Considering Computational Costs in Non-Cooperative Games

Thesis Dissertation

Department of Economics, University of Macedonia

Stylianos Daskalinas (eco19109)

Supervising Professor: Eleftherios Filippiades

Thessaloniki, June 2022

Abstract

In this thesis we explore the effects of computations on non-cooperative games. Initially, we examine the practical implications of large computational processes to the solution of games. Then, basic notions and concepts of complexity theory are introduced in order to understand the effects of complexity on games. We proceed to discuss some central topics of the field of game complexity, with the most important one being the intractability of the concept of Nash equilibrium. The examination of complexity from the perspective of players follows. Finally, a concept of the effects of computational complexity on strategic decision making is developed and discussed.

Key words: Non-cooperative games, algorithm, running time, complexity class, benefit, cost

Acknowledgements

I would like to thank my supervising professor, Eleftherios Filippiades, for his guidance, comments and assistance, which helped improve this thesis, as well as for the useful advice and direction he provided regarding my postgraduate course. I would also like to thank my parents and my brother for the invaluable support and motivation they provided throughout the writing of this thesis.

Contents

List of Tables

1. Introduction

Game theory is a most important field of modern economics. A simple search will yield a variety of definitions but, in general, we can think of Game Theory as the study of strategic interaction. The mathematical structure and the deep connection to the principles of economic philosophy make Game Theory a nearly flawless framework for the study of economic interaction. With relatively reasonable assumptions, such as rationality of economic agents, Game Theory provides a toolbox we can use to understand and predict each agent's behaviour, and, also the outcome of the interaction, namely the equilibrium of a game. Furthermore, thanks to the great mathematician and game theorist John F. Nash, we have a mathematical proof that, given some minimal requirements, such an equilibrium does exist. Consequently, we can predict the outcome of numerous interactions of economic nature by reducing them to simple mathematical models we can then easily solve.

Meanwhile, Game Theory has found application in many other disciplines, providing methods of analysis as well as solutions, in problems of economic nature. In Biology, non-cooperative dynamic games can be used to model and predict the interaction of species within an ecosystem. In Computer Science, notions such as Pareto equilibrium ensure that a computer device optimally allocates its resources with respect to the tasks it is programmed to do. In Physics, and more specifically, in the field of Quantum Cryptography, games such as the Battle of Sexes with Incomplete Information and other coordination games are used in conjunction to principles of Quantum Mechanics in order to coordinate communication under uncertainty and achieve virtually perfect secrecy.

1.1 Facing the Problem

Unfortunately, all the benefits game theory provides come with an important caveat. The methods of game theory can only be useful when their application is computationally tractable. That is, when we can carry out all the necessary computations involved in analysing interactions (through game-theoretic methods) within a humanly acceptable time interval. This can be quite demanding for humans as our computational potence can only get us so far.

We can demonstrate this by a simple and interesting example. Let us consider the game of Hex. Invented by Piet Hein in 1942 (and independently reinvented by John Nash in 1948), the game of Hex is a 2-player game played on a board of 11x11, 13x13 or 19x19 rhombus-like tiles. The purpose is to mark tiles with one colour in order to create a path between two opposing sides of the board (a certain colour and a pair of opposing sides are assigned to each player before

the game starts). The game can be obviously classified as a perfect information sequential game. Furthermore, by the famous Hex theorem (proven by Nash in 1949), we are assured that this game cannot end in a draw, effectively foreseeing that there will be a winner and a loser. Additionally, given that one of the players has a winning strategy, we can deduce whether the game will end in an odd or an even number of total moves. Yet, a sequential game approach will probably be challenging for humans. In an 11x11 board, a rough estimate is that $\binom{11^2}{\frac{1}{4}11^2} \times \binom{\frac{3}{4}11^2}{\frac{1}{4}11^2} \times \frac{1}{2} = 1.60262586 \times 10^{52}$ games can be carried out. This number is nothing but enormous. Let us point out that, in seconds, the lifetime of the universe is only $4.36117077 \times 10^{17}$. Therefore, thoroughly calculating every possible contingency of the game of Hex would be impossible for a human.

The game of Hex is only one of many cases of strategic interaction that require a significant number of computations to be thoroughly studied. Furthermore, the burden of having to execute an overwhelming number of computations to derive equilibria is not only present in sequential games. To support this claim, we can consider the following example: Assume 700 residents of a village are about to vote on their municipality's next public project where they can vote for a public library or a new park. In the case where a simple voting rule is applied (such as the majority rule) and given that every voter has well defined preferences (either prefers the library or prefers the park or is indifferent between the two), a Nash equilibrium is guaranteed to exist. Each resident has only two available actions (vote for library and vote for park), yet, since 700 residents participate, there are 2^{700} possible outcomes. Now, to derive an equilibrium for such a voting game, we would have to consider each outcome from the perspective of each voter, therefore, we would need to account for as many as $700 \times 2^{700} = 3.68209513 \times 10^{213}$ payoffs[1]. This huge number is certain to challenge any computer as it is even greater that the number of possible Hex games. Imagine, then, how this computational challenge would escalate if three or four alternatives were available to even more voters.

Cases such as those discussed above justify and even enforce the employment of computer devices to deduce equilibria in a reasonable time interval. Given their computational might, computers are capable of solving games for us much faster and without the possibility of human error (assuming they are properly instructed). However, computers would struggle and fail to handle solving the examples presented above. Unfortunately, even computers are limited in

[1] Despite the large number of data we need to consider, not much else is required to solve this particular game, since the Nash equilibrium is simply the case where everyone votes their preferred choice or randomly chooses between the two if they are indifferent.

how many operations they can execute in a second. For instance, Intel and Cary's Aurora supercomputer (located in the US, about to be completed in late 2022) will be able to execute 10^{18} computations per second. This means that it will be able to properly analyse the voting problem discussed above in approximately $700 \times \frac{2^{700}}{2 \times 10^{18}} = 1.84104757 \times 10^{195}$ seconds (assuming that we are comparing payoffs in search of a pure strategy equilibrium). This means that the Aurora will have to perform a computation lasting multiple times the lifetime of the universe.

The above discussion establishes that the computational magnitude of the process we will have to go through to solve a game is a defining parameter to whether we can find its solution. At this point, we need to invoke some notions of computer science in our discussion to formally examine the computational issues of game theory.

1.2 A Quick Reminder

Before we turn to computer science, we must go over some basic notions of game theory. A static, perfect information game is a tuple of the form $G = (N, A, u)$, where, N is the set of players (we also use N for the number of players), $A = \times_{i=1}^{N} A_i$ is the action set, where A_i is the set of actions available to player i and $u = (u_1, u_2, \dots, u_N): A \to R$ is an array of payoffs. For each player i, $u_i(a_1, a_2, \dots, a_N)$ assigns a payoff (player i will receive) to each possible combination of actions which may be chosen by the players. Players are assumed to be rational, seeking to maximise their payoff. Therefore, for every combination of other players choices, they choose the action $a_i = argmax_{x \in A_i} u_i(x, a_{-i})$, also called a best response of player i to the combination of a_{-i} opposing actions.

A Nash equilibrium (NE) in pure strategies is an array of actions $a^* \in A$ such that, for every player i, there is no incentive to unilaterally deviate, i.e., $u_i(a_i, a^*_{-i}) \leq u_i(a^*), \forall\, a_i \in A_i \backslash \{a^*_i\}$. This concept is central in game theory as it describes a situation where each player responds optimally to every other player. In the case where there is no array satisfying the condition for a pure-strategy NE, players need to turn to mixed strategies.

A mixed strategy is a probability distribution over the player's actions. Since the player cannot choose an action based on his opponents' actions, he will randomize over his options, attempting to maximise his expected payoff. The randomization case includes the pure-strategy equilibrium (via a distribution where all probabilities but those of the pure-strategy equilibrium actions are zero). Formally, a collection of distributions over the action set $r =$

$((r_i(a))_{a\in A_i})_{i\in N}$ is a mixed-strategy NE if and only if, for every player i, $u_i(r) \geq \sum_{a\in A_i} \rho_i(a)u_i\,(a, r_{-i})$ for any other distribution ρ_i over the action set A_i . This is the general form of the NE and, as proven by Nash, for every finite game (i.e., every game where, for every $i \in N,\ |A_i| \in \mathbb{N}$) there exists at least one collection of distributions satisfying the condition set above.

2. Concepts from the Theory of Computation

The theory of computation is mainly focused on the study of computational problems. These are problems which typically require a large and complex sequence of operations to be solved, usually too large for humans to efficiently undertake. So far, for the problems which cannot be simplified or tackled by abstract and theoretical means and researchers' ingenuity, only a combination of human guidance and computers' power has proven capable to provide effective solutions.

2.1 Algorithms and Computational Problems

Computers contribute their capabilities for fast, tireless, and flawless execution of steps while humans provide a carefully designed list of instructions for the computers to follow, in order to tackle difficult problems, which require super-human computational potence. Let us examine the nature of these problems.

A most simple problem is that of SORTING, where we are given a collection of numbers (or other objects) and we are asked to list them in accord to a given ordering. Another well-known problem is that of SATISFIABILITY. In this problem, we are given a list of Boolean variables and a Boolean expression, and we are asked to deduce whether a combination of values for the given variables exists, such that the final value of the expression is "truth". However, not only abstract computational problems can be tackled by algorithms. Economic problems can also be solved by algorithms. Consider the well-known Matching Problem, which, for example may refer to assigning students to schools such that no school or student has an incentive to deviate (i.e., no student wishes to change school and no school wishes to reject a student), given each school's and student's preferences. The famous Gale-Shapley algorithm has been proven to efficiently solve this problem. Among other scientific achievements, this algorithm helped Lloyd Shapley earn the Nobel Prize in Economics in 2012.

Some important classes of problems, categorised with respect to their output, are:

I. Decision problems: the output is either true or false
II. Optimization problems: the output is the value of an object or structure when optimized
III. Search problems: the output is the point/value at which some given condition is satisfied

For example, finding a pure-strategy Nash equilibrium can be considered as a search problem, where we look for a best response action available to the player. The player's payoffs, the number of players and each player's actions corresponding to each payoff would be the initial data of the problem, while the solution would be a best response action. The main reason for this division is that the form of the solution dictates (to some extent) the nature of the solution process, therefore establishing some common features between the solutions of each category's problems. However, it is not uncommon for a problem to allow for more than one type of solution to be produced, or to be formulated to match more than one category. For example, finding the cheapest route to a destination could provide an optimization related solution (the cost of the cheapest route) and a search related solution (the cheapest route itself).

Problems are generally solved with algorithms. An algorithm is technically a sequence of well-defined steps one has to execute in order to solve a problem. Each algorithm has an input and an output. The input is a collection of given parameters relevant to the problem the algorithm is designed to solve while the output is the solution of the problem. The algorithm contains a list of instructions a computer device has to follow which, using the inputs, should produce an output which solves a specific problem. This is where complexity is involved. As established above, we need the problem to be solved in a practical amount of time, something even computers may struggle with (usually due to a very long sequence of repetitions of certain operations). We also need our algorithms to use reasonable amounts of resources such as memory space, but for our discussion only time is relevant. It should be noted that an algorithm is not a program, but rather the part of the program which is relevant to solving the problem at hand.

2.2 Time Complexity

So far, we have discussed algorithms and problems but made almost no comment on which device carries out the computations for us. In complexity theory, problems are usually considered to be computed by a Turing Machine (TM). It is a theoretical model of an abstract device which can carry out computations and has infinite storage space. It is equivalent to a programming language with a specified set of functions. Informally, it includes a set of tools

which, with proper instructions (an algorithm), can be used to solve a problem. TM can only receive certain symbols as inputs and produce only certain symbols as outputs. The famous "Machine Language", $\{0,1\}$ is the set of symbols compatible with the most generalised version of a TM.

Considering a computational problem P as a function $P(input)$, we say that a TM computes $P(input)$ in $T(input)$ if and only if, given some input x, its output is $P(x)$ and is conducted in $T(x)$ steps. Many details of the definition and properties of TM are omitted here, since our concern is mainly the intuitive understanding of time complexity, necessary for the analysis of complexity within the context of game theory.

One of John F. Nash's academic interests was cryptography. In his research, he made three very important realisations. First, that we must measure the time it takes for an algorithm to execute all the steps it includes not in physical time (seconds) but in "running" time, which refers to the number of steps the algorithm includes. The use of running time allows us to study the complexity-related properties of the algorithm independently of any computational device that might be used, but also allows for compatibility with any computational device, since we can simply multiply the number of steps with the time a particular device requires to execute one step. Second, Nash deduced that we must examine the running time of an algorithm as a function of the number of inputs. This way, we can treat inputs as a variable and trace the growth of the running time. Most fundamental monotonically increasing functions are met in the study of algorithms' running time. Linear, quadratic, cubic and exponential expressions are the ones most frequently met in algorithm analysis. Nash's third finding is only indirectly relevant to our discussion but certainly worth mentioning. Nash conjectured that perfect encryption may be achieved not by extremely sophisticated encryption methods but instead, by encryptions which are computationally impractical. For us, a reversed version of this statement might suggest that if computations take impractically long to execute, some problems will forever remain unsolved.

Nash's idea of representing time complexity as a function of inputs has been widely accepted as the main method of expressing time complexity. This approach has a very important feature which makes it extremely efficient with respect to algorithm analysis and complexity analysis. That is, algorithms usually include loops. Loops are structures of repetition which are included in the solution of a problem and/or the verification of a calculated solution. In order to better understand the concept, let us examine SATISFIABILITY. In this problem, the simplest

method of solution is the sequential testing of vectors of Boolean variables against the given Boolean formula until we reach an acceptable vector (one that satisfies the Boolean formula, making it take a value of 1 instead of 0) or until we run out of vectors to test. The latter case, as well as the case where the last combination of values is the only one satisfying the formula, will require an exponential number of tests.

```
SATISFIABILITY algorithm
Input: Φ(x_1, x_2, … , x_n): {0.1}^n → {0,1}
t ← 0
While (x_1, x_2, … , x_n) ∈ {0.1}^n  AND  t = 0
  If  Φ(x_1, x_2, … , x_n) = 1 Then
    t ← 1
  End_If
End_While
Write t
```

Table 1 - SATISFIABILITY Algorithm

Satisfiability is included in the class of decision problems since its output (t) is either 1 if the problem admits an acceptable combination of 0s and 1s and is 0 otherwise. As we can see from Table 1, with a given formula of n variables, where each variable $x_i \in \{0,1\}$, the algorithm will test all the possible combinations of values the variables may take until any of the two conditions set in the "While" loop is violated. The second condition ceases to hold only in the case where a satisfactory combination is found while the first will hold until every possible combination is tested. In the worst-case scenarios, the algorithm will exhaust all the possible combinations. The running time will be equal to the number of possible combinations which is simply the cardinality of the cartesian product of each variable's domain ($\{0.1\}$). The cardinality of $\{0,1\}^n$ is 2^n since it is the product of the cardinalities of n 0-1 sets. This algorithm for Satisfiability has a running time of $T(n) = 2^n$, therefore, as inputs increase, the time complexity grows exponentially.

We can trace certain common traits between Satisfiability and the voting problem discussed in the introductory section. Firstly, for the case of two voting options, we can transform each player's action into a Boolean variable. Player i's action will be $x_i = 1$ if the player votes for the first option and $\neg x_i = 0$ if the player votes for the second option. Second, the inputs of the voting game are the same as the number of steps involved in the search for an acceptable combination of Boolean variables, which is 2^n. Third, with some relatively simple additions and alterations, the algorithm presented and discussed above would be a decent candidate for the simplistic computation of a best response in pure strategies for the voting game.

Calculating running times is important to deduce both the feasibility of a solution but also for comparing solutions with respect to resource consumption. Now, for the comparison of algorithms, it is often useful to omit certain details of each algorithm's running time. The number of commands included outside and inside of loops are parameters which, despite being important in individual algorithm analysis, might be misleading in the case of comparison. When comparing algorithms, we need to pay attention to large scale differences between the algorithms. For example, if both algorithms' running times are polynomial, but of different order, we can neglect the lower order terms of each running time and find an input size for which either the first or the second running time is lower.

The generalisation of this concept is the famous big-O notation. With big-O notation, we do not focus on the exact running time function of an algorithm but rather on the (functionally simplest) lowest possible expression which bounds our function from above, for a certain group of input sizes. Mathematically, we say that $T(n)$ is $O(f(n))$ if and only if $\exists\ c > 0$ and $n_0 \geq 1$ such that $T(n) \leq cf(n)\ \ \forall\ n \geq n_0$. This definition's main advantage is that we only focus on the variable part of the running time but not the parameters. The functional form of $O\big(f(n)\big)$ can help us classify problems with respect to the time complexity of the algorithms that solve them. The functional forms most frequently met are the polynomial ($O(n^k)$) and the exponential ($O(k^n)$) forms, which are graphically represented as such:

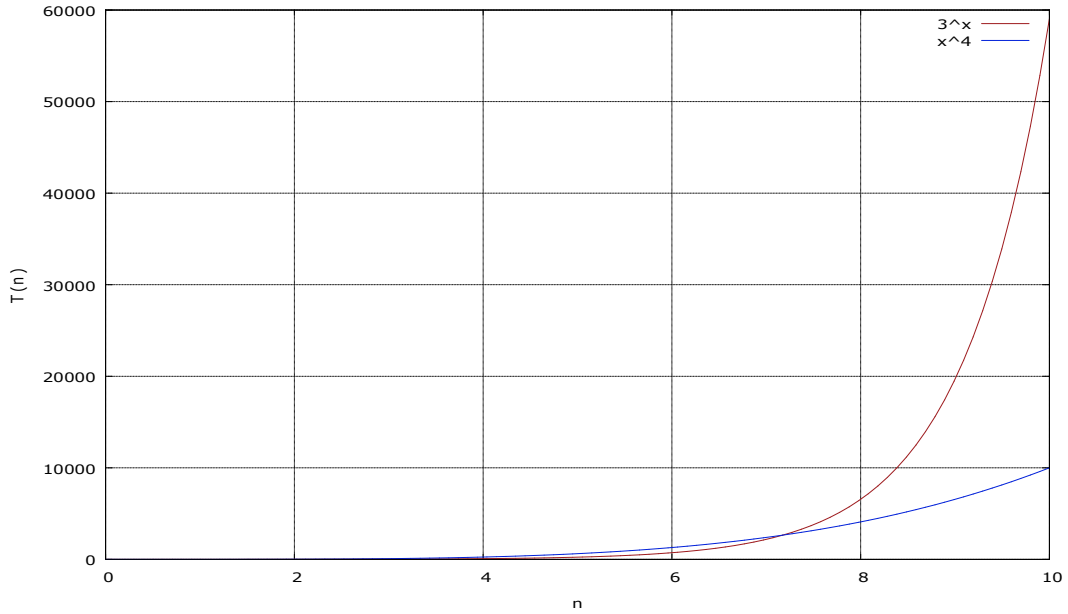

Table 2 – Plot of Running Time Growth for Exponential and Polynomial Time

2.3 Complexity Classes

In general, it is useful to classify problems with respect to their bounding running time, since common time complexities suggest that problems are solved in a similar manner.

In formal terms, a complexity class is a collection of problems which can be computed in a bounded number of steps. For example, for n inputs, SATISFIABILITY's running time may be anywhere between 1 and $O(2^n)$. Similarly, sorting a list of n numbers may require anywhere between n and $O(n^2)$ steps while, exactly $O(n^2)$ steps will be made to achieve a proper matching between, say hospitals and doctors.

The largest complexity classes relevant to our discussion are $DTIME(T(n))$ and $NTIME(T(n))$. The first class includes all the problems which can be solved in $T(n)$ steps by a deterministic TM, which is any TM formulated to execute a completely predesigned process. The second class includes all the problems which can be solved by non-deterministic TM, which are TM capable of following more than one predesigned processes. Non-deterministic TM are relevant to decision problems (where output is 0 or 1) since those are problems which may run indefinitely for an input and may admit no solution for a finite input. In contrast, algorithms for optimization problems and search problems may run in exponential time but are bound to eventually yield the desired result for any finite input.

A most important complexity class in our discussion is that of **P**. This class contains all the problems in $DTIME(T(n))$ with a running time of $O(n^k), k \in \mathbb{N}$. Formally, we can say that $\mathbf{P} = \bigcup_{k\geq 1} DTIME(n^k)$, which means that it contains all the problems the running time of which has a polynomial upper bound. Many important problems such as that of Sorting and the Matching problem are in **P**. It represents the set of problems we can feasibly calculate, since a shift in the input size will cause a percentage change of (#inputs-1). For an $O(n^5)$ problem, 100 inputs will only require 10^{10} computations, much smaller than those met in the voting problem. We would wish every problem to be in **P**, in which case all we would have to do to solve problems faster would be to improve our technology. Unfortunately, many important problems are outside of **P**, for one of two reasons. Either they are not in $DTIME$ or they are not polynomial.

The second case is pretty hard to alleviate as, by definition, some problems are just too hard to solve. For example, consider the following problem: Given a list of n objects, find all the possible permutations of the objects. By elementary combinatorics we can deduce that $n!$ possible permutations exist. By Stirling's approximation we know that $n!$ is lower bound by an

exponential expression (formally, $n!$ is $\Omega(n^{\frac{1}{2}}\left(\frac{n}{e}\right)^n)$ where Ω is similar to the Big-O notation but for lower bounds), therefore the running time will be even greater than exponential.

However, the case we are most concerned with is the first one, where a problem's running time might be outside of $DTIME$. Here, we need to introduce the most famous class of problems, the **NP** class. It is defined as the class containing all the problems which can be verified in polynomial time, in other words, if we are given a solution for a problem and we are able to devise an algorithm with polynomial time in order to verify the solution, then the problem belongs in **NP**. The initials **NP** stand for "**N**on-deterministic **P**olynomial" and it can be proven that **NP** $= \bigcup_{k\geq 1} NTIME(O(n^k))$. Many important problems such as the Traveling Salesman Problem (optimal path that crosses every node on a graph, given that crossing each edge inflicts costs) and the SATISFIABILITY problem we discussed earlier belong in **NP**. By including some verifying commands in the algorithm of a problem of **P**, we can attach the verification process in the algorithm which makes a **P** problem's verification polynomial in time, therefore, we understand that a problem in **P** also belongs in **NP**. Despite the similarity between **P** and **NP** due to their polynomial nature, we cannot (for now) claim that the two classes are equivalent or equal. **NP** contains problems which, so far seem impossible to solve in polynomial time. It is worth noting that the Nash equilibrium belongs in **NP**. To see why, consider the following algorithm which takes an array of actions of an N-player game as inputs and outputs a yes/no answer depending on whether or not the array is a NE in pure strategies:

```
NASH VERIFICATION algorithm
Input: G = (N, A, u), a* ∈ A
 t ← 0
While i ∈ N AND t = 0
  While a_i ∈ A_i\{a_i*} AND t = 0
    If u_i(a_i, a*_-i) > u_i(a*) Then
      t ← 1
    End_If
  End_While
End_While
Write t
```

Table 3 - Nash Verification Algorithm

The algorithm depicted in Table 3 has a trivial verification process. By the definition of the NE in pure strategies, no player must be willing to deviate (the inverse of which is tested in the if/then structure), given that all the other players choose their corresponding equilibrium actions. The inner and the outer loop will terminate once someone willing to deviate is found, signifying that the given array is not a Pure-Strategy equilibrium. The worst-case running time

is governed by product of the length of the outer and the inner While loops. In fact, this is simply $\sum_{i\in N}(|A_i|-1)$. In a convenient symmetric game example with k actions available to each player, the running time would be $O(N(k-1))$, which is linear (and therefore polynomial), both in N and in k. This verifies that the NE in pure strategies in in **NP**.

It is important to note that, if we were to run this algorithm through a computer, we would need to transform all the information of the game to machine language. Generally, the transformation to machine language alters the running time function ($T(n)$) to $klog|\Theta|T(n)$, where k is a constant and Θ stands for the set symbols we use in the original formation of the problem. Here we neglect this transformation as we need to focus on the complexity of the problems alone and not their relation to computers.

Now, let us examine the concept of Reduction. In the case where a problem is particularly difficult to solve, we might be able to relate it to a simpler problem which is easier to solve. This is the main concept behind reduction. A thorough review of the reduction of problems requires the examination of some advanced concepts and techniques of the theory of computation which are best avoided in our game-theoretic discussion. However, there is a last concept we will need in order to better understand the NE. This concept is "completeness". We say that a problem is complete for a certain complexity class when all the problems of that class can be reduced to this problem. For example, it can be proven that SATISFIABILITY is **NP**-complete. This implies that any problem in **NP** can be transformed to the search for a Boolean vector that satisfies a Boolean expression. One might expect that NE would also be an **NP**-complete problem but, as we will infer later, that is unlikely.

3. The Computational Complexity of Games

Since games can be treated as problems, their time complexity can be a complication for us, both in analysing and in solving them. Unfortunately, even a simple n player symmetric static game where each player has k available actions, has an input size of k^n for each player's optimal action and total input size of nk^n . The running time will be derived by a long sorting process each player will have to conduct for every other player's actions and the player's own actions. However, this will only be sufficient for a search in pure strategies, while mixed strategies will further increase the running time. This "super-exponential" input can easily

make games impossible to handle for a third-party observer, since the mixed strategy of every player must be derived.

3.1 Graphical Games

An important step towards reducing overwhelming input size is the introduction of Graphical Games. The method of graphical games suggests that, instead of representing games as a large collection of matrices (n matrices for a symmetric game of n players, each with s^n entries), we can, in certain cases, use a collection of smaller dimension matrices along with an n-node undirected graph. This representation depends on the fact that, if a player's payoffs are only affected by a small group of other players, then we can omit all the unnecessary matrix entries. In the graph of the game, each player is represented by a node and two nodes are connected only in the case where the two players' payoffs are directly dependent. The matrices' dimensions are equal to the cardinality of the largest closed neighbourhood found in the graph. This means that for any case where not all players' payoffs are interconnected, we can reduce the total input number. Such reductions may make sorting through all the possible outcomes of the game a much easier (yet, unfortunately still exponential) task.

The format of graphical games has one more advantage against the usual (tabular) representation format. In tabular representation, the simplest method of search for Nash equilibria is a huge sorting of all the inputs, possibly requiring a running time of $O([n2^n]^k)$, which can rapidly escalate to overwhelmingly large amounts of steps.

On the other hand, graphs are compatible to more sophisticated search processes, the employment of which may help us reduce not only the input of the game but also the running time for a solution algorithm. Search in graphs has important advantages as it is of sequential nature, somewhat similar to dynamic programming. The creators of graphical games {to be mentioned later in the lit review section} actually went as far as proposing a graph search algorithm which can find Nash equilibria in exponential time (certainly preferable to the super-exponential alternative of tabular games). However, the proposed algorithm can run in polynomial time given a relaxation of our requirements regarding the result, from a Nash equilibrium to an approximate Nash equilibrium (also called ε-equilibrium)

By employing certain results of graph theory and by carefully manipulating certain inequality relations, we can deduce ε-equilibria for any game. More specifically, we can produce $h\varepsilon$-equilibria where h stands for the cardinality of the largest closed neighbourhood found in the game's graph. The proofs regarding the validity of the proposed algorithm's results both in

absolute and approximate terms, as well as the running time proofs, are the result of an intelligent combination of graph theory, algorithm design and game theory, quite beyond the scope of the present study. We must note that the efficiency of graph search for finding equilibria can only be harnessed in the case where not all payoffs are interconnected. In other words, if all the vertices of a game's graph are connected, then the search process ceases to be beneficial.

3.2 Equilibria Calculation

The deduction of an efficient algorithm for finding pure and mixed strategies equilibria in multiplayer games is an open problem in game theory and any other disciplines involved. The direct approach would be a large sequence of comparisons which would inevitably require an exponential number of steps. With more than two players, the data of a game would correspond to a tensor rather than a matrix, requiring long lists of pivoting operations which would rapidly increase running time. A more computer-science oriented approach would include the use of an algorithm similar to the one discussed in 2.4, where the very structure of verification can be advantageous in the quest for an equilibrium in pure strategies. Consider the following example:

```
NASH DERIVATION algorithm
Input: G = (N, A, u)
 t ← 0
While a ∈ A AND t = 0
 k ← 0
 While i ∈ N AND k = 0
  While b ∈ A_i\{a_i} AND k = 0
   If u_i(b, a_−i) > u_i(a) Then
    k ← 1
   End_If
  End_While
 End_While
 If k ≠ 0 Then
  a* ← a
  t ← 1
 End_If
End_While
If t = 0 Then
 ∄ a NE in pure strategies
Else
 Write a*
End_If
```

Table 4 - Nash Derivation Algorithm

The algorithm presented in table 4 is optimised for quick search. First of all, it is designed to terminate search once the condition for Nash equilibria is breached, therefore no additional running time is spent in the given array of actions. Additionally, if a pure strategy NE exists, the algorithm will stop once the corresponding array is found, decreasing to the lower bound of the running time. The outer "While" loop will at most run for as many as $\prod_{i\in N}|A_i| = |2^A|$ steps. As discussed above, the inner structure has a running time of $\sum_{i\in N}(|A_i| - 1)$, therefore, the total running time will be $\prod_{i\in N}|A_i| = |2^A| \times \sum_{i\in N}(|A_i| - 1)$. Considering a symmetric game with k available actions, the running time will be $O(N(k-1)k^N)$, which is greater than exponential. The algorithm can easily be modified to calculate all the pure-strategy equilibria of the game by adding a few commands and turning some of the "While" loops into "For" loops. However, this would increase the lower bounds of the algorithm. The derivation of mixed strategy equilibrium would require a very different approach since we would need to conduct a search process similar to the one presented above over a continuous set of distributions. In games with only two players, some geometric arguments can be made to reduce the derivation of mixed strategies to pivoting, however, the process cannot (yet) be generalised to n-player games and, even for two players, the algorithm to reach an equilibrium has a super-polynomial running time.

3.3 The Complexity of Nash Equilibria

The preceding discussion essentially sets the stage for this section. As we have seen, computational complexity is important in solving problems just as much as the solution method itself. Furthermore, games are not beyond complexity implications and we can find more than a few examples where conventional methods of game analysis may prove computationally intractable. However, complexity is not an isolated incident of only a few games but is proven to be present in the very nature of our most praised and used solution concept, the Nash equilibrium.

In 1951, John F. Nash proved beyond doubt that in every finite game, each player has at least one mixed strategy he is not willing to deviate from. In his proof, Nash made use of the very nature of mixed strategies and of a very important theorem of mathematical analysis in order to make his case. The theorem was Brouwer's very famous Fixed Point Theorem, which has many uses in economics, mainly in general equilibrium theory (for the proof of existence of market equilibria).

Nash's proof depended on the following steps: At first, he defined the (non-negative) excess benefit from deviating from a mixed strategy to a particular pure strategy. Then he moved to define a function of this benefit and the mixed strategies for which Brouwer's theorem applies, carefully designing it in accordance to the definition of mixed strategies (henceforth called the Nash function). He proceeded to apply Brouwer's theorem to this function. The function's fixed point must be a mixed strategy since, by definition, the Nash function maps mixed strategies to mixed strategies. From there, Nash made his case by proving that if players were to deviate from the mixed strategy proposed by the fixed point, the relation produced from the application of Brouwer's theorem would result to a contradiction.

In another proof, Nash made use of Kakutani's fixed point theorem which mainly concerns correspondences instead of singleton functions. Here, the existence of a fixed point suggests that the fixpoint best response array is a maximum for each player's payoff, meaning players have no actual motive to deviate. This proof makes use of the hemi-continuity and convexity of best response functions as well as the structure of the cartesian product of the mixed strategies sets. Convexity flows from the nature of probability while hemi-continuity depends on the properties of the utility function in conjunction to the definition of the best response function (proved by Berge's Maximum theorem).

The proof which utilises Brouwer's fixed point theorem has two very important consequences. The first is that a mixed strategy array no player wants to deviate from necessarily exists. The second is that the desired mixed strategy is a fixed point. The importance of the first result is already recognised by the vast majority of the academic world and praised in most game-theoretic texts. However, the second result is what our discussion will focus on. The fact that the desired equilibrium of the game is a fixed point of the Nash function is frustrating. That is, because Brouwer's fixed point theorem proves the existence of the fixed point but does not help us calculate its value.

The search for fixed points, when we know they exist, ultimately involves sequential steps of repeated approximations and verifications, therefore an algorithm is required. However, Brouwer's proof is not constructive, in other words, it does not include a process of construction for the fixed point but rather demonstrates that its absence will lead to a contradiction. Consequently, creativity is a prerequisite in the search for a fixed point.

The discussion on efficient algorithms for fixpoint calculation is certainly an interesting one, concerning mathematicians, economists, computer scientists and specialists from other

disciplines. Unfortunately, a proof by Hirsch, Papadimitriou and Vavasis (1989) totally eliminates any chance of fixpoint location in non-exponential time. Through a sophisticated combination of geometry, distance and continuity arguments of mathematical analysis, we can see that the running time for an algorithm scouting for Brouwer fixed points has a lower bound exponential to the level of accuracy we wish to achieve.

Now, this difficulty of tracking fixed points and correspondingly, Nash equilibria, may lead one to attempt to reduce our problem to another, simpler one, in the same class and achieve solution through reduction. As we stated before, NE belongs in the **NP** class so we could try to reduce it to another **NP** problem but this technique is unlikely to yield the results we wish. This is because Nash equilibrium is not **NP**-complete. To see this, let us think of SATISFIABILITY. Since it is proven to be **NP**-complete, and assuming NE is also **NP**-complete, we should be able to reduce each problem to the other. However, this proves to be highly improbable due to the fact that, as we have discussed above, NE certainly has at least one solution, while SATISFIABILITY may or may not admit one. Therefore, the reduction technique seems inappropriate for the study of NE. Consequently, we need another class of problems in which NE will be complete. Such a class must include computational problems for which the existence of a solution will be guaranteed. Now, if this class includes computationally intractable problems, i.e., problems with running time greater than polynomial, we will be certain that the concept of NE is not computationally efficient.

Such a collection of classes was introduced by C. Papadimitriou (1991). More specifically, the class we are interested in is **PPAD**, which stands for "Polynomial Parity Argument for Directed graphs". At the centre of this class lies one very special computational problem called "End of Line" (EL). The formulation of the problem is as follows: in a directed graph, a vertex is called "unbalanced" if it has a different number of incoming and outcoming edges. If such a vertex is given, find another unbalanced vertex in the graph. Technically, Papadimitriou defines **PPAD** to be "the class of all search problems that can be reduced to EL". One might notice that all these problems admit a solution since, if a graph has an unbalanced vertex, then, at least one more must be present, therefore the existence of a solution is guaranteed. This is the very parity argument the whole complexity class relies upon.

Papadimitriou also proved that **PPAD** is a superset of **P** but also a subset of **NP**, therefore, unless **P=NP** (which most researchers consider unlikely), **PPAD** will also contain intractable problems. The connection of NE to **PPAD** follows.

In an elaborate and elegant, yet complex publication (completing a long series of preceding publications), Daskalakis, Papadimitriou and Goldberg proved that NE belongs in **PPAD** and that NE is **PPAD-**complete. This proof contains many important consequences worthy of consideration. However, before elaborating on these consequences, we must first discuss the proof. Of course, the mathematical and computer science methods and concepts implicated in the completeness proof are far beyond the scope of the present study, therefore we are forced to consider a simplified view. As we discussed before, the proof for NE relies on Brouwer's theorem. Now, among other proofs, a most important one relies on Sperner's famous lemma. In two dimensions, this lemma proves that for a triangular graph, for any triangulation and any 3-colouring of its nodes, there exists at least one triangle such that all its nodes are differently coloured. This result relates NE to a graph and, fortunately, if this graph is directed, there will be at least one (and consequently at least one more) unbalanced node.

Therefore, we can reduce NE to EL, effectively proving that NE∈**PPAD**. Now, reducing EL to NE is equivalent to transforming a directed graph to a game, which is much more demanding than connecting a game to a graph. Daskalakis, Papadimitriou and Goldberg (2006) start by connecting a directed graph to a Brouwer function, which they then relate to a game. The continuity of the Brouwer function, along with the necessity of it being a mixed strategy (and therefore relating a unit cube to a unit cube), suggest that distance arguments can be used to bound the graph and construct a function based on any 4-colouring (in three dimensions) we may choose for the graph. This suggests that an EL graph can be reduced to a Brouwer function. Since Brouwer functions also connect to EL through Sperner's lemma, we can infer that finding Brouwer fixpoints belongs in **PPAD**. Now, to relate the deduction of fixed points for a Brouwer function to NE, the writers make use of arithmetic circuits which can simulate arithmetic operations such as addition and multiplication. These circuits can be represented by 2-player and 3-player games which can then be condensed into a larger NE of a graphical game.

Eventually, it is proven that for every NE there exists an EL graph and for every EL graph there exists a NE. The consequences of this deduction are crucial for the importance of NE. Unless **P=PPAD**, which is considered unlikely, NE is a computationally intractable solution concept, which implies that attempting to use NE to solve games with large inputs will be a futile effort, since, regardless of the machines or technology we use, there will always exist an input size we will not be able to comprehend. We must be careful to remember that it is not the validity of the NE which is compromised but the length of the process we have to go through to find it, therefore, for small input games, the NE is still an extremely efficient equilibrium concept.

Furthermore, the connection to graphs, proven in the above theorem, allows for the utilisation of the powerful toolbox of graph theory results and concepts mathematicians have developed over the years in service of the search for an efficient algorithm for NE deduction.

As Daskalakis, Papadimitriou and Goldberg suggest, the search for approximate NE might alleviate the problem of intractability, however, we need to be able to arbitrarily approximate, which might be challenging to program.

4. Players' View of Complexity

The discussion around game complexity is quite intense, both in its game theoretic and its computer science aspects. However, there is a third, economic aspect. That is, the study of optimal response from the player's perspective.

Except for some sort of omniscient player who would know his best responses automatically, all other players are expected to go through a computation process to derive a best response. Under this view, computation capabilities can be treated as some sort of resource players must use to maximise the payoff they will receive. This approach to strategic decision making is not very well known since it lies within the intersection of two separate disciplines while it is also somewhat separate from the search for Nash equilibria.

4.1 Finite Automata

An interesting approach by Rubinstein depicts the existence of the resource of computation as the cost of strategic variety. In this model, Rubinstein studies the infinitely repeated prisoner's dilemma from a "mechanistic" perspective. In this approach players choose their optimal action at every round using a finite automaton. A finite automaton is a theoretical computational model such as the Turing machine, but of simpler function. More specifically, Rubinstein introduces a Moore machine for each player. Such an automaton can be described by a tuple of the form $M_i = (Q_i, q_i^0, g_i, m_i)$. Here, Q_i represents a (nonempty) finite set of states player i's machine can be in, q_i^0 represents the initial state of player i's machine, $g_i: Q_i \to A_i$ is a mapping translating states into actions and finally, $m_i: Q_i \times A_{-i} \to Q_i$ is a function assigning combinations of states and opponent actions into states. Formally, at every round, player i observes his opponent's action and, using the current state of his own machine, he produces a new state for the next round ($q_i^{t+1} = m_i(q_i^t, a_{-i}^t)$). This new state is then used to produce an optimal action for the next round ($a_i^{t+1} = g_i(q_i^{t+1}) = g_i(m_i(q_i^t, a_{-i}^t))$).

The function of the Moore machine is relatively simple. At every round, it uses inputs from the game (opponent's action) and previous games (current state of the machine), to produce a new state which is related to an optimal response for the next round. This model of strategic decision making has certain advantages. First of all, the optimality of the decision making is based on prior information of the game, effectively resolving the problem of infinite repetition. Additionally, the decision making is completely dependent on the choice made by each player at the first round, therefore, if the functions m and g are 1-1, we can find all the preceding outcomes of the game by back-tracking states to their corresponding actions and states.

Now let's turn to computational costs. The Moore machines, as described above, depend on the use of states. If the set of states of a Moore machine consisted of only one state (the initial one), then any strategic variability would directly result from the opponent's choice of actions. This implies that actions cannot be used to implement some sort of long-run strategy which may gather greater payoffs. On the other hand, a large number of states can provide the possibility of multiple strategies which may involve antagonization, cooperation, betrayal and punishment and more. Rubinstein suggests that the complexity of a Moore machine used by the players is measured by the number of states it contains. A realistic perception of complexity in this model would be the deposition of some sort of fee per state, needed in order to operate a complex (with respect to states) machine. As Rubinstein notes, a large number of states could inflict an overwhelming amount of fees which would possibly discourage the employment of extremely elaborate strategies. Consequently, we can infer that the cost imposed to the players by the complexity of the game is likely to affect their decision making.

This framework could possibly be generalised to n-player repeated games and to finite repetitions. Unfortunately, the use of finite automata is completely pointless for the study of complexity in static games, since automata function based on the change of states, which would take place only once. Additionally, there is no particular reason to restrict players' decision making on the use of finite automata.

4.2 The Economic Viewpoint

As we mentioned above, the complexity of strategic decision making can be perceived as some sort of resource constraint. Under this viewpoint, we could interpret any deviations from the normal predictions of game theory as cases where computational costs exceeded the potential benefit of thorough game analysis. The benefit-cost approach is definitely a tool of economic nature, totally irrelevant to the operation fees of the Moore machines discussed above.

However, it might be the proper instrument to examine how complexity affects strategic interaction.

At this point, we need to outline an important feature of this approach. Benefit to cost analysis becomes much simpler by assuming that the computational process players execute in search of an optimal action can be divided into two smaller parts. The pure strategy part and the mixed strategy part. The pure strategy part includes the search for optimal responses over the set of actions while the mixed strategy part refers to the search of an optimal response distribution.

The main reason for this division lies in the significant difference between the two search processes. The first process is likely to discern a specific action out of a discrete set while the second aims at deriving a distribution out of the $[0,1]$ continuum. Additionally, an elementary mixed strategy part can be completely set up by information the player already possesses, therefore we can substitute a best-response mixed strategy with a total randomization i.e., apply the uniform distribution.

This division implies that the player, when analysing the game, has two options. The player can carry out the pure strategy computations and then use the uniform distribution, or the player can carry out both the pure and the mixed strategy computations. The pure-strategy part would require a search algorithm similar (but not identical) to the ones we discussed above. However, the mixed strategy part would be significantly different. That is because there is no universal method for the derivation of an optimal response distribution. It is also reasonable to assume that the player will initially search for a pure-strategy best response and, if no such a response is found, he will proceed to the search for an optimal randomisation. Informally, we could say that the players follow the general "algorithm" described in Table 5 below to find their best responses:

BEST RESPONSE SEARCH process
Input: $G_i = (N, A, u_i)$
Search in Pure Strategies
If an Optimal Response id Found Then
Stop
Else
Search in Mixed Strategies
End_If

Table 5 - Best Response Search process

Additionally, we need to acknowledge that the "third-person" search for equilibria is significantly different to the "first-person" search for optimal responses. The search for

equilibria can be resolved by equating, for each player, the expected payoffs yielded by each available action and solving for the probability of every opponent action. On the other hand, searching for optimal responses from the player's perspective requires that we find the player's action which yields the greatest possible expected payoff. If more than one such actions exist, the player will have to randomise among them.

Now let us examine a candidate algorithm, presented in Table 5 below, for the mixed strategy part:

```
MIXED STRATEGY SEARCH algorithm
Input: G_i = (N, A, u_i), Π_i = {Pr(a_{-i}) ∈ [0,1] | a_{-i} ∈ A_{-i}}
Γ ← ∅
For a ∈ A_i
  EU(a) ← Σ_{a_{-i} ∈ A_{-i}} Pr(a_{-i}) u_i(a, a_{-i})
  Pr(a) ← 0
End_For
m ← b, b ∈ A_i
For a ∈ A_i \ {b}
  If EU(a) > EU(m) Then
    m ← a
  End_If
End_For
For a ∈ A_i
  If EU(a) = EU(m) Then
    Γ ← Γ ∪ {a}
  End_If
End_For
For a ∈ Γ
  Pr(a) ← 1/|Γ|
End_For
```

Table 6 - Mixed Strategy Search Algorithm

The above algorithm is not some generally optimal method for mixed strategy best response tracking but rather an example of how the process might look. As is evident, the algorithm depends on the prior knowledge of the probability each array of opponent actions has to occur. These probabilities are contained in the set Π_i which we considered to be an input. Obviously, each array is (assuming players make individual decisions) the product of the probability of each action in the array, which we do not write here to avoid notational confusion. The process followed by the algorithm is relatively simple. First, we assign each action the probability of 0. Then, we examine which actions maximise the expected payoff and isolate them in a set Γ. Finally, the player randomises among the actions of Γ. Intuitively, the use of the uniform distribution appears to properly assign probabilities in the elements of Γ, since each of them

yields the same payoff, therefore, we claim that each action will be played with the same probability $(1/|\Gamma|)$.

First of all, we can see that the fundamental rule of probability holds since Γ effectively divides the actions: $\sum_{a\in A_i} \Pr(a) = \sum_{a\in\Gamma} \Pr(a) + \sum_{a\in\Gamma^c} \Pr(a) = \sum_{a\in\Gamma} \frac{1}{|\Gamma|} + \sum_{a\in\Gamma^c} 0 = |\Gamma| \frac{1}{|\Gamma|} = 1.$ Additionally, a trivial lemma (see section 8.1) of game theory proves that the probability of choosing an action will only be non-zero if the expected payoff of the action is equal to the maximum expected payoff. Therefore, the process of isolating the maximising actions in a set and assigning a probability of 0 to every action out of the set is valid. Furthermore, it is possible that Γ and A coincide, if every action yields the same expected payoff. It is worth noting that we could modify the algorithm to include risk by, for example, making comparisons based on coefficients of variation or some other similar criterion. Regarding running time, we can see that it is of the form $O(|A_i|)$, since there are no nested loops.

The algorithm discussed above is not perfect, nor is it the only option in mixed strategy derivation. However, it is a decent approximation of what the process might look like. It is important to note that the same expected payoff would be extracted from every other mixed strategy as long as the probabilities of all the elements outside of Γ were zero. Notice that, for some distribution of the form $x(a) = \begin{cases} f(a)\ if\ a \in \Gamma \\ 0\ otherwise \end{cases}$, the total expected payoff of the player will be $\sum_{a\in A_i} x(a)EU(a) = \sum_{a\in\Gamma} f(a)EU(a) + \sum_{a\in\Gamma^c} 0 * EU(a) = m\sum_{a\in\Gamma} f(a) = m$, since, as we noted above, the sum of the non-zero probabilities must sum up to 1 and, by definition, all the elements of Γ yield the same expected payoff (which is the maximum expected payoff the player can achieve). The use of the uniform distribution comes as the naturally simplest choice, since no additional information or process is required to derive it.

We know that its running time is less than that of a pure strategy search since the latter would require comparisons which can only be made by nested loops which would augment the worst-case running time to at least quadratic. Additionally, we assumed that the player "knows" the probability of each array of opponent actions taking place, which is a significant but also justified simplification, since such information could be considered situational. One could argue that the player will attempt to formulate these probabilities through considering payoffs and the use of strategic reasoning but taking this into consideration will cause unnecessary complications which are beyond our discussion.

4.3 Asking the Natural Question

With the preceding discussion, a standard, for an economist, question comes to mind: is the derivation of a mixed strategy worth the additional cost? It may be the case that the deduction of which actions belong in the set Γ and the proper setting of probabilities do not lead to an increase in payoff, such that the additional running time is compensated for. A common economic tool we can use to examine this case is the benefit to cost ratio which generally examines how beneficial an investment is with respect to the resources its execution requires. This suggests that we need the running time to inflict some sort of expense the player must make to carry out computations. In order to avoid algebraic complications and notational confusion, we will simply refer to the cost of the pure strategy search as C_p and the cost of the mixed strategy search as C_m, where $C_p > C_m$ since, as is mentioned above, the pure strategy search contains more nested loops (due to more comparisons). In our comparison, we need to consider the case where an optimal response is not found in pure strategies as, in the opposite case, the search would have already been terminated. Since the player knows opponent arrays' probabilities, he can either directly apply the uniform distribution to all his available actions and derive a randomisation where every action may be chosen with probability $\frac{1}{|A|}$ (notice that we drop the player's indicator i since we will not need it to discern between players), or he can proceed to the derivation of the set Γ, where every element may be played with probability $\frac{1}{|\Gamma|}$ and every other action will not be chosen. It is important to note that, as the algorithm implies, every action in Γ will yield the same expected payoff.

Let us now form the comparison. The player's benefit to cost ratio will simply be the expected payoff the player will receive in each case, divided by the corresponding cost. In the total randomisation case, where the distribution $\Pr(a) = \frac{1}{|A|}\ \forall a \in A$ is used, the player's benefit to cost ratio is: $T = \frac{\sum_{a \in A} \frac{EU(a)}{|A|}}{C_p}$. In the mixed strategy randomisation case, where the distribution $\Pr(a) = \begin{cases} \frac{1}{|\Gamma|}, & a \in \Gamma \\ 0, & a \in \Gamma^c \end{cases}$ is used, the benefit to cost ratio is: $M = \frac{\sum_{a \in \Gamma} \frac{EU(a)}{|\Gamma|}}{C_p + C_m} = \frac{\frac{v|\Gamma|}{|\Gamma|}}{C_p + C_m} = \frac{v}{C_p + C_m}$.

Note that $v = max_{a \in A} EU(a)$ and every element in Γ yields v by definition. The question stated above can be expressed in mathematical terms: Does there exist any combination of inputs such that $T > M$?

We can immediately discern a special case for which this inequality holds. It is the case where every action yields the same expected payoff (which inevitably is the maximum payoff v) where $T = \frac{\sum_{a\in A}\frac{EU(a)}{|A|}}{C_p} = \frac{\frac{u|A|}{|A|}}{C_p} = \frac{v}{C_p}$. We know that $C_m > 0 \Leftrightarrow C_p + C_m > C_p \Leftrightarrow \frac{v}{C_p} > \frac{v}{C_p+C_m}$ $\Leftrightarrow T > M$. Despite the fact that this is an extreme case of expected payoff formation, we can intuitively speculate that a lower variability may make the search for mixed strategies less beneficial.

Let us now attempt to derive a general condition for which the inequality $T > M$ holds. Examining the ratio of the total randomisation, we can see that the sum of all the expected payoffs can be simplified: $\sum_{a\in A} EU(a) = \sum_{a\in \Gamma} EU(a) + \sum_{a\in \Gamma^c} EU(a)$. Let us denote the arithmetic mean of the elements outside of Γ as w (i.e., $w = \sum_{a\in \Gamma^c}\frac{EU(a)}{|\Gamma^c|}$). We can express the sum of expected payoffs of actions not in Γ as $\sum_{a\in \Gamma^c} EU(a) = w|\Gamma^c|$, therefore, our initial sum can be written as: $\sum_{a\in A} EU(a) = |\Gamma|v + |\Gamma^c|w$ (remember that every payoff in Γ yields v). By definition, $|\Gamma^c| = |A| - |\Gamma|$ so, dividing the sum by $|A|$, we get: $\sum_{a\in A}\frac{EU(a)}{|A|} = \frac{|\Gamma|}{|A|}v + (1 - \frac{|\Gamma|}{|A|})w$. To simplify notation, we will denote the ratio $\frac{|\Gamma|}{|A|}$ as g and, obviously, $g \leq 1$. Returning to the inequality, we see that $\frac{gv+(1-g)w}{C_p} > \frac{v}{C_P+C_m} \Leftrightarrow (1 + \frac{C_m}{C_p})(gv + (1-g)w > v$. Again, let us simplify by expressing $\frac{C_m}{C_p}$ as c, where $c \ll 1$ since the pure strategy search has significantly more nested loops than the mixed strategy search. Consequently, our expression can be written as $(1 + c)(gv + (1 - g)w) > v \Leftrightarrow w > \frac{1-g(1+c)}{(1-g)(1+c)}v$. Now, this expression is much more manageable. Regarding $\frac{1-g(1+c)}{(1-g)(1+c)}$, it is obviously smaller than 1 but is only positive when $g < \frac{1}{1+c}$.

Eventually, the derived conditions that need to be satisfied in order for mixed strategy search to be less beneficial than total randomisation are $w > \frac{1-g(1+c)}{(1-g)(1+c)}v$ and $g < \frac{1}{1+c}$. The first condition suggests that total randomisation will be preferable to mixed search if and only if the average expected payoff of the non-maximising actions is greater than the maximum expected payoff scaled by a factor of $\frac{1-g(1+c)}{(1-g)(1+c)}$ which depends on the details of the game.

We could call this result "Strategic Behaviour Aversion" (SBA) as it suggests that behaving strategically, i.e., deriving a probability distribution to optimally respond, is (in certain cases) less beneficial (and therefore avoided) to the comprehension of the situation as a decision problem under uncertainty. SBA suggests that less planning might be optimal. This could possibly be an explanation for many of the cases where an economic agent does not behave in accord to the predictions of game theory. After conducting both search processes once, the player will have all the information needed to test the conditions derived above. If both conditions are met, the player will shift to only pure strategy search in any future occurrences of similar games. Of course, in the real world, economic agents would probably reach a similar criterion and mindset after multiple experiences where mixed strategy search might have "felt" unrewarding.

The second condition must necessarily hold for the factor of the first condition to be nonnegative, which is vital for the first condition to even be worth examining. Unfortunately, given that the running time of the pure strategy search is probably exponential (see section 8.2), while that of the mixed strategy search is linear, the value of c will permit for a very small number of games for which both the conditions hold (as we mentioned above, $c \ll 1$). The second condition is more likely to hold in games where g is incredibly small. Such games would be those with a large action set and a very small number of maximising actions. This restriction critically decreases the applicability of SBA as a concept, since a very small class of games could match the requirements of the second condition, while there is virtually no guarantee for the first condition.

4.4 Refining the "Strategic Behaviour Aversion" Concept

The low applicability of our result can be resolved by dropping the assumption that the probability distributions over opponent players' actions are exogenously given. Despite being convenient, this assumption is relatively unconvincing. The rational assumption is that, assuming all the information players use in their decision making is that of the game, they formulate their expectations regarding opponent actions by processing opponents' payoffs in different situations. This suggests that, in the case where only pure strategy search is applied, the expected payoff will simply be $\sum_{a \in A} \frac{u_i(a)}{|A|}$ where A here stands for the cartesian product of every player's action set ($A = \times_{n \in N} A_n$). The player has to add $|A| = |A_1| \times |A_2| \times \ldots \times |A_N|$ elements, each of which may occur with probability $1/|A|$, which is the only distribution the player can apply without additional data, both to predict opponents' actions and to randomise

over his own. Obviously, the probability mass function $1/|A|$ is a joint probability distribution of independent random variables ($\Pr(a) = \frac{1}{|A|} = \frac{1}{\prod_{n\in N}|A_n|} = \prod_{n\in N}\frac{1}{|A_n|} = \prod_{n\in N}\Pr(a_n)$).

The right-hand side of the inequality $T > M$ will remain the same, but with a crucial difference. Now, C_m will be much larger. That is, because, as is claimed above, the player derives probability distributions for opponent actions based on game information, including their payoffs, therefore a search over A_{-i} increases the search running time to at least exponential, making the cost for the mixed strategies search almost as expensive as that of the pure strategy search (if not greater).

The total randomisation benefit to cost ratio transforms to $T = (\sum_{a\in A}\frac{u_i(a)}{|A|})/C_P$ while the mixed strategy search ratio remains the same. Just as a reminder, technically, the payoff in the case of the mixed strategy search is $v = \sum_{a\in A} x(a)u_i(a)$, but here, A is the total action set and $x(a)$ is the joint probability distribution of an array of actions, produced by multiplying the (independent) distributions over every player's action set, derived by the mixed strategy search (formally, $x(a) = \prod_{n\in N} x_n(a_n)$). Slightly manipulating the inequality $T > M$ leaves us with an elegant and significant (yet computationally "blunt") result which encapsulates the essence of the Strategic Behaviour Aversion. Notice that the expected payoff in the total search case is exactly the same as the arithmetic mean of the player's payoff. Then we can simply write our expression as $\bar{u}_i > \frac{v_i}{1+c}$, where $\bar{u}_i = \sum_{a\in A}\frac{u_i(a)}{|A|}$.

This relation, despite being much more simplistic than the one previously derived, makes a very important statement: When the average payoff of a player is greater than the maximum expected payoff scaled by a factor of $1/(1+c)$, then it is in the player's best interest not to proceed in deriving an optimal probability distribution over his actions but to simply apply a randomisation using the fundamental information of the game (the set A). Additionally, both the pure and the mixed strategy search are now similar in running time and, since the lower order terms and constants are neglected when measuring running time, we can simplify the inequality presented above by claiming that $C_p \approx C_m$, therefore, we can approximate the condition for SBA as: $\bar{u}_i > v_i/2$.

Again, this condition will arise out of experience. If the player frequently receives a maximum expected payoff smaller than $2\bar{u}_i$, it is likely that, intuitively, his reaction will be the aversion of strategic behaviour (derivation of a distribution). Maybe, this concept can explain why, in

certain cases, players deviate from their optimal responses, as those are predicted by game-theoretic results. However, since deviation from the predictions of game theory is not always the case, it is likely that players could gradually categorise games in three classes with respect to their payoff structures: a class of games for which pure strategy best responses exist, a class of games worthy of strategic behaviour and a class of games for which total randomisation is preferable. Studying the payoff structures that match each class could possibly help us predict in which cases economic agents are to behave sub optimally.

5. Literature Review

A fundamental topic of our discussion is the complexity of the Nash equilibrium, and more specifically, the fact that it belongs to a complexity class of probably intractable problems (**PPAD**). The proof that NE∈**PPAD** by Daskalakis, Papadimitriou and Goldberg (2006), depends on a reduction of NE to the End of Line problem, which is the core problem of the **PPAD** class, and a reduction of the End of Line to a NE. The difficulty of the proof is apparent from the large number of ideas and concepts necessarily introduced in order to connect a graph problem to an N-person game. In a similar manner, Goldberg and Papadimitriou (2005) have previously proven the result for three player games, while Chen, Deng and Teng (2006a) constructed a simpler proof for bimatrix games, using similar techniques to those used for proving the N-player case.

In view of the intractability of NE, the concept of approximate NE (or ε-NE) has become quite appealing to most researchers. Chen, Deng and Teng (2006b) have proven that, even for approximate NE (where the sufficient condition for equilibria is relatively relaxed), symmetric two-player games cannot be solved in polynomial time. Kontogiannis, Panagopoulou and Spirakis (2006) also tackle the tractability of approximate NE in two-player games and manage to generate a polynomial time algorithm for finding ε-NE for a specific constant value of approximation (ε). A similar result is independently produced by Daskalakis, Mehta and Papadimitriou (2006). An interesting approach by Kontogiannis and Spirakis (2007) studies approximate NE tractability of search for equilibrium supports with a focus on binary payoff games (win-lose games), for which they construct an algorithm that tracks 0.5-NE in polynomial time. In a later publication, Kontogiannis and Spirakis (2010) prove the limitation of the size of approximate NE supports for normalised games (where payoffs are transformed to belong in [0,1])

The tracing of equilibria is also central in our discussion. Work by Lemke and Howson (1964) shows that NE in bimatrix games can be traced by a sequence of algebraic operations over the game matrices. Unfortunately, the process requires exponential running time in its worst-case scenario. Another algorithm, proposed by Lipton, Markakis and Mehta (2003) is shown to require quasi-polynomial running time to produce approximate NE, while an upper limit for equilibrium support derivation is also proved in the same publication. In the case of games with large population (where it is possible that not all players' payoffs are directly dependent), an algorithm proposed by Kearns and Mansour (2002) can find approximate NE through a learning process in polynomial time.

6. Discussion

As we established in 1.1, computational complexity is a significant obstacle to the use of game theory. A large running time can "compromise" even our most praised and esteemed concepts, such as the Nash equilibrium. Fortunately, as proven in our discussion so far, the use of computer science in service of the study of game theory can be beneficial, both in terms of problem "diagnosis" and in terms of innovative solutions. Despite its difficult concepts and techniques, computer science can help us make deductions of critical importance such as the intractability of the Nash Equilibrium (unless **PPAD⊆P**, which is considered unlikely) as well as allow us to develop processes to derive NE. Additionally, by borrowing tools from the theory of complexity, such as the complexity measurement techniques, we can compare these processes and possibly find better ones. It is also possible to utilise the mathematics of computer science and introduce methods of combinatorics and graph theory in game theory to create different expression formats and game solving methods. It is worth noting that the problem of intractability in games (and generally) is not relevant to the technological means we use to conduct computations. It is the case that for every problem in **NP**, there will always exist a level of input beyond which any machine will be overwhelmed. The same probably holds for problems in **PPAD** which usually are super-polynomial in running time.

Another implication of complexity we avoided to discuss is that of space complexity. Solving a problem requires "space" (data storage capabilities) which, like time, is an essential resource in problem solving. In order to discuss the complexity of NE (possibly the most important application of computer science in game theory) we had to focus on time complexity, however it is important to note that large games could also be demanding in terms of space. Relatively

accurate calculations suggest that a 10x10 game of Hex would require the machine that carries out its computation to be larger than $1km^3$. Of course, that is, in the case where we do not use advanced methods such as dynamic programming to find best responses.

In the latter part of chapter 4 an interesting concept was developed. We made use of the concept of computation itself to attempt to approximate the process players (possibly) go through to deduce their optimal responses, which, in turn, revealed a condition ($\bar{u}_i > v_i/(1+c)$) that possibly explains how deviations from the predictions of game theory may be beneficial (and therefore observed in the real world). The concept of Strategic Behaviour Aversion suggests that, if no unique pure strategy best response is found, players may choose the simplest randomisation over their options rather than derive an optimal randomisation, if the second choice is unlikely to yield a large enough payoff.

The condition derived (as well as the process of derivation) appears relatively convincing, however many aspects of the condition and the concept itself must be carefully examined before we can establish SBA as a valid result for non-cooperative games. Maybe, at least until more work is done on the subject, we should use the term Strategic Behaviour Aversion Hypothesis (SBAH). It is the case that we assumed players will develop an understanding of the "performance" of each option (total randomisation or mixed strategy search) based on the benefit to cost ratio which is not the only indicator of the relation between costs and benefits.

Additionally, the nature of the cost of running time (from the players' perspective) is mostly unknown, except for the reasonable assumption that it is increasing with respect to running time. The complete specification of a cost function for running time would probably require a more behavioural approach as it would implicate the cognitive perception of running time players have. Furthermore, we are clueless as to how the probability distributions over opponent players' actions are derived. In 4.3 we speculated that the process involves search over all the possible arrays of opponent actions which will certainly raise the running time from linear to exponential, however it is not impossible that the running time is even larger. In this case, the value of the factor $1/(1+c)$ will make the condition for SBAH even easier to satisfy.

Despite the aforementioned complications, the concept of Strategic Behaviour Aversion could prove to be an important tool in understanding the behaviour of economic agents and the structure of the relation between computations and individual optimal responses. Therefore extensive research is needed to properly develop and extend this concept, as well as to make use of its applications.

7. References

[1] Arora S. and Barak B., *Computational Complexity Theory, a Modern Approach*, Cambridge University Press, 2009

[2] Chen X. and Deng X., *3-NASH is PPAD Complete*, Electronic Colloquium on Computational Complexity, 2005

[3] Chen X. and Deng X. and Teng S., *Settling the Complexity of Computing Two-Player Nash Equilibria*, Proceedings of the 47th Annual Symposium on Foundations of Computer Science, 2006a

[4] Chen X. and Deng X. and Teng S., *Computing Nash Equilibria: Approximation and Smoothed Complexity*, Proceedings of the 47th Annual Symposium on Foundations of Computer Science, 2006b

[5] Chen X. and Deng X., *On Algorithms for Discrete and Approximate Brouwer Fixed Points*, Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005

[6] Conitzer V. and Sandholm T., *Complexity Results about Nash Equilibria*, Proceedings of the 18th International Joint Conference on Artificial Intelligence, 2003

[7] Daskalakis C., Papadimitriou C. and Goldberg P., *The Complexity of Computing a Nash Equilibrium*, SIAM Journal of Computing, 2006

[8] Daskalakis C., *On the Complexity of Approximating a Nash Equilibrium*, ACM Transactions on Algorithms, 2013

[9] Daskalakis C. and Papadimitriou C., *Three-Player Games are Hard*, Electronic Colloquium on Computational Complexity, 2005

[10] Daskalakis C. and Papadimitriou C., *Computing Equilibria in Anonymous Games*, 48th Annual IEEE Symposium on Foundations of Computer Science, 2007

[11] Daskalakis C., Mehta A. and Papadimitriou C., *A Note on Approximate Nash Equilibria*, 2nd Workshop on Internet and Network Economics, 2006

[12] Daskalakis C. and Demaine E., *The Complexity of Hex and the Jordan Curve Theorem*, 43rd International Colloquium on Automata, Languages, and Programming, 2016

[13] Fiat A. and Papadimitriou C., *When the Players Are Not Expectation Maximisers*, International Symposium on Algorithmic Game Theory, 2010

[14] Fabrikant A., Papadimitriou C. and Talwar K., *The Complexity of Pure Nash Equilibria*, Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, 2004

[15] Goldberg P. and Papadimitriou C., *Reducibility Among Equilibrium Problems*, Electronic Colloquium on Computational Complexity, 2005

[16] Halpern J. and Pass R., *Algorithmic Rationality: Game Theory with Costly Computation*, Journal of Economic Theory, 2015

[17] Hirsch M., Papadimitriou C. and Vavasis S., *Exponential Lower Bounds for Finding Fixed Points*, Journal of Complexity, 1989

[18] Kontogiannis S., Panagopoulou P., Spirakis P., *Polynomial Algorithms for Approximating Nash Equilibria of Bimatrix Games*, 2nd Workshop on Internet and Network Economics, 2006

[19] Kontogiannis S. and Spirakis P., *Efficient Algorithms for Constant Well Supported Approximate Equilibria in Bimatrix Games*, International Colloquium in Automata, Languages and Programming, 2007

[20] Kontogiannis S. and Spirakis P., *Well Supported Approximate Equilibria in Bimatrix Games: A Graph Theoretic Approach*, Algorithmica, 2010

[21] Kearns M. and Mansour Y., *Efficient Nash Computation in Large Population Games with Bounded Influence*. Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, 2002

[22] Kearns M., Littman M., Singh S., *Graphical Models for Game Theory*, Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence, 2001

[23] Kalai E., *Bounded Rationality and Strategic Complexity in Repeated Games*, Game Theory and Applications. San Diego: Academic Press, 1990

[24] Lemke C. E. and Howson J. T. Jr, *Equilibrium Points in Bimatrix Games*, Journal of the Society for Industrial and Applied Mathematics, 1964

[25] Lipton, R., Markakis, E., Mehta, A., *Playing Large Games Using Simple Strategies*, Proceedings of the 4th ACM Conference on Electronic Commerce, 2003

[26] Maarup T., *Hex – Everything You Always Wanted to Know About Hex But Were Afraid to Ask*, Masters Thesis, University of Southern Denmark, 2005

[27] Myerson R., *Game Theory Analysis of Conflict*, Harvard University Press, 2013

[28] Noam N., Roughgarden T., Tardos E. and Vazirani V., *Algorithmic Game Theory*, Cambridge University Press, 2007

[29] Porter R., Nudelman E. and Shoham Y., *Simple Search Methods for Finding a Nash Equilibrium*, Games and Economic Behaviour, 2008

[30] Papadimitriou C. and Yannakakis M., *On Complexity as Bounded Rationality*, 26th ACM Symposium on Theory of Computing, 1994

[31] Papadimitriou C. and Yannakakis M., *Optimization, Approximation and Complexity Classes*, Journal of Computer and System Sciences, 1991

[32] Papadimitriou C., *Computational Complexity*, Pearson, 1993

[33] Papadimitriou C., *On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence*, Journal of Computer and System Sciences, 1994

[34] Rubinstein A., *Finite Automata Play the Repeated Prisoner's Dilemma*, Journal of Economic Theory, 1986

[35] Robinson A. and Goldman A., *The Set Coincidence Game: Complexity, Attainability, and Symmetric Strategies*, Journal of Computer and System Sciences, 1989

[36] Von Neumann J., Morgenstern O., *Theory of Games and Economic Behavior*, Princeton University Press, 1953

8. Appendix

8.1 Best Response Condition Lemma

The lemma mentioned in section 4.2 proves that, when optimising his expected payoff, a player must choose a randomisation over his actions such that, only the actions which yield the maximum expected payoff may be assigned non-zero probabilities. We avoid any indicator of the specific player for simplification. Here, by A, we denote the set of actions available to the specific player and not the cartesian product of all the players' actions.

Let $x(a), EU(a): A \to \mathbb{R}$, where $x(a) \geq 0 \; \forall a \in A$ is the probability assigned by the player to the action a and $EU(a)$ is the expected payoff from choosing the action a. Let us denote the maximum expected payoff from individual actions as $m = max_{a \in A} EU(a)$. The total expected payoff the player will receive with the randomisation x will be: $\sum_{a \in A} x(a) EU(a) = \sum_{a \in A} x(a)[m - (m - EU(a))] = m - \sum_{a \in A} x(a)[m - EU(a)] \leq m$, since $\sum_{a \in A} x(a) = 1$ and, by definition, $x(a) \geq 0$ and $m - EU(a) \geq 0$. The quantity $\sum_{a \in A} x(a)[m - EU(a)]$ must be greater than, or equal to 0, therefore the total expected payoff of the player will be maximised only if $\sum_{a \in A} x(a)[m - EU(a)] = 0$. Now, since both $x(a)$ and $[m - EU(a)]$ are greater than, or equal to 0, the quantity $x(a)[m - EU(a)]$ must be equal to 0 for every $a \in A$.

Now, whenever $EU(a) = m$, necessarily, $x(a)[m - EU(a)] = 0$, therefore $x(a) \geq 0$, but when $EU(a) \leq m$, $x(a)[m - EU(a)] = 0$ if and only if $x(a) = 0$. Consequently, it must be the case that, whenever an action does not yield the maximum expected payoff, the probability assigned to it must be 0, assuming that the player maximises his total expected payoff. Now, this lemma does not guarantee that all the probabilities of the maximum payoff-yielding actions are non-zero, however, we know that at least one of them must be, in order for $\sum_{a \in A} x(a)$ to equal 1.

8.2 Pure Strategy Search Algorithm

In section 4.2, the player's search for an optimal response is divided into two smaller parts. The first part is the pure strategy search and the second is the mixed strategy search. In 4.2, an algorithm for the latter part is presented in order to explain, to a certain extent, what the process of deriving mixed strategies may be. The discussion of a pure strategy search algorithm is avoided as its exact nature is irrelevant to the topic of whether Strategic Behaviour Aversion is beneficial, which the discussion of 4.2 builds the foundation for. However, claims regarding the pure search running time are made, therefore we need to provide a candidate algorithm to justify them. Hence, an example pure strategy search algorithm follows:

```
PURE STRATEGY SEARCH algorithm
Input: G_i = (N, A, u_i)
x ← 0
For a ∈ A_i
 I(a) ← 0
End_For
While x = 0 AND y ∈ A_-i
 t ← b
 For a ∈ A_i\{b}
  If u_i(a, y) > u_i(t, y) Then
   t ← a
  End_If
 End_For
 I(t) ← I(t) + 1
 If I(t) = |A_-i| Then
  x ← 1
  a* ← t
 End_If
End_For
If x = 0 Then
 ∄ pure-strategy best response
Else
 a* is the player's best response
End_If
```

Table 7 - Pure Strategy Search Algorithm

Evidently, the nature of the algorithm presented above is similar to that of the mixed strategy search, which we discussed in detail in section 4.2. However, here, the second outer "While" loop will make $|A_{-i}|$ repetitions. For every instance of opponent arrays, an action b is arbitrarily chosen and every other action of the player is tested for optimality against b. If a specific action gathers "optimality points" ($I(a)$) equal to the total number of opponent arrays ($|A_{-i}|$), then it is optimal in every occasion and consequently, it is designated as an optimal response. Finding all the best responses would require additional lines of code, therefore this algorithm is targeted to that best response which yields the highest payoff in every instance (assuming best-response actions exist).

Regarding the running time, the worst-case scenario requires a running time of $O(A)$ (of course, we neglect all the lower order terms of A_i which would be in $T(n)$). This implies that each player would require exponential time to deduce a best response, therefore, this algorithm is computationally inefficient for players (as is the majority of best response search algorithms proposed by researchers). Nevertheless, in addition to the assumption that every computation executed by the player requires a certain constant cost, this proves the claim made in 4.2, regarding the exponentiality of the pure strategy search cost.