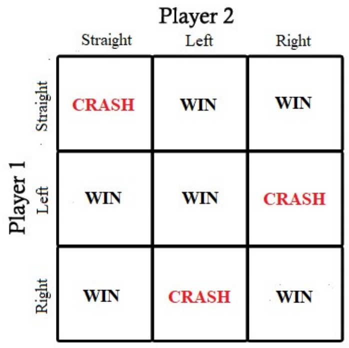
Game-Theoretic Scheduling in the Context of High-Level Synthesis

Gordan Konevski
gordan.konevski@stud.hshl.de

Hochschule Hamm-Lippstadt 2022

***Abstract*—In the context of computation scheduling, a number of different approaches and methods have been developed, all with their unique uses and applications, typically designed to bring solutions to specified problems. It can be presumed that any method (that deals with scheduling) relies on specified - or (automatically) specifiable - tasks; an approach to execute said tasks; and on a system that is able to verify the results of each task and of the method as a whole. Game Theory, uniquely presents a method that would rely on task executors which specifically have limited independent agency, can combine a multitude of additional methods, and can, additionally, test their efficiency in a environment where the task executors would compete with each other, in order to find the most optimal solution to the specified problem at hand. But there is more to unpack, as the actual versatility of this method and its ”ease-of-use” have to be questioned and delineated, in order to substantiate the credibility of Game Theory as a viable scheduling method in the context of High-Level Synthesis.**



		Player 2		
		Straight	Left	Right
Player 1	Straight	CRASH	WIN	WIN
	Left	WIN	WIN	CRASH
	Right	WIN	CRASH	WIN

Fig. 1. Game Theory simple example [1]

I. Introduction

John von Neumann and Oskar Morgenstern coined the term ”Game Theory” in 1944 when they published their work entitled ”Theory of Games and Economic Behavior.” It offers helpful mathematical tools for understanding the potential strategies that individuals may take when competing against one another in games or working together with other players. Since that time, people in fields as diverse as political science, economics, international relations, biology, philosophy, engineering, and computer science have discovered game theory to be applicable to their fields. Therefore, it can clearly be seen that game theory has an enormous scope of applicable ares and the same is true for scheduling algorithms.

II. Scope of Game Theory

A. Game Theory Characteristics and Research

Currently, monkeys are used in the majority of nerve science and game theory experiments, which is less common in human-to-human comparisons. Game theory is concerned with the study of social group decision-making. [1] A game is characterized by a collection of options or strategies available to each player, as well as a payoff matrix that determines the outcome (utility) for each player based on all players' choices. Nash Equilibrium is a term used in game theory to describe a situation in which players have no reason to change their original strategy when the best outcome of a game is one, i.e. a concept in game theory that suggests the best outcome of a game is when no player has a reason to change his strategy after thinking about his opponent's choice. Unfortunately, there are theoretical and practical limitations to this vital concept. D. Lee et al. investigated the decisions monkeys make during a binary zero-sum game known as matching pennies in their investigations [2]. They demonstrated that the strategies of the animal's opponent can change the animal's behavior by training monkeys to play such a competitive game against a computer opponent. They studied monkeys' decision-making in the rock-paper-scissors game, a simple zero-sum game with ternary alternatives. Two factors prompted this decision. For starters, systematic comparative studies of choice behavior in non-human primates have the potential to shed light on the evolutionary origins of human decision-making. Second, such monkey decision-making models could help researchers better understand the neurological mechanisms that underpin human decision-making.

B. Non-cooperative Game

The players are rational in the game. Therefore, the players' best responses are developed based on the expected payoff. So, each player has selected the appropriate strategy based on their anticipation about other players' best responses and strategy profiles. [4]

C. Cooperative Game

When playing this type of game, players are required to sign a legal contract in order to collaborate with one another and establish a coalition. The engagement of all of the players in the coalition will eventually result in the formation of a

”grand coalition”. [4] A distinctive function is generated by the value of coalitions in the game using a matrix form, and this function is associated with the game type.

As a result, the findings of the research have led to the formation of a set of linear equations that are based on the requirements for the characteristic function and the core stability. Based on the conditions for stability, the grand coalition reward has been demonstrated to offer a superior payoff to that of another coalition payoff for all players. Therefore, the core is stable as the equilibrium point in the cooperative game type since the prerequisites for core stability in a grand coalition have been met. [4]

D. Evolutionary Game Theory

The term ”evolutionary game theory” refers to a subfield of game theory that was developed in the 1970s by several evolutionary biologists. This subfield is bringing out new insights from the original theory. This same evolutionary game theory has the benefit of eliminating the primary problem of the conventional game theory, which is the characterisation of rational behavior as well as the requirement to anticipate the behaviors of other players. This is a significant benefit of the evolutionary game theory. In practical terms, at the very beginning of conventional game theory, it is encouraged to focus on the rational behavior of players in interactions. This means that players should be aware of the conditions they are in and should be thinking of ways to enhance their payoff. The idea of rationality cannot be clearly defined, mostly due to overwhelming challenges brought into the game by the concepts of trust, available information, and subjectivity of the participants. Therefore, any concept of strategic decision and anticipation, and by extension, any notion of rationality, is completely disregarded in evolutionary games. A game that investigates the development of interaction-dependent strategies in populations is referred to as an evolutionary game. One of the most important ideas to keep in mind when playing evolutionary games is the evolutionary stable strategy, also known as ESS. An ESS is defined as a strategy that holds the following premises: a population in which all players are playing this strategy is impervious to incursion by a small group of strains who play a different strategy. If players are only allowed to use pure strategies, then there is no such thing as an ESS, but if players are allowed to utilize mixed strategies, then there is a genuine ESS. It has been mentioned that in a hybrid model, a player gives a probability to each natural approach, and then uses a randomization mechanism to pick which strategy to play. This is how the hybrid strategy works. It is fairly likely that the chance of an individual adopting a particular strategy is proportional to the expected return that the population can anticipate from using that strategy. As a result, the equation for implementing the technique can be stated as in Fig 2 (algorithm by R. Fisher) [5]. Where the numerator $p_i^t F_i(p)$ is the expected payoff of a player using strategy i, and the denominator $\sum_{i=1}^{n} p_i^t F_i(p)$ is the total expected payoff of a player using different strategies, which

$$p_i^{t+1} = \frac{p_i^t F_i(p)}{\sum_{i=1}^{n} p_i^t F_i(p)}$$

Fig. 2. R. Fisher’s Algorithm [5]

is the normalization term that ensures $\sum_{i=1}^{n} p_i^{t+1} F_i(p) = 1$. [5]

III. High-Level (i.e. Behavioral) Synthesis

In the early 1990s, (Very Large Scale Integration) VLSI technology was able to achieve densities of more than one million transistors of random logic on a single chip. When designing such complicated systems, it is very difficult to do it by handcrafting each transistor or by defining each signal in terms of logic gates. When there is a rise in the complexity of systems, there will be a corresponding increase in the demand for design automation on more abstract levels, which are levels at which functionality and tradeoffs are easier to comprehend. Additionally, very large scale integration (VLSI) technology has reached a maturity level, meaning that it is well recognized and no longer offers a competitive advantage on its own. In order to boost overall productivity and get a leg up on the competition, the industry has begun taking a more holistic approach to analyzing the product development cycle. In the 1990s, one of the most important design goals was to fully automate the entire process, from the initial conception to the fabrication of the silicon. Since quite some time ago, the reality for those who design hardware has been marked by increased design complexity as well as shorter design cycles. In the early 1990s, the development of RTL synthesis tools allowed designers to shift away from the complexities of technology-specific schematics and instead define features and functions at a quite abstract, technology-independent plane using different languages like VHDL and Verilog. The design team for each given design has access to a wide variety of different hardware architectures from which they can choose to accomplish their objectives. Finding a good architecture involves a tremendous amount of effort from even the most skilled designer. When just logic synthesis techniques are used, an RTL description of the architecture must be written in a hardware description language (HDL), which often consists of reams and reams of custom code. The designer is responsible for specifying all of the specifics in the code, such as the logic states and the operations that are carried out according to those states. A typical design will include a predetermined number of data path components (e.g. multipliers and adders). The utilization of these resources is controlled by a finite state machine, and intermediate values are either saved in a memory or in registers. Every one of these components is given a clear definition within the context of an RTL description. When an RTL specification is handled with an RTL synthesis tool, there is a reduction in the amount of design space available

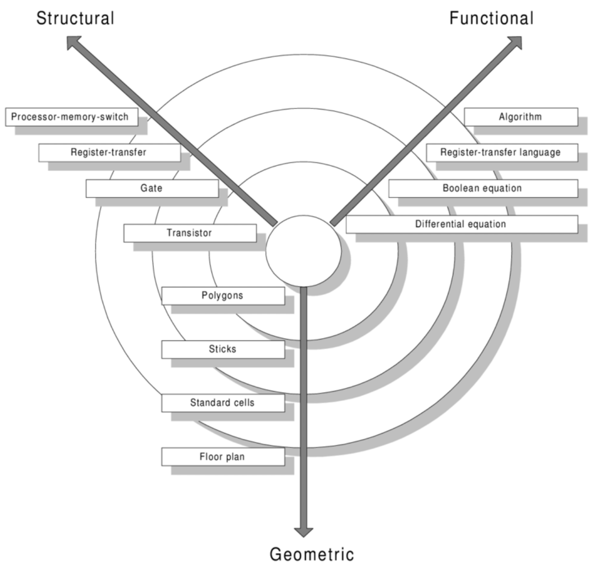


Fig. 3. VLSI Design

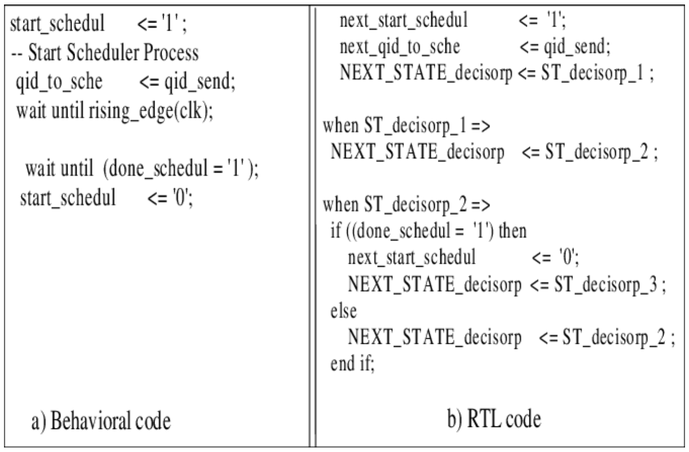
for exploration. The designer is allowed to make a few minor concessions in terms of either delay or area, but the underlying architecture of the system must always be explicitly defined in the code. It is necessary to rewrite the RTL description in order to look at the area-delay curve in a different part of the graph. Rewriting the RTL code is a time-consuming procedure that is prone to introducing errors. There is no assurance that the revised version will be of higher quality, and the rewrite could take many weeks to finish.

A. Behavioral Approach Compared to an RTL Approach

A behavioral design approach, as opposed to something like an RTL-based design method, changes the emphasis from the specifics to the general behavior of the design. This would be in contrast to an RTL-based design method. Teams working on designs have become able to make decisions about those designs on the most abstract, computational level. A behavioral description outlines the algorithm to be executed with little or no technical specifics. Because of this, behavioral descriptions are generally five to ten times shorter than their comparable RTL explanations. By adjusting the parameters of the constraints, designers lead the behavioral synthesis tools to produce alternative architectural layouts (such as number and type of data path elements, clock period, and desired number of clock cycles). Tools for behavioral synthesis begin with a description of the desired behavior and then automatically arrange the necessary operations into the appropriate clock cycles. The final result is RTL code, which describes the memory or registers, the data path elements, and the finite state machine. This RTL code is complete and ready to be used in logic synthesis. These RTL representations can be optimized in a level that is comparable to that of a hand-written RTL description; however, the architectures that serve as the baselines for these optimizations are very different from one another. The early stages of the design process are an ideal time for designers to use behavioral synthesis to rapidly examine a wide variety of potential designs. After the algorithm has been finished, the behavioral specification that was created is then used as an input in the tool that develops the behavior. The designer is able to rapidly analyze a variety of designs by adjusting the design constraints, including the clock period, the timing of I/O processes, the number and type of data path elements, and the desired number of clock cycles. After the behavioral synthesis tool has developed an architecture that satisfies the designer's requirements, the behavioral synthesis tool will then automatically build the RTL code for the newly created architecture. In addition to the capability of generating several implements for an algorithm, the behavioral design flow provides efficiency gains for the developer. A designer may choose to create a behavioral model as part of the RTL design flow in order to verify the operation of the algorithm. However, in order for the designer to verify each architecture, he or she needs to provide an RTL specification and test the execution of that code. To determine whether or not an architecture meets requirements, it is necessary to run through the RTL synthesis procedure in its entirety. If an existing architecture does not meet the requirements, a new RTL model will need to be developed and validated. This can result in a lengthy procedure that may or may not be anticipated. In contrast to the RTL design flow, the behavioral design flow has a direct link between both the behavioral code and the RTL code that is being synthesized. This link is not present in the RTL design flow. During the behavioral flow, the behavioral synthesis tool is used to do a speedy evaluation of several possible topologies. The RTL code is automatically generated by the behavioral synthesis tool once a good architecture has been acquired. Due to the strong connection that exists between the behavioral specification and the RTL code that was generated as a result of that specification, the potential of having to perform gate-level results over and over again is drastically reduced, if not eradicated entirely. Even when it is essential to select a different architecture in the behavioral design flow because the characteristics of the gate-level design are insufficient, this is still a pretty straightforward process. This is true even if it is necessary to do so. To generate a different design, the constraints that are sent into the behavioral synthesis tool can have their parameters adjusted, but the behavioral code itself does not have to be rewritten. During the RTL flow, new code needs to be developed in order to accurately represent the design's revised architecture once changes were made.

IV. Binding in Behavioral Synthesis

As an example solution, this paper will try to explain the algorithm presented by N. Ranganathan and Ashok K. Murugavel in their paper: "A Game-Theoretic Approach for Binding in Behavioral Synthesis". For the purpose of the bidding strategies auction problem, the binding problem is described as a Nash equilibrium function. Even though the functional units in the datapath have been optimized for low



```
start_schedul      <= '1' ;
-- Start Scheduler Process
 qid_to_sche      <= qid_send;
 wait until rising_edge(clk);

   wait until  (done_schedul = '1' );
  start_schedul      <= '0';

   a) Behavioral code
```

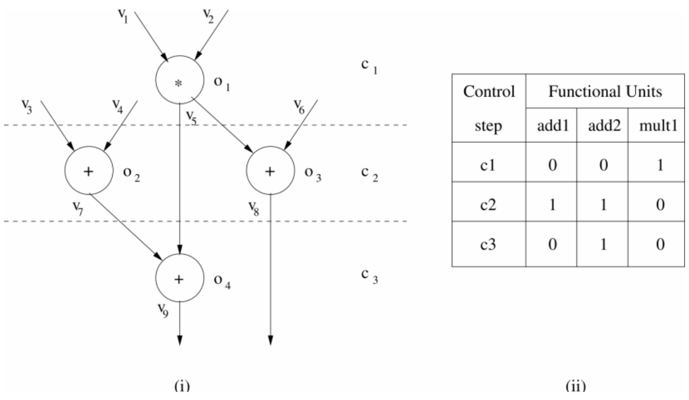
```
    next_start_schedul          <=  '1';
    next_qid_to_sche            <= qid_send;
    NEXT_STATE_decisorp <= ST_decisorp_1 ;

when ST_decisorp_1 =>
  NEXT_STATE_decisorp    <= ST_decisorp_2 ;

when ST_decisorp_2 =>
  if ((done_schedul =  '1') then
     next_start_schedul          <=  '0';
     NEXT_STATE_decisorp <= ST_decisorp_3 ;
  else
     NEXT_STATE_decisorp    <= ST_decisorp_2 ;
  end if;

                 b) RTL code
```

Fig. 4. RTL vs Behavioral Code



Control	Functional Units		
step	add1	add2	mult1
c1	0	0	1
c2	1	1	0
c3	0	1	0

Fig. 5. scheduled Data Flow Graph [3]

power consumption, there is still room for additional power savings during the binding process. Since game theory focuses on optimizing systems that contain multiple components, it is an excellent resource for solving the binding problem. The binding problem is an example of an NP-complete problem. As a result of the fact that the difficulty of locating the Nash equilibrium is proven to be between ”P” and ”NP” in, this fact serves as the primary impetus for our research into the Nash equilibrium model for binding. After that, we proceed to propose a game-theoretic framework for the binding problem that is based on auction tactics. A set of sellers is modeled as the functional units or modules, while a set of buyers is depicted as the actions that make up a control cycle. The modules compete for the operation by submitting anonymous bids, and the winner is determined by who offered the lowest price. When performing an operation at a module, the amount of power consumed becomes the module’s bid. In our formulation, we make the following presumptions about the conditions. The plan that is devised for each control cycle is considered to be an independent entity, and the overarching strategy does not extend across numerous control cycles. All of the game’s operators have access to comprehensive information regarding each module.

$$co(m,o) = P_{sw} - P_{inp}$$

Fig. 6. Cost Function [3]

Within any particular cycle, a module can be allocated to just one of cycle’s operations. If more than one module submits the same cheap offer for an operation, the operation will choose between them based on other requirements; if all of the other requirements are the same, then it will be chosen at random. The switching cost, denoted by P_{sw}, and the cost incurred as a result of varying inputs, known as neighborhood operations, are the two components that go into defining the cost function P_{inp}. Simulations are used to establish the cost reflecting the power consumption that is caused by the number of inputs changing (two, one, or none) and the activity of switching modules. It is assumed that there are two changing inputs, which is what causes the switching cost P_{sw}. As a result, the switching cost that is caused by the number of changing inputs P_{inp} needs to be deducted from the total cost of switching a whole module before it can be taken into account. Now that that’s out of the way, let’s walk through an example to illustrate it, using the DFG shown in Fig. 5 as a guide, and with the resource limit set at three adders and one multiplier. It is important to note that the strategies are developed independently for each set of compatible modules. For example, in order to accommodate two sets of compatible modules (adders and multipliers), it is necessary to develop and use two distinct methods. This constitutes a constraint. Because there is just one multiplier, the simplest and most optimal way is to link the multiplication operation to the multiplier module, which is not taken into account here. A mathematical representation of the cost function is as shown in Fig. 6.

In control step *c2*, there are two added operations and three adders. The cost matrix, abbreviated as CM, is as shown in Fig. 7. In the paper, K. Murugavel, et al. describe the rest of the algorithm as follows [3]:

> ”*a,c,e* are the cost associated with binding operation *o1* with the adders *add1, add2, add3* respectively and *b,d,f* are the cost associated with binding operation *02* with the adders *add1, add2, add3* respectively. Simply, *co(m,o)* is the cost of binding module *m* with operation *o*. The bidding strategy in this work, will be the bids from the modules for the operators 1 and 2. The Nash Equilibrium is the set of strategies i.e., bids for the operators 1 and 2, such that no module can gain by deviating unilaterally, i.e., a module cannot gain by changing the bids for any of the operators.
>
> The complete set of module and operation pairs that are possible for a given cycle is referred to as *MO*. In the examples if adder 1 is combined with operation 1 and adder 3 with operation 2, the *MO* will be *(1,1),(3,2)*. For the set

$$\begin{array}{cc} o1 & o2 \\ \left[\begin{array}{cc} a & b \\ c & d \\ e & f \end{array}\right] \end{array}$$

Fig. 7. Cost Matrix [3]

```
input   : Cost Matrix $CM$, Number of player $N$,
          Set of various possible strategies $S$
output  : Nash equilibrium solution $NE$
  for each player $i$, where $i \leftarrow 1$ to $N$ do
      for each set of strategies $(n_1 \in S_1)$,...,$(n_N \in S_N)$
      do
          Calculate $p^i_{(n_1,...,n_N)} = CM[i, n_i]$ ;
      end
% The payoff for strategy set $(n_1, ..., n_N)$ is calculated ;
      for each strategy $(n_i \in S_i)$ do
          $n^*_i \leftarrow$ Nash equilibrium strategy for
          player $i$ calculated using the inequality
          $p^i_{(n^*_1,...,n^*_i,...,n_N)} \geq p^i_{(n^*_1,...,n_i,...,n_N)}$ ;
      end
      $NE \leftarrow NE \cup n^*_i$ ;
  end
% $NE = \{n^*_1, ..., n^*_i, ..., n^*_N\}$, set of optimized strate-
  gies for all $N$ players
```

Fig. 8. Nash Equilibrium Solution [3]

> *MO*, the total cost $CO = co(1,1) + co(3,2)$. Now, for each control step, the problem is to find the minimum of all $COs(CO_{min})$, $CO_{min} = min_{l,k=1,2,3 l \neq k}[co(m_l, 1) + co(m_k, 2)]$. "

A. Nash Equilibrium Solution

The authors (K. Murugavel, et al.) continue on with two algorithms. This paper will only present the one that was deemed the most optimal: the "Nash Equilibrium Solution". They explain as follows [3]:

> "The worst case complexity of the proposed binding algorithm depends on *(i)* latency l of the *S-DFG*, *(ii)* number of compatible operations n, and *(iii)* payoff matrix size. Thus, the complexity of the algorithm is dominated by that of determining the Nash equilibrium... a polynomial time algorithm has been proposed to determine the Nash equilibrium."
> The algorithm consists of elements described in Fig. 9 and the code itself is presented in Fig. 8.

It is ultimately concluded that the bid value is equal to the cost when using the Nash equilibrium solution (see Fig. 10 and Fig. 11), showing that the non-cooperative approach was the most successful.

G_f	scheduled dataflow graph,
G_a	architecture,
O_i	set of all operations in a control step,
x_j	number of modules of type j in architecture,
y_j	number of operations of type j,
M_{ij}	set of all modules of type j in step i,
O_{ij}	set of all operations of type j in step 1,
CM	the cost matrix, $CM = [co(m, o)]$,
NE	set of strategies satisfying the Nash equilibrium,
S	set of all module-operation pairs available,
A	$A \subset S$ any feasible set of module-operation pairs,
MS_A	module set corresponding to A
CO	total cost for a specific $A \subset S$,
CO_{min}	the minimum total cost of any $A \subset S$.

The generalized cost matrix CM of size $x_j \times y_j$ is given as,

$$CM = \begin{bmatrix} co(1,1) & \cdots & co(1, y_j) \\ \ldots & \ldots & \ldots \\ co(x_j, 1) & \cdots & co(x_j, y_j) \end{bmatrix}$$

Fig. 9. Algorithm Legend [3]

$$\sum_{u=1,..,y_i} co(m_A(u), u) \geq\geq \sum_{O_2 \cup O_3} co(u, u) + \sum_{o_1} co(u, u)$$
$$therefore CO(A) \geq CO_{min} \forall A \in S$$

Fig. 10. Bid Value is Equal to Cost [3]

Benchmark Circuit	Resource constraint A, M	Random approach mW	Greedy approach mW	LP-based approach [16]		Game-theoretic approach		% Improvement over LP
				Power mW (P_{LP})	% savings w.r.t. Rand.	Power mW (P_{GT})	% savings w.r.t. Rand.	$\frac{(P_{LP})-(P_{GT})}{(P_{LP})}$
Diff. eqn	2, 2	26.6	25.0	22.9	14.0	20.1	24.4	12.2
FIR	3, 3	95.7	88.4	72.0	24.8	57.7	39.7	19.8
IIR	2, 2	18.3	17.1	15.5	15.3	12.3	32.8	20.6
Lattice	3, 3	113.0	98.6	71.9	36.3	64.1	43.2	10.8
EWF	2, 2	317.2	276.9	234.3	26.1	210.0	33.8	10.4
WAVE	2, 2	291.9	268.2	215.1	26.3	196.3	32.7	8.7
NC filter	2, 2	438.1	414.0	368.5	15.9	312.8	28.6	15.1

Fig. 11. Results of Different Experiments [3]

V. Conclusion

Game theory has proven to have a vast scope of applications, but not all of which are the most optimal approach at any given time. Different uses result in different outcomes, but the engaged and active nature of the concept has made game theory usable in a variety of different algorithms which, as shown in the examples in this paper, are definitely translatable into behavior synthesis designs. However, it must be noted that game theory does indeed increase the complexity of the abstraction that high-level synthesis brings. Nonetheless, it still presents itself as a worthwhile consideration for projects that are actively determining the most high efficiency approach to is problem-sets.

References

[1] Reinforcement Learning and Decision Making in Monkeys During a Competitive Game - D. Lee, M.L. Conroy, B.P. McGreevy, and D.J. Barraclough

[2] Mathematical Social Sciences - Branzei, R., D. Dimitrov, and S. Tijs

[3] A Game-Theoretic Approach for Binding in Behavioral Synthesis - N. Ranganathan and Ashok K. Murugavel

[4] Game Theory and its Applications (Static and Dynamic Games with Complete Information) - G. Abdoli

[5] The Genetical Theory of Natural Selection - R. Fisher

Other material used:

- Fig. 1 https://commons.wikimedia.org/wiki/File:Collision_game.jpg
- Fig. 3 https://www.researchgate.net/publication/2306800/
- Fig. 4 https://www.researchgate.net/profile/Mario-Diaz-Nava/publication/3810046/

VI. Declaration of Authenticity

I herewith declare that I wrote this thesis on my own and did not use any unnamed sources or aid. Thus, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made by correct citation. This includes any thoughts taken over directly or indirectly from printed books and articles as well as all kinds of online material.