                Sum = math.exp((beta)*(ExperctedMatrix[j][j+1] - Max)) + math.exp((beta)*(
                    ExperctedMatrix[j][n-1] - Max))
                ProbabilityMatrix.itemset((j, j+1), (math.exp((beta)*(ExperctedMatrix[j][j+1]-
                    Max))/Sum))
                ProbabilityMatrix.itemset((j, n-1), (math.exp((beta)*(ExperctedMatrix[j][n-1]-
                    Max))/Sum))
            elif j==(n-1):
                Max = max(ExperctedMatrix[j])
                Sum = math.exp((beta)*(ExperctedMatrix[j][j-1]-Max)) + math.exp((beta)*(
                    ExperctedMatrix[j][0]-Max))
                ProbabilityMatrix.itemset((j, j-1), (math.exp((beta)*(ExperctedMatrix[j][j-1]-
                    Max))/Sum))
                ProbabilityMatrix.itemset((j, 0), (math.exp((beta)*(ExperctedMatrix[j][0]-Max))/
                    Sum))
            else:
                Max = max(ExperctedMatrix[j])
                Sum = math.exp((beta)*(ExperctedMatrix[j][j-1]-Max)) + math.exp((beta)*(
                    ExperctedMatrix[j][j+1]-Max))
                ProbabilityMatrix.itemset((j, j-1), (math.exp((beta)*(ExperctedMatrix[j][j-1]-
                    Max))/Sum))
                ProbabilityMatrix.itemset((j, j+1), (math.exp((beta)*(ExperctedMatrix[j][j+1]-
                    Max))/Sum))

    PlotList1 = []
    PlotList2 = []
    PlotList3 = []
    PlotList4 = []
    PlotList5 = []
    TimeList = []
```

```
start = rm.randint(0, n-1)
for x in range(0,n):
    if start == x:
        PlayersCount.itemset((x), 1)
        State = x
    else:
        PlayersCount.itemset((x), 0)

print("Start state: " + str(State))

stateList = [State]

for i in range(1, T) :
    for k in range(0, n):
        if State == k:
            if LastPlayersAction[k] == -1:
                LastTimePlayersPassage.itemset((k), i)
            else:
                for s in range(0, n):
                    if s == LastPlayersAction[k]:
                        if ExperctedMatrix[k][s] == 0:
                            ExperctedMatrix.itemset((k, s), (i - LastTimePlayersPassage[k]))
                            LastTimePlayersPassage.itemset((k), i)
                        else:
                            if alpha == -1:
                                ExperctedMatrix.itemset((k, s), ((ExperctedMatrix[k][s] * (1
                                    - (1/PlayerPerPlayerCount[k][s]))) + ((i -
                                    LastTimePlayersPassage[k]) * (1/PlayerPerPlayerCount[k][
                                    s]))))
                                LastTimePlayersPassage.itemset((k), i)
                            else:
                                ExperctedMatrix.itemset((k, s), (((ExperctedMatrix[k][s] *
                                    (1 - alpha))) + ((i - LastTimePlayersPassage[k]) * (
                                    alpha))))
                                LastTimePlayersPassage.itemset((k), i)
                        break

            if beta == -1:
                for w in range(0, n):
                    if round(max(ExperctedMatrix[k]), 3) == round(ExperctedMatrix[k][w], 3):
                        ProbabilityMatrix.itemset((k, w), 1)
                    else:
                        ProbabilityMatrix.itemset((k, w), 0)
            else:
                if k==0:
                    Max = max(ExperctedMatrix[k])
                    Sum = math.exp((beta)*(ExperctedMatrix[k][k+1] - Max)) + math.exp((beta)
                        *(ExperctedMatrix[k][n-1] - Max))
                    ProbabilityMatrix.itemset((k, k+1), (math.exp((beta)*(ExperctedMatrix[k
                        ][k+1]-Max))/Sum))
                    ProbabilityMatrix.itemset((k, n-1), (math.exp((beta)*(ExperctedMatrix[k
                        ][n-1]-Max))/Sum))
                elif k==(n-1):
                    Max = max(ExperctedMatrix[k])
                    Sum = math.exp((beta)*(ExperctedMatrix[k][k-1]-Max)) + math.exp((beta)*(
                        ExperctedMatrix[k][0]-Max))
                    ProbabilityMatrix.itemset((k, k-1), (math.exp((beta)*(ExperctedMatrix[k
                        ][k-1]-Max))/Sum))
                    ProbabilityMatrix.itemset((k, 0), (math.exp((beta)*(ExperctedMatrix[k
                        ][0]-Max))/Sum))
                else:
                    Max = max(ExperctedMatrix[k])
                    Sum = math.exp((beta)*(ExperctedMatrix[k][k-1]-Max)) + math.exp((beta)*(
                        ExperctedMatrix[k][k+1]-Max))
                    ProbabilityMatrix.itemset((k, k-1), (math.exp((beta)*(ExperctedMatrix[k
                        ][k-1]-Max))/Sum))
                    ProbabilityMatrix.itemset((k, k+1), (math.exp((beta)*(ExperctedMatrix[k
                        ][k+1]-Max))/Sum))

            if 0. in ExperctedMatrix[k]:
                numbers = []
                if k==0:
                    numbers.append(k+1)
                    numbers.append(n-1)
                elif k==(n-1):
                    numbers.append(0)
                    numbers.append(k-1)
                else:
                    numbers.append(k+1)
                    numbers.append(k-1)
                print(k)
                print(numbers)
                randomico = rm.choice(numbers)
                for x in range(0,n):
                    if randomico == x:
                        stateList.append(x)
                        PlayersCount.itemset((x), (PlayersCount[x] + 1))
```

```
                        PlayerPerPlayerCount.itemset((k, x), (PlayerPerPlayerCount[k][x]+1))
                        LastPlayersAction.itemset((k), x)
                        State = x
                        break
            else:
                if beta == -1:
                    Possibilities = []
                    for f in range(0, n):
                        if ProbabilityMatrix[k][f] == 1:
                            Possibilities.append(f)
                    randomico2 = rm.choice(Possibilities)
                    for x in range(0,n):
                        if randomico2 == x:
                            stateList.append(x)
                            PlayersCount.itemset((x), (PlayersCount[x] + 1))
                            PlayerPerPlayerCount.itemset((k, x), (PlayerPerPlayerCount[k][x
                                ]+1))
                            LastPlayersAction.itemset((k), x)
                            State = x
                            break

                else:
                    change = np.random.choice(players,p=ProbabilityMatrix[k])
                    for c in range(0, n):
                        if c == change:
                            stateList.append(c)
                            PlayersCount.itemset((c), (PlayersCount[c] + 1))
                            PlayerPerPlayerCount.itemset((k, c), (PlayerPerPlayerCount[k][c
                                ]+1))
                            LastPlayersAction.itemset((k), c)
                            State = c
                            break


            break
    if i \% 100 == 0 :
        print(str(i))
        print(ExperctedMatrix)
        print(ProbabilityMatrix)
        TimeList.append(i)
        PlotList1.append((ProbabilityMatrix[0])[1])
        PlotList2.append((ProbabilityMatrix[1])[2])
        if n>3:
            PlotList3.append((ProbabilityMatrix[2])[3])
        else:
            PlotList3.append((ProbabilityMatrix[2])[0])
        if n>4:
            PlotList4.append((ProbabilityMatrix[3])[4])
            PlotList5.append((ProbabilityMatrix[4])[0])
        else:
            PlotList4.append((ProbabilityMatrix[3])[0])

        print(' ')

TotalCount = sum(PlayersCount)
ProbList = np.array([0.]*n)
for b in range(0, n):
    ProbList.itemset((b), (PlayersCount[b]/(TotalCount)))
TotalProb = sum(ProbList)
CountList = list(ProbList)
for c in range(0,n):
    if ProbList[c] == 0:
        CountList[c] = 'undefined'
    else:
        CountList[c] = 1/(ProbList[c])

for i in range(0, len(players)):
    players[i] = int(players[i])

RoundedProbabilityMatrix = [[0]*n]*n

for d in range(0, n):
    ListExpectedD = list(ProbabilityMatrix[d])
    for j in range(0, n):
        if ListExpectedD[j] >= 0.999:
            ListExpectedD[j] = 1
        elif ListExpectedD[j] <= 0.001:
            ListExpectedD[j] = 0
    RoundedProbabilityMatrix[d] = ListExpectedD

print(' ')
print(ExperctedMatrix)
print(' ')
print(ProbabilityMatrix)
print(' ')
print(RoundedProbabilityMatrix)
print(' ')
```

```
print("Counter:  ")
print(str(np.matrix([players, PlayersCount])))
print("Total:    " + str(TotalCount))
print(' ')
print("Frequency: ")
print(str(np.matrix([players, ProbList])))
print("Total:    " + str(TotalProb))
print(' ')
print("Expected time for the buck to return: ")
print(str(np.matrix([players, CountList])))
plt.plot(TimeList, ([1]*len(TimeList)), 'w')
plt.plot(TimeList, ([0]*len(TimeList)), 'w')
plt.plot(TimeList, PlotList1)
plt.plot(TimeList, PlotList2)
plt.plot(TimeList, PlotList3)
plt.plot(TimeList, PlotList4)
plt.plot(TimeList, PlotList5)
plt.show()
print((PlayerPerPlayerCount))
print("States path: " + str(stateList[T-101: T-1]))
print(ProbabilityMatrix)
```

11.2 Python codes: Complete Network

```
import numpy as np
import random as rm
import math
np.set_printoptions(suppress=True)
import matplotlib.pyplot as plt

def Complete_Graph_Activity(T, n, alpha, beta):

    players = list(range(0, n))

    PlayersCount = np.array([0]*n)

    PlayerPerPlayerCount = np.array([[0]*n]*n)

    LastTimePlayersPassage = np.array([0]*n)

    LastPlayersAction = np.array([-1]*n)

    ExperctedMatrix =np.array([[0.]*n]*n)
    for k in range(0,n):
        ExperctedMatrix.itemset((k, k), -1.)

    ProbabilityMatrix = np.array([[0.]*n]*n)

    if beta == -1:
        for h in range(0, n):
            for w in range(0, n):
                if max(ExperctedMatrix[h]) == ExperctedMatrix[h][w]:
                    ProbabilityMatrix.itemset((h, w), 1.)

    else:
        for h in range(0, n):
            Max = max(ProbabilityMatrix[h])
            ListProbability = np.array([0]*n)
            NoDiagonal = list(range(0, h)) + list(range(h+1, n))
            for v in NoDiagonal:
                ListProbability.itemset((v), (math.exp((beta)*(ExperctedMatrix[h][v]-Max))))
            for w in NoDiagonal:
                ProbabilityMatrix.itemset((h, w), ((ListProbability[w])/(sum(ListProbability))))


    start = rm.randint(0, n-1)
    for x in range(0,n):
        if start == x:
            PlayersCount.itemset((x), 1)
            State = x
        else:
            PlayersCount.itemset((x), 0)

    print("Start state: " + str(State))

    stateList = [State]


    for i in range(1, T) :
        for k in range(0, n):
            if State == k:
                if LastPlayersAction[k] == -1:
                    LastTimePlayersPassage.itemset((k), i)
                else:
                    for s in range(0, n):
                        if s == LastPlayersAction[k]:
                            if ExperctedMatrix[k][s] == 0:
                                ExperctedMatrix.itemset((k, s), (i - LastTimePlayersPassage[k]))
                                LastTimePlayersPassage.itemset((k), i)
                            else:
                                if alpha == -1:
                                    ExperctedMatrix.itemset((k, s), ((ExperctedMatrix[k][s] * (1
                                        - (1/PlayerPerPlayerCount[k][s]))) + ((i -
                                        LastTimePlayersPassage[k]) * (1/PlayerPerPlayerCount[k][
                                        s]))))
                                    LastTimePlayersPassage.itemset((k), i)
                                else:
                                    ExperctedMatrix.itemset((k, s), (((ExperctedMatrix[k][s] *
                                        (1 - alpha))) + ((i - LastTimePlayersPassage[k]) * (
                                        alpha))))
                                    LastTimePlayersPassage.itemset((k), i)
                            break

                if beta == -1:
                    for w in range(0, n):
                        if max(ExperctedMatrix[k]) == ExperctedMatrix[k][w]:
```

```
                    ProbabilityMatrix.itemset((k, w), 1.)
                else:
                    ProbabilityMatrix.itemset((k, w), 0.)

            else:
                Max = max(ProbabilityMatrix[k])
                ListProbability = np.array([0.]*n)
                NoDiagonal = list(range(0, k)) + list(range(k+1, n))
                for v in NoDiagonal:
                    ListProbability.itemset((v), (math.exp((beta)*(ExperctedMatrix[k][v]-Max
                        ))))
                for w in NoDiagonal:
                    ProbabilityMatrix.itemset((k, w), ((ListProbability[w])/(sum(
                        ListProbability))))



            if 0. in ExperctedMatrix[k]:
                numbers = list(range(0, k)) + list(range(k+1, n))
                randomico = rm.choice(numbers)
                for x in range(0,n):
                    if randomico == x:
                        stateList.append(x)
                        PlayersCount.itemset((x), (PlayersCount[x] + 1))
                        PlayerPerPlayerCount.itemset((k, x), (PlayerPerPlayerCount[k][x]+1))
                        LastPlayersAction.itemset((k), x)
                        State = x
                        break
            else:
                if beta == -1:
                    Possibilities = []
                    for f in range(0, n):
                        if ProbabilityMatrix[k][f] == 1.:
                            Possibilities.append(f)
                    randomico2 = rm.choice(Possibilities)
                    for x in range(0,n):
                        if randomico2 == x:
                            stateList.append(x)
                            PlayersCount.itemset((x), (PlayersCount[x] + 1))
                            PlayerPerPlayerCount.itemset((k, x), (PlayerPerPlayerCount[k][x
                                ]+1))
                            LastPlayersAction.itemset((k), x)
                            State = x
                            break

                else:
                    change = np.random.choice(players,p=ProbabilityMatrix[k])
                    for c in range(0, n):
                        if c == change:
                            stateList.append(c)
                            PlayersCount.itemset((c), (PlayersCount[c] + 1))
                            PlayerPerPlayerCount.itemset((k, c), (PlayerPerPlayerCount[k][c
                                ]+1))
                            LastPlayersAction.itemset((k), c)
                            State = c
                            break


            break

    TotalCount = sum(PlayersCount)
    ProbList = np.array([0.]*n)
    for b in range(0,n):
        ProbList[b] = PlayersCount[b]/(TotalCount)
    TotalProb = sum(ProbList)
    CountList = list(ProbList)
    for c in range(0,n):
        if ProbList[c] == 0:
            CountList[c] = 'undefined'
        else:
            CountList[c] = 1/(ProbList[c])



    print(' ')
    print(ExperctedMatrix)
    print(' ')
    print(ProbabilityMatrix)
    print(' ')
    print(' ')
    print("Counter:  ")
    print(str(np.matrix([players, PlayersCount])))
    print("Total:    " + str(TotalCount))
    print(' ')
    print("Frequency: ")
    print(str(np.matrix([players, ProbList])))
    print("Total:    " + str(TotalProb))
```

```
print(' ')
print("Expected time for the buck to return: ")
print(str(np.matrix([players, CountList])))
```