University of Novi Sad
Faculty of Sciences
Department of Mathematics and Informatics

Analysis of fast winning strategies in Avoider-Enforcer “Non-bipartite“ game

- Master Thesis -

Student: Aleksandra Hajder
Mentor: Prof. Miloš Stojaković

Novi Sad, 2021.

Acknowledgments

Firstly, I would like to thank and express my gratitude to my supervisor Dr. Miloš Stojaković for his support and guidance throughout the whole project. It was such an honor to have a mentor like him.

Secondly, I wish to show a great appreciation towards my colleagues who were major support in every aspect. Without them, this journey of studying would be much harder. Special thanks to Bogdan Stankovski who was a great study partner and beyond that the best friend I could wish for. Also, I would like to thank Klaudia Belić who was my partner in the numerous projects, it was always a pleasure working with her.

And last but not least, I would like to extend my thanks to my family and friends for their emotional support both throughout the studies and the thesis. You always believed in me, even when I did not believe in myself.

Abstract

Positional games are often played on different types of graphs. They involve two players whose goals oppose. The most popular game of this type is Tic-Tac-Toe and its higher-dimensional generalizations. The game that is the focus of this paper is called the Non-bipartite game, played on the complete graph. Players are called Avoider and Enforcer. Their names say a lot about their goals. Avoider is trying to avoid creating a non-bipartite subgraph while Enforcer is trying to enforce Avoider to do exactly that. Our goal is to verify that the game is going to be played within the proven boundaries and to see where exactly is the duration of the game when both players stick to their optimal strategies. Avoider's strategy was laid out in detail, but for Enforcer, we have tried to make some improvements. In the future, it would be interesting to see how the duration of the game fluctuates when we change the priorities of the moves for each player. There is a lot of space for future investigation of this particular game, but also in general of positional games.

Contents

Introduction

Positional games are a type of combinatorial games, researching a variety of two-player games to purely abstract games played on graphs and hypergraphs [5]. They can be described as an alternate occupation of the previously unclaimed elements of a given set X that is called the board of the game. The focus of each player is a family $F = \{A_{1,\dots}, A_n\} \subseteq 2^X$ of finite subsets of X and we call them winning sets. This family is sometimes called a hypergraph of the game. There are three possible outcomes of each positional game: the first player has a winning strategy, the second one has it and both players have drawing strategies. The focus of this thesis is the Avoider-Enforcer games. In games of that type, we have two players. The first one is trying to avoid a graph property whilst the second one is trying to force him to claim the edges that he wants to avoid. The goal of Avoider is defined through a negation – he wins if he does not occupy any member of the hypergraph-losing set [5]. Sometimes the victory of Enforcer is inevitable. In that case, a new and more interesting question arises, in how many moves will he manage to win? We measure the speed of victory in the number of moves (or sometimes rounds) needed for that victory to happen. Fast winning strategies for Avoider-Enforcer Non-bipartite game will be the main part of this paper. This game is played on a complete graph where players alternately claim an edge following some strategies. Avoider loses the game as soon as a graph made up of his previously claimed edges becomes non-bipartite. On the other hand, Enforcer is trying to enforce Avoider to claim an edge that is going to make his graph non-bipartite as early as possible during the game.

The main contribution of this work is the implementation of different strategies and analysis of a different number of rounds required for Enforcer to win. It was known to us from before the size of the interval within which the game must end. But in this thesis, it is analyzed for the first time exactly where within it the values are and how we can maybe move them closer to the desired part of the interval.

1. Graph theory, preliminaries

1.1 Definition and representation of a graph

In the everyday world, we can find many relationships, structures, connections, etc. that can be represented using some mathematical objects. If a structure consists of a set of points that are usually named or marked in a way and if those points can be related somehow with some sort of lines, we can use graphs to represent the wanted structure and after that to do an interesting analysis on it.

For better understanding, let us start with an example. We can take a group of people and represent each person as a point. The relationship between them can be represented with a line. We have a line that connects two points if two persons know each other otherwise there are no connections between them.

Introducing the graph theoretic notation and well-known statements, we follow [9]. A graph G is usually defined as ordered pair $(V(G), E(G))$ consisting of a set $V(G)$ that is a set of vertices also called nodes, and $E(G)$ that is a set of edges that are unordered pair of vertices $E(G) \subseteq \{\{x, y\} \mid x, y \in V(G) \text{ and } x \neq y\}$ together with an incidence function ψ_G which associates with each edge of G an unordered pair of vertices of G.

Two main parameters that can easily be calculated are the order and size of a graph. Order is the number of vertices and usually is denoted by $v(G)$ and size is the number of edges usually denoted by $e(G)$.

Graphs can easily be represented graphically and that is why they are named like that. Sometimes, it can be important to do the representation in a nice, clean way, because it can be easier to notice some of the properties the graph has. Furthermore, the same graph can be drawn in many different ways and you can find one example in Figure 1.1.

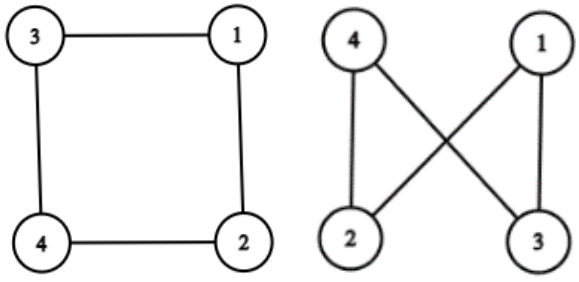

FIGURE 1.1 DIFFERENT DRAWINGS OF THE SAME GRAPH

Terms incident and adjacent are often used. The edge is said to be incident with its end vertices and the other way also holds. We use the term adjacent when we have two vertices that are incident with a common edge and also when having two edges that are

incident with a common vertex. Vertices are called neighbors if they are distinct and adjacent. The neighborhood of a vertex v in graph G is a set of vertices that contains all vertices adjacent to v.

An edge that starts and ends in the same vertex is called a loop and an edge with distinct ends is called a link. If there are two links with the same pair of ends, then we have parallel edges. A graph is simple if it has no loops or parallel edges.

1.2 Subgraphs and special families of graphs

Starting from graph G, two common ways can be used to derive smaller graphs from G. As one can assume, we can delete an edge or a vertex in some ways. Two operations that can be helpful are edge deletion and vertex deletion. $G \setminus e$ is a graph obtained from G by deleting the edge e. Similarly, $G - v$ is a graph obtained by deleting vertex v together with all the edges incident with it. Using these operations, we can create subgraphs.

Speaking in a more general way, a graph F is called a subgraph of a graph G if $V(F) \subseteq V(G)$, $E(F) \subseteq E(G)$, and ψ_F is the restriction of ψ_G to $E(F)$.

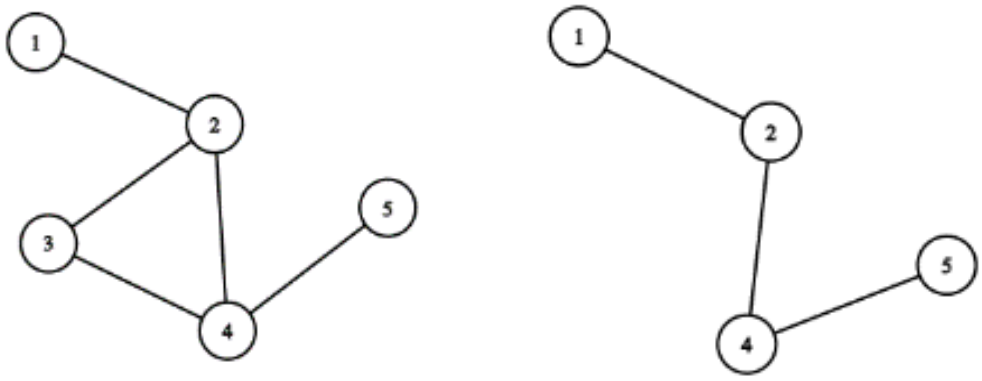

FIGURE 1.2 GRAPH G (ON THE LEFT), SUBGRAPH F OF THE GRAPH G (ON THE RIGHT)

A *spanning subgraph* of graph G is a subgraph obtained by edge deletions only. Another way to define it is to say that a spanning subgraph is a subgraph whose vertex set is the entire vertex set of G. If we define S to be the set of deleted edges, then this subgraph of G is denoted by $G \setminus S$.

A *complete graph* is a simple graph in which any two vertices are adjacent. A regular graph is a graph where each vertex has the same number of neighbors. A degree of a vertex of a graph is the number of edges incident with that vertex. The complete graph on n vertices I usually denoted by K_n. It has $n(n-1)/2$ edges. It is a regular graph and has a degree $n-1$. [7,9]

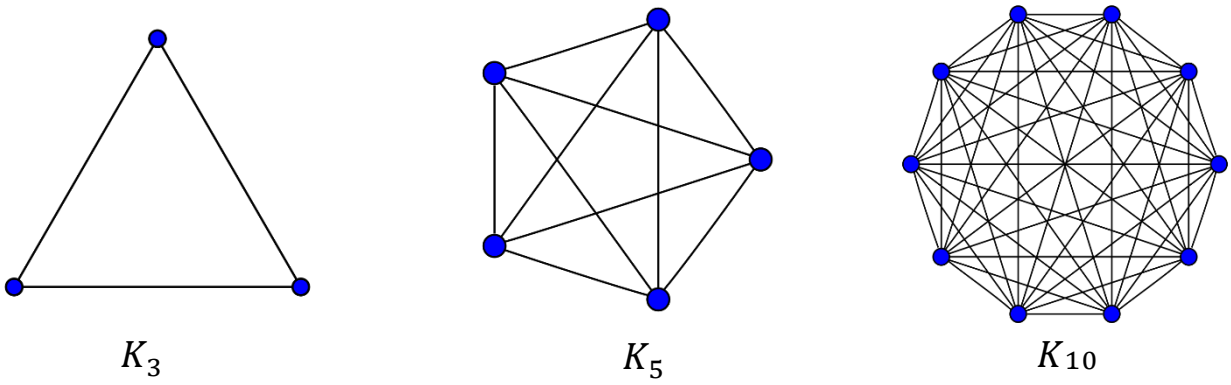

FIGURE 1.3 EXAMPLES OF COMPLETE GRAPHS WITH DIFFERENT NUMBER OF VERTICES

A graph is called *bipartite* if its vertex set can be partitioned into two subsets X and Y so that every edge has one end in X and one end in Y. That kind of partition (X,Y) is called a bipartition of the graph.

A cycle is a simple graph whose vertices can be arranged in a cyclic sequence in a way that two vertices are adjacent if they are consecutive in the sequence. A cycle is consisted of at least three vertices. The length of a cycle is the number of its edges, and we can have odd and even cycles depending on their length.

A *path* is a simple graph whose vertices can be arranged in a linear sequence in such a way that two vertices are adjacent if they are consecutive in the sequence, and are nonadjacent otherwise.

In graph theory, Turán's theorem bounds the number of edges that can be included in an undirected graph that does not have a complete subgraph of a given size [22]. The special case of Turán's theorem is Mantel's theorem that will be helpful later in the paper.

Theorem 1.1 (Mantel's): [17] If a graph G on n vertices contains no triangle then it contains at most $\frac{n^2}{4}$ edges.

1.3 Trees

A *tree* is a connected acyclic graph and an acyclic graph is one that contains no cycles. Each component of an acyclic graph is a tree, these acyclic graphs are called forests. A connected graph must contain at least one path between any two vertices. So, trees are always connected, but we have exactly one path between any two vertices.

Any graph in which all degrees are at least two contains a cycle. From that, it can be concluded that every tree contains a vertex of degree at most one and if the tree is nontrivial, it must contain that one vertex, and it is called a *leaf* of the tree. In Figure 1.4 you can find a few examples of trees on six vertices.

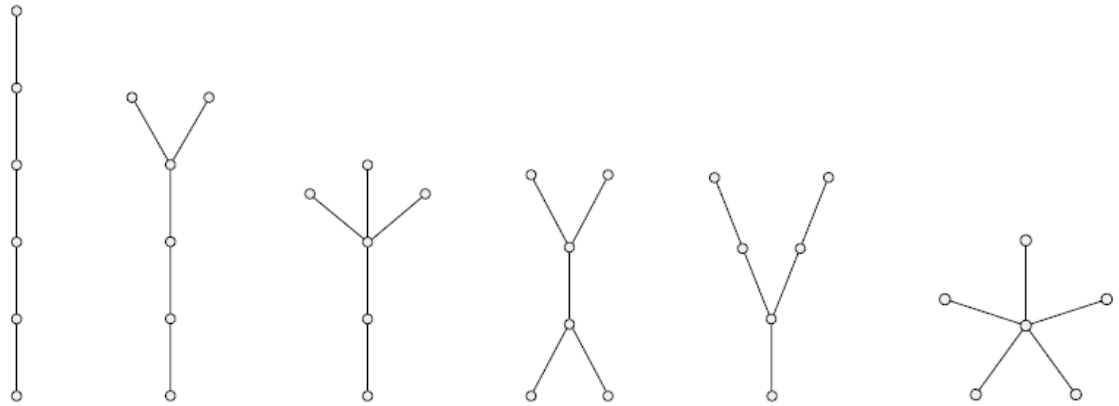

FIGURE 1.4 THE TREES ON SIX VERTICES

A *subtree* of a graph is a subgraph which is a tree [9]. If this tree is a spanning subgraph, we call it a *spanning tree.*

Theorem 1.2: A graph is connected if and only if it has a spanning tree.

Proposition 1.3: In a tree, any two vertices are connected by exactly one path.

Theorem 1.4: A graph is bipartite if and only if it contains no odd cycle.

Proof:

Firstly, we can easily see that a graph is bipartite if and only if each of its components is bipartite. Also, a graph contains an odd cycle if and only if one of its components contains an odd cycle. This is what we will need in further proving.

⇒ Let $G[X,Y]$ be a connected bipartite graph. Then the vertices of any path in G belong alternately to X and to Y. All paths that are connecting vertices in different parts are of

odd length and all paths connecting vertices in the same part are of even length. By the definition of G, each edge of G has one end in X and one ned in Y. From that, we can conclude that every cycle of G is of even length.

⇐ Now, suppose that G is a connected graph without odd cycles. We will need Theorem 1.2 and Proposition 1.3 to complete the proof.

From Theorem 1.2 we can immediately conclude that G has a spanning tree T because it is connected. Now let x be a vertex in T. Because of Proposition 1.3, we know that any vertex v of T is connected to x by a unique path in T. Let X denote the set of vertices for which this path is of even length, and set $Y := V \setminus X$. Then (X, Y) is a bipartition of T. It is left to prove that this is also a bipartition of G.

Let us consider an edge $e = uv$ of $E(G) \setminus E(T)$ and let $P := uTv$ be the unique uv - path in T. The cycle $P + e$ is even, so P must be odd. Therefore, the ends of P, and hence the ends of e, belong to distinct parts. From here we can conclude that (X, Y) is a bipartition of G. □

1.4 Tree-search algorithms

The two most important and most used algorithms on graphs are BFS – Breadth-first search and DFS – Depth-first search.

By graph traversal, it is meant visiting every vertex exactly once in a well-defined order.

1.4.1 BFS

Bondy and Murty dealt with and researched these algorithms in detail in their book [9]. BFS is a traversing algorithm that follows the rule 'first com first served'. That means that starting from the root/source (arbitrary vertex) it takes into account all of its neighbors first then moves along with visiting neighbors' neighbors. For the implementation of this algorithm, vertices are kept in a queue. A queue is a list Q that is updated when two situations occur. The first update can be adding a new element always at the end (the tail of the queue) and the second one is removing an element from the top (the head of the queue). Below you can find Algorithm 1.1 [9] together with a short illustration of how the algorithm works on the graph with 8 vertices that are connected in a way on Figure 1.5.

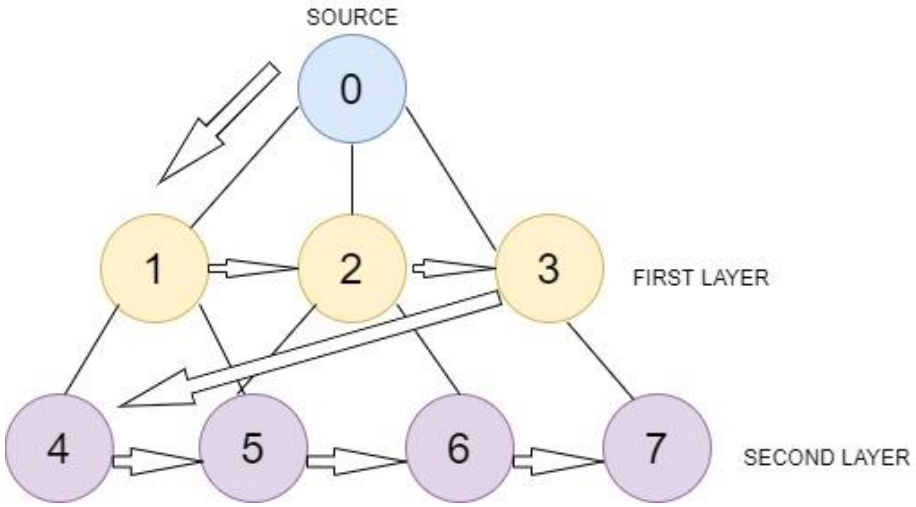

FIGURE 1.5 BFS EXAMPLE

INPUT: A connected graph *G*
OUTPUT: An r-tree T in G with predecessor function p, a level function l such that $l(v) = d_G(r, v)$ for all $v \in V$, and a time function t

1: set $i := 0$ and $Q := \emptyset$
2: increment i by 1
3: colour r black
4: set $l(r) := 0$ and $t(r) := i$
5: append r to Q
6: **while** Q is nonempty **do**
7: consider the head x of Q
8: **if** x has an uncoloured neighbour y **then**
9: increment i by 1
10: colour y black
11: set $p(y) := x$, $l(y) := l(x) + 1$ and $t(y) := i$
12: append y to Q
13: **else**
14: remove x from Q
15: **end if**
16: **end while**
17: return (p, l, t)

ALGORITHM 1.1 BFS ALGORITHM [9]

1.4.2 DFS

DFS is also a traversing algorithm but it follows another kind of rules than BFS. It can be explained in the following way: it starts from the source/root vertex and goes as far as possible through the branch before backtracking. It is a recursive algorithm because it uses backtracking. We search for vertices by going ahead, if possible, else by backtracking. This algorithm can be implemented by using a stack. A stack is a list S and it may be updated in two ways – by adding a new element at the top or by removing an element from its top. You can read more about this algorithm defined through steps [9]. In Figure 1.6 there is a simple example of how this algorithm works on a tree with 5 vertices.

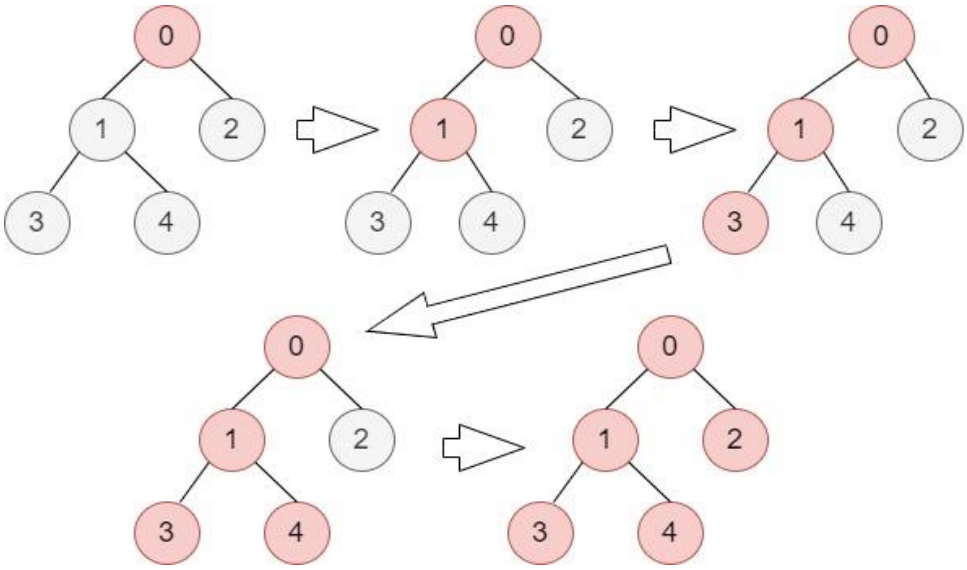

FIGURE 1.6 DFS EXAMPLE

```
INPUT: A connected graph G
OUTPUT: A rooted spanning tree of G with predecessor function p, and
two time functions f and l

1: set i := 0 and S := ∅
2: choose any vertex r (as root)
3: increment i by
4: colour r black
5: set f(r) := i
6: add r to S
7: while S is nonempty do
8:      consider the top vertex x of S
9:      increment i by 1
10:     if x has an uncoloured neighbour y then
11:             colour y black
12:             set p(y) := x, f(y) := i
13:             add y to the top of S
14:     else
15:             set l(x) := i
16:             remove x from S
17:     end if
18: end while
19: return (p, f, l)
```

ALGORITHM 1.2 DFS ALGORITHM [9]

1.4.3 Bipartite graph and BFS algorithm

An interesting thing is that we can use the BFS algorithm to find out whether a graph is bipartite or not. We already talked about checking if a graph contains an odd cycle, but this is a different approach. By following these steps, we can determine the wanted property. Two colors are needed, we will use red and blue for simplicity.

1) Assign a blue color to the source vertex.
2) Color all of its neighbors with red color.
3) Color all neighbor's neighbor with blue color.
4) By repeating this process, assign a color to all the vertices in the graph.
5) While assigning, if we find neighbors that are of the same color, then the graph is not bipartite, otherwise, it is.

The following Theorem 1.5 can also be useful to understand why we can use this approach explained above.

Theorem 1.5: [16] Let G be a graph. Then G is 2-colorable if and only if G is bipartite.

Proof:

Proof of this theorem is pretty straightforward.

⇒ Let G be a 2-colorable graph. That simply means that we can color every vertex either red or blue, and no edge will have both endpoints colored the same color. Let X denote the subset of red vertices, and let Y denote the subset of blue vertices. Since all vertices of X are red, and all vertices of Y are blue, we can conclude that every edge has one endpoint in X and the other in Y. So, G is bipartite.

⇐ Now suppose that G is a bipartite graph. That means that we can partition the vertices into two subsets X and Y in a way that every edge has one end in X and another in Y. If we color all the edges from X in red and all the edges from Y in blue, we will get a proper coloring. Because two colors are used, we can say that G ls 2-colorable. □

2. Positional games, preliminaries

The term 'positional games' can be wrongly understood as being a part of classical Game Theory. Classical Game Theory is mostly based on the notions of uncertainty and lack of perfect information. On the other hand, positional games are perfect information games and because of that, they can be solved completely by an all-powerful computer. Positional games are closer to the so-called "Combinatorial Game Theory" in which games are based on algebraic arguments and various notions of decomposition [2].

Positional games are games that can be described as an alternate occupation of the elements of a given set X that is called the *board* of the game [2,19]. We assume that X is finite. *Winning sets* are the focus of each player, and they can be described as a family $F = \{A_{1,\dots}, A_n\} \subseteq 2^X$ of finite subsets of X, this family is sometimes called the hypergraph of the game. [14] The outcomes of the game are – the first player wins / the second player loses, the second player wins / the first player loses, or a draw. Each game scenario has exactly one of the outcomes. There is no randomness involved in these games. The outcome of each positional game is determined and speaking of the outcomes, combining with the strategies, these are the only possible ones [2]:

1. the first player has a winning strategy,
2. the second player has a winning strategy,
3. both players have drawing strategies.

Knowing that a game is determined and finding its actual outcome are two very different things. In principle, every game can be described by a tree of all possible plays, called the game tree. There is a vertex for every sequence of allowed moves of both players, including the empty sequence for the root of the game tree. Each sequence of moves is connected by an edge to a sequence one move shorter. Leaves are the final positions of the games. [2,12]

The most famous positional game is Tic-Tac-Toe in two dimensions. As we know, this game is played by two players, alternately claiming one unoccupied cell from a 3-by-3 board. A player who completes a winning line first wins. We have eight winning lines, three vertical lines, three horizontal lines, and two diagonals. If none of these lines are claimed by neither one of the players, in that case, we have a draw.

2.1 Maker – Breaker games

Generally speaking, in every positional game both players are trying to do two things simultaneously: try to occupy a complete winning set and prevent the other player from occupying one for themselves. For many reasons, analyzing this approach is impractical and very complex. Because of that, we focus on games where the second player (SP) is not interested in occupying a winning set but achieving a draw, or basically, his strategy is focused on preventing the first player (FP) to win. Additionally, FP can concentrate on offense and completely forget about playing defense. By changing the strategies for both players, we are simplifying the game.

Definition 2.1: Let X be a finite set and $\mathbb{F} \subseteq 2^X$ a family of subsets. In a Maker-Breaker game over the hypergraph $(X, \mathbb{F})$:

- the set X is called the board and the elements of $\mathbb{F} \subseteq 2^X$ are the winning sets;
- the players are called Maker and Breaker;
- during a particular play, the players alternately occupy elements of X; as a default, we set Maker to start (unless stated otherwise);
- the winner is:
    - Maker, if he occupies a winning set completely by the end of the game,
    - Breaker, if he occupies an element in every winning set.

2.2 Biased games

In many different Maker-Breaker games, Maker wins easily, so Chvátal and Erdös [23] were first to suggest that Breaker claims more than one edge per move in order to increase his chance of winning.

Definition 2.2: Let p and q be positive integers, let X be a finite set, and let $\mathbb{F} \subseteq 2^X$ be a family of subsets of X. *The biased* $(p: q)$ Maker-Breaker game $(X; \mathbb{F})$ is the same as the Maker-Breaker game $(X; \mathbb{F})$, except that Maker claims p free board elements per move and Breaker claims q free board elements per move. The integers p and q are referred to as the bias of Maker and Breaker, respectively. In the last move of the game, if there are fewer free board elements than his bias, a player claims every free board element.

2.3 Avoider – Enforcer games

Avoider – Enforcer games are in a way the opposite of Maker–Breaker games and that is why they are sometimes called Antimaker – Antibreaker games. As their name says, we have two players, the first one is trying to avoid a graph property and the second one is trying to force him to claim the edges that he wants to avoid. The general setup is pretty much the same as in other positional games that are already described, we have the board X and the collection of winning sets, but in these games, we refer to that collection as the collection of losing sets $\mathbb{F}$. Avoider is starting the game unless it is specified differently. [2,5]

Let p and q be positive integers and let $\mathbb{F}$ be any hypergraph. In a $(p, q, \mathbb{F})$ biased Avoider–Enforcer game two players take turns selecting previously unclaimed vertices of $\mathbb{F}$. Avoider selects exactly p vertices per move and Enforcer selects exactly q vertices per move. If the number of unclaimed vertices is strictly less than p (or q) before a move of Avoider (or Enforcer, respectively), then he must claim all of the remaining free vertices [3]. The game ends when all the elements of the board are claimed either by Avoider or Enforcer. The goal of Avoider is defined through a negation, that is, he wins if he does not occupy any member of the hypergraph–losing set [5]. Enforcer wins if Avoider claims a whole set from the collection of losing sets. We can have a biased and unbiased version of this game. A biased game is more general and it is introduced to increase the players' chances to win. An unbiased version is one where p and q are equal to 1. [2,3,5]

The most popular game of this type is the so-called 'Sim'. The game is played on the complete graph with 6 vertices. In every move, each player is coloring an edge in one color, for simplicity, we can say that Avoider is coloring in red, and Enforcer in blue. Losing sets are all triangles. If Avoider had created a red triangle by coloring edges, he had lost, otherwise, he is a winner.

2.4 Fast winning strategies

Both Maker-Breaker and Avoider-Enforcer games can be analyzed in order to create strategies that will take one or the other player towards the win. Another interesting question that we can ask is how long will it take for a player to win rather than who is going to win. [13]

2.4.1 Fast winning in Maker-Breaker games

Our focus here is on the unbiased games played by two players that are taking turns in selecting edges of a complete graph. For quite a few Maker–Breaker games, it is rather easy to determine the identity of the winner [11]. For example, Maker wins very easily in the connectivity game [6]. In that particular game, his goal is to claim a connected and spanning subgraph. Another good example is the Non-planarity game [4,15] with $n > 11$, where his goal is to create a non-planar graph. The maker will definitely manage to claim such edges that will create a non-planar graph irregardless of his strategy because it is known that every graph with more than $3n - 6$ edges on n vertices is non-planar. In these and similar games, the most significant part is 'how fast one can win?'

2.4.2 Fast winning in Avoider-Enforcer games

Fast winning strategies for Avoider-Enforcer games will be the main part of this paper, particularly for the Non-bipartite game. Several well-studied positional games are an easy win for Enforcer. The previously mentioned non-planarity game can also be a good example of this. These strategies for the fast win in the non-planarity game are described in detail in [1] and later in this paper, we will be dealing with strategies for the Non-bipartite game. It is known that Enforcer will eventually win, but the interesting part is how long will Avoider manage to avoid losing.

3. Fast winning strategies in Avoider-Enforcer games

As it was mentioned previously, one player has a strategy for winning. In case we know the winner we are moving to the question of how fast that player can win. We will assume that we have a complete graph K_n, the game is played on its edges $E(K_n)$ and that the game is unbiased unless it is said otherwise.

For a hypergraph $\mathbb{F}$, $\tau_E(\mathbb{F})$ is said to be the smallest integer t such that Enforcer has a strategy to win the game on $\mathbb{F}$ within t moves. If Avoider wins, we say that $\tau_E(\mathbb{F}) = \infty$. We are interested in determining the value $\tau_E(\mathbb{F})$. Let us assume that the set of hyperedges of $\mathbb{F}$ is a monotone increasing family. If the assumption is not correct, we can always extend it to an increasing family by adding all the supersets of its elements. [1]

Definition 3.1: *The extremal number* of the hypergraph $\mathbb{F}$ is defined by the following equation:

$$ex(\mathbb{F}) = max\{|A| : A \subseteq V(\mathbb{F}), A \notin E(\mathbb{F})\}.$$

Theorem 3.2: [1] Giving a monotone increasing family $\mathbb{F}$ of hyperedges, we have

$$\frac{1}{2}ex(\mathbb{F}) + 1 \leq \tau_E(\mathbb{F}) \leq ex(\mathbb{F}) + 1.$$

Proof:

We have two bounds for $\tau_E(\mathbb{F})$, the upper and the lower one. First, let us prove the lower bound. Let Avoider fix an arbitrary $A \subseteq V(\mathbb{F})$ before the game starts in a way that A is an edge of $\mathbb{F}$ and $|A| = ex(\mathbb{F})$. During the game, Avoider claims only the elements of A as long as possible. By doing that, he will be able to claim at least half of the elements of A without losing.

Enforcer will surely win after $ex(\mathbb{F}) + 1$ rounds, no matter what his strategy is. At that point, Avoider has claimed $ex(\mathbb{F}) + 1$ vertices and a set with that many vertices must be an edge of $\mathbb{F}$, because of the way that $ex(\mathbb{F})$ was defined. That is how we got the upper bound. □

3.1 Non-bipartite game

We now take a closer look at the Non-bipartite game. As it was mentioned, this game is played on a complete graph where players alternately claim an edge following its strategy. Avoider loses the game as soon as his graph becomes non-bipartite. As its name says, Enforcer is trying to enforce the Avoider to claim an edge that is going to make his subgraph non-bipartite. Enforcer will eventually win the game, but the interesting question is – how many moves will be necessary for Enforcer to achieve his goals.

Theorem 1.4 can be very helpful. That theorem equates this game with the game in which Enforcer aims to make sure that Avoider creates an odd cycle by claiming the edges in every round and Avoider is trying not to claim it. From Theorem 1.4, we know that if a graph contains an odd cycle it cannot be bipartite.

Now, let us denote by NC_n^2 the hypergraph whose hyperedges are the edge-sets of all non-bipartite graphs on n vertices.

From Theorem 3.2 and Theorem 1.1 we can conclude the following:

$$\frac{1}{2}\left\lfloor\frac{n^2}{4}\right\rfloor + 1 \leq \tau_E(NC_n^2) \leq \left\lfloor\frac{n^2}{4}\right\rfloor + 1.$$

It turns out that both upper and lower bounds can be improved.

Theorem 3.3: [9]

$$\tau_E(NC_n^2) = \frac{n^2}{8} + \theta(n).$$

As it was already proven by Hefetz, Krivelevich, Stojaković, and Szabó in [1], we can get more accurate boundaries than one stated in Theorem 3.3. Let us denote by τ the number of rounds needed for Enforcer's win in the Non-bipartite game. This means that after exactly τ rounds, Avoider will claim an edge that will create an odd cycle together with his previously claimed edges.

Theorem 3.4:

$$\frac{n^2}{8} + \frac{n-2}{12} \leq \tau \leq \frac{n^2}{8} + \frac{n}{2} + 1.$$

Proof:

Upper bound - forcing an odd cycle fast

Enforcer's strategy is based on claiming the edges in a way that all the edges left for Avoider to choose are going to make Avoider's graph non-bipartite.

His strategy should force Avoider to claim the edges of an odd cycle, and by doing that to lose the game, during the first $\frac{n^2}{8} + \frac{n}{2} + 1$ moves. Each connected component of Avoider's graph in every stage of the game is bipartite. If that would not be the case, then the whole graph would not be bipartite and Avoider would have already lost.

In every move, Enforcer's primary goal is to claim an edge that connects two opposite sides of the bipartition of one of the connected components of Avoider's graph. If that is not possible and no such edge is 'free' then he will claim an arbitrary edge. The edge that has been chosen arbitrarily is marked as 'possibly bad'. It is obvious that in the next move Avoider cannot play inside any of his connected components, because by doing that he

would create an odd cycle. So, he is forced to merge two of his components. We know that the game starts with n connected components (each vertex is one component because no edge has been claimed), this situation of merging two components can occur at most $n - 1$ times.

Therefore, when it comes to the move where Avoider is not able to claim any edge without creating an odd cycle, his graph is of course still bipartite and all of Enforcer's edges are compatible with that bipartition of Avoider's graph, except the ones that we marked as 'possibly bad'. The total number of claimed edges to this point is at most $\frac{n^2}{4} + n - 1$. The total number of the claimed edges is obtained based on Theorem 1.1 which says that if a graph on n vertices does not contain a triangle, which is an odd cycle of length 3, then it contains at most $\frac{n^2}{4}$ edges, together with the knowledge of how many times merging of connected components can happen which is $n - 1$. So because of that the total number of moves Avoider has played in the entire game is at most $\frac{n^2}{8} + \frac{n}{2} + 1$.

Lower bound - avoiding odd cycles for long

The strategy that will be explained below is a strategy for Avoider to keep his graph bipartite for at least $\frac{n^2}{8} + \frac{n-2}{12}$ rounds. For technical reasons let us assume that n is even. The idea is for Avoider to maintain a family of ordered pairs (V_1, V_2), where $V_1, V_2 \subseteq V(K_n)$, $V_1 \cap V_2 = \emptyset$ and $|V_1| = |V_2|$. The ordered pair that satisfies the conditions stated above is called a bi-bunch. Two bi-bunches (V_1, V_2) and (V_3, V_4) are disjoint if $(V_1 \cup V_2) \cap (V_3 \cup V_4) = \emptyset$. A vertex is called untouched if it does not belong to any bi-bunch and all the edges incident with it are unclaimed. These terms have been introduced because, during the game, we will maintain a partition of the vertex set $V(K_n)$ into several pairwise disjoint bi-bunches, and a set of untouched vertices.

Before the game starts, we have n untouched vertices and no bi-bunches. Avoider's strategy is defined in the following way:

The primary goal is to claim an edge across some existing bi-bunch, in other words, an edge (x, y) where $x \in V_1$ and $y \in V_2$ for some bi-bunch (V_1, V_2). If no such edge is available, then he will try to claim an edge (x, y) where x and y are untouched vertices. In that case, a new bi-bunch must be created, so we will have $(\{x\}, \{y\})$. If neither of that is possible, he will claim an edge connecting two existing bi-bunches, that is, (x, y) such that there exist (V_1, V_2) and (V_3, V_4) with $x \in V_1$ and $y \in V_3$. In this step, he needs to replace two existing bi-bunches with a single new one $(V_1 \cup V_4, V_2 \cup V_3)$.

Some changes must be done on bi-bunches, depending on the edge that has been claimed by Enforcer. When claimed edge (x, y) is such that neither x nor y belongs to any bi-bunch, a new bi-bunch is introduced $(\{x, y\}, \{u, v\})$, where u and v are arbitrary untouched vertices. If there are no two untouched vertices (that can happen only once in the game) then the new bi-bunch is $(\{x\}, \{y\})$. If Enforcer claims an edge (x, y) such that

$x \in V_1$ for some bi-bunch (V_1, V_2) and y is untouched, then we need to update the bi-bunch or, in other words, replace the existing one with $(V_1 \cup \{y\}, V_2 \cup \{u\})$, where u is an arbitrary untouched vertex. The next option is that the edge (x, y) claimed by the Enforcer is such that there are bi-bunches (V_1, V_2) and (V_3, V_4) with $x \in V_1$, $y \in V_3$. Then, these two bi-bunches are replaced with a single new one $(V_1 \cup V_3, V_2 \cup V_4)$. Notice that Avoider's graph will not contain an edge with both endpoints in the same side of a bi-bunch if we follow everything described above. Also note that if Enforcer claims an edge (x, y), such that before that move one of the vertices was untouched, then that edge will be contained in the same side of some bi-bunch.

Assume that Avoider claims an edge (x, y) such that before that move x was untouched. Then y must also be untouched and there are no unclaimed edges across a bi-bunch at that point because of the Avoider's strategy. In Enforcer's next move he will not be able to claim an edge across a bi-bunch and because of all that, the edge he will claim will have both endpoints in the same side of some bi-bunch. We can conclude that after every round in which one or both players claim an edge that is incident with an untouched vertex (that is not the next to last untouched vertex), the edge claimed by the Enforcer will be contained in the same side of some bi-bunch.

By the bi-bunch maintenance rules explained throughout the proof, during every round the number of untouched vertices is decreased by at most 6. Therefore, by the time all but two vertices are not untouched at least $\frac{n-2}{6}$ Enforcer's edges will be contained on the same side of a bi-bunch. Consequently, when Avoider must claim an edge that will create an odd cycle, both players have claimed together all the edges of a balanced bipartite graph that complies with the bi-bunch bipartition, and at least another $\frac{n-2}{6}$ edges. Putting all of this together we get a total of at least $\frac{n}{2} \cdot \frac{n}{2} + \frac{n-2}{6}$ edges claimed so at least $\frac{n^2}{8} + \frac{n-2}{12}$ rounds were played. □

Now, let us go through one example of how this game can be played according to the above described strategies of Avoider and Enforcer. We will take a complete graph on 6 vertices. In Figure 2.1 Avoider's edges are the blue ones and Enforcer's are the red ones. In his first move, Avoider claims the edge (1,2) and we marked it as blue. We immediately create a bi-bunch $(\{1\}, \{2\})$. Enforcer claims (3,4) and we marked it as red. After that move, a new bi-bunch is created $(\{3,4\}, \{0,5\})$, and the first round is finished. In the next one, Avoider claims (4,5) and that move does not require any changes on any element of the list of bi-bunches. Enforcer claims (1,3) and after that move, we are merging the bi-bunches and now we have one bi-bunch $(\{1,3,4\}, \{2,0,5\})$. Further, Avoider chooses (0,1). In every move, his primary goal is to claim an edge that connects two opposite sides of a bi-bunch. Enforcer claims (1,4). The following move of Avoider is claiming (2,3) and of Enforcer the claimed edge is (0,3). Then, by following their strategies, Avoider chooses (2,4) and Enforcer's choice is (0,4). There were no changes on the bi-bunch list

after these three moves. Avoider now claims $(1,5)$ and Enforcer follows with claiming $(3,5)$. These moves also do not require any changes on bi-bunches. And after all these rounds the only edge possible for Avoider to choose is $(0,2)$ and Enforcer follows with $(2,5)$. Now, let us take a look at the bottom right graph in Figure 2.1. Only three edges are not claimed at this point. It is Avoider's turn to play. Whatever edge he chooses, the blue subgraph will stop being bipartite or in other words, he will create a cycle of odd length. By doing that, he loses the game. So, in this particular example, Avoider manages to keep his graph bipartite for 6 rounds, and in the 7th round, he will claim an edge that is going to end the game. If we calculate the upper and lower bound we will get 4.83 for the lower one and 8.5 for the upper one and obtained result, in this case, was 6.

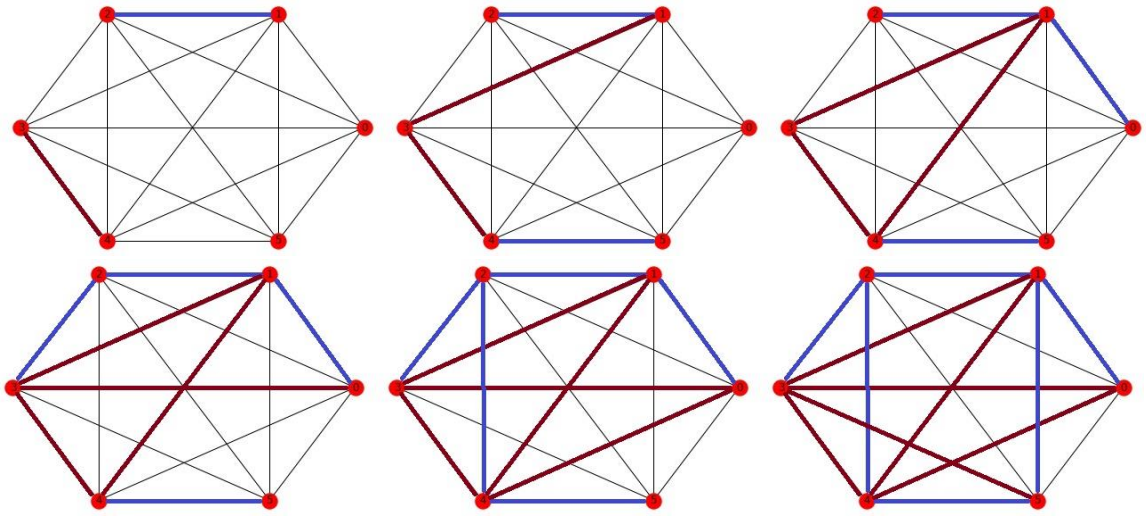

FIGURE 3.1 NON-BIPARTITE GAME ON THE COMPLETE GRAPH WITH 6 VERTICES

3.2 Improving Enforcer's strategy

Based on the description of the strategies of both players, it can be concluded that Avoider's strategy is completely deterministic. On the other hand, within Enforcer's strategy, there is an arbitrary part that leaves room for potential improvement.

Enforcer's strategy is easy to follow and does not contain many steps. His primary goal is to claim an edge that connects two opposite sides of the bipartition of Avoider's graph. Every connected component of Avoider's graph is a separate bipartition. The code for that part is explained in Algorithm 4.2. If he is unable to do that, he simply chooses an arbitrary edge. The main idea was to replace the arbitrary step with well-defined steps that will potentially improve his strategy and give him a faster win.

After a thorough analysis of potential changes in Enforcer's way of playing, we decided to compare two different approaches to defining his strategy. The first one is named 'Random strategy' and the second 'Twin strategy'. In the Random strategy, Enforcer has

only two possible choices. If it is possible, he will claim an edge that connects two opposite sides of the bipartition of one of the Avoider's connected components. This step was explained in the proof of Theorem 3.4. If that kind of edge does not exist, he is going to claim a randomly chosen edge. Because of this random part, every time the game is simulated it is possible to get a different result. The second strategy called the Twin strategy consists of three parts. First is always the same, so it has been already explained. The second and the third options are making a 'twin' to Avoider's strategy. We want Enforcer to claim the edges that Avoider has an intention to claim in his future moves. So, the second option for him is to claim an edge that is consisted of two untouched vertices. But, following his strategy for creating or altering bi-bunch, we need two more edges that are also untouched to put in a bi-bunch created at that point. If this move is not possible also, he will try to find an edge that connects two bi-bunches. By doing so, alterations must be done on bi-bunches – they should be merged oppositely of merging when Avoider claims that kind of edge. If neither of these options is possible, he again claims an edge by random choice.

After comparing these two types of strategies, one more idea came to life. We wanted to remove the random part completely, so after the first possible choice for Enforcer, which is already explained many times, the second option was to claim an edge whose vertex degrees give the smallest value when added together. Degree refers to a subgraph of claimed edges by that point. We called this strategy 'The smallest'.

4. Implementation of strategies

4.1 Players' strategies

After a thorough analysis of game setting and strategies for each player, the next step was to implement them. The focus was on the question of how fast a player can win. Enforcer has a sure victory, but how fast he can achieve it? We were interested in seeing how the number of rounds will change within the proven boundaries. As it was described and explained in the proof of Theorem 3.4, each player has predefined ways of choosing an edge in every round. Strategy for Avoider is completely deterministic, but for Enforcer, it is not as in some parts we have more than one option for the following move. Later, we will see how and does the results can be improved if we try to upgrade his strategy.

The strategy of each player consists of different possible moves defined by priority. Therefore, in the Enforcer's case, the situation is as follows:

He will try to claim an edge which will connect two opposite sides of the bipartition of one of the Avoider's connected component. In every move that is his primary goal. Sometimes he will be unable to do that. In those cases he will claim an arbitrary edge.

For Avoider, the situation is a little different. He has three possible moves defined by priorities and in every round, he will claim an edge by following one of these rules. When the moment comes that none of these moves are possible, at that time he is forced to choose the edge for which he will lose the game. His strategy can be explained in the following way:

His first choice, if possible, is to claim an edge across some existing bi-bunch. If that kind of edge is not free (all edges of that type are already claimed) or does not exist (e.g. in the first move it will not) then he will try to claim an edge whose vertices are untouched by that point. If neither of that is possible, he will claim an edge connecting two existing bi-bunches.

4.2 Python

The whole code was done in Python. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It has a simple syntax and his programs are clear and easy to read. It supports both procedural and object-oriented programming. Python supports modules and packages, which can be very helpful if you need reuse. [8,21]

Packages that were used the most in the programming process were described in the following paragraphs:

1. Numpy

NumPy stands for Numerical Python. It is the fundamental package for scientific computing and it is mostly used for working with arrays. Lists are slaw to process so NumPy gives as an array object that is up to 50 times faster than lists in Python. Many other packages are also built on top of this one.

2. Matplotlib

Matplotlib is a library used for visualization in Python. It is used for creating plots, histograms, bar charts, scatterplots, etc.

3. NetworkX

NetworkX is a library that is used the most in the programming of the game that we talked about earlier. It is a library for studying graphs and networks. NetworkX is a Python package for the creation, manipulation, and study of complex networks.

We want to use data as effectively as possible, so that is why is important to store it properly. Data structures can be classified in several ways and you can see this classification in Figure 4.1.

Using classes when coding was very useful because they are a way to define new sorts of stuff not previously used and implemented by other users.

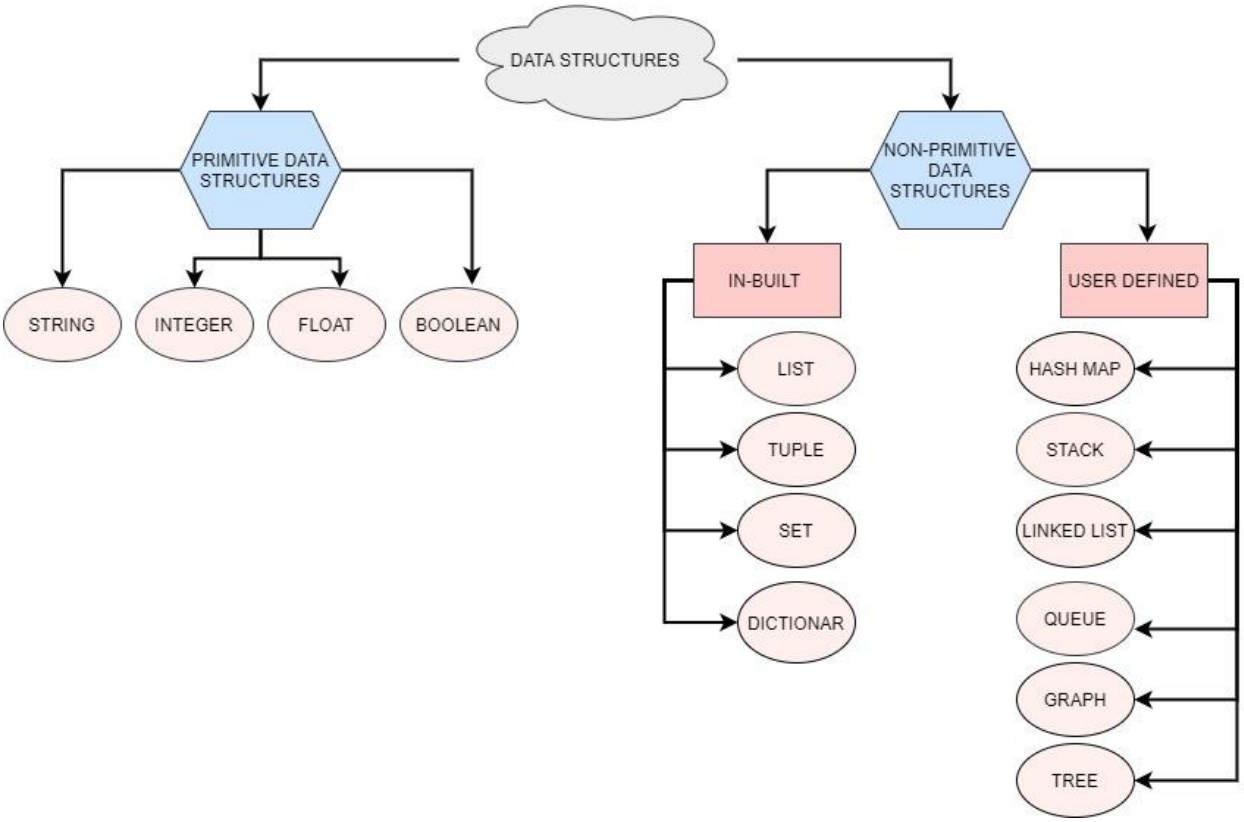

FIGURE 4.1 CLASSIFICATION OF DATA STRUCTURES

4.2.1 Game simulation code

For the beginning, let us briefly recall the game and players' strategies that are supposed to be implemented. The game is played on a complete graph on n vertices. Many different values of n were used. We have two opposing sides – Avoider and Enforcer. The game is played until Avoider claims an edge that is going to make his graph non-bipartite. Each player has its strategy. Avoider is trying to play as long as possible without compromising the bipartite property of his graph. On the other hand, Enforcer is trying to force the Avoider to claim exactly that kind of edge that is going to make his graph non-bipartite. The focus of our experiment was to see how fast Enforcer can win. The main question is how many rounds are going to pass until we get to the point where Avoider's every choice from all the remained edges is going to make his graph non-bipartite.

As we explained earlier in the paper, three different strategies for Enforcer have been implemented and analyzed. In accordance with these changes, it was necessary to adjust and change the code itself. However, some pieces of the code remained the same in each of the strategies and the way the data is stored has not changed.

For graph representation, NetworkX was used. The Non-bipartite game is always played on a complete graph. In addition to constructing graphs node by node or edge by edge,

they can also be generated using a constructive generator. In our case, we use generator `complete_graph(n)`, where `n` represents the number of vertices. Every edge was represented as a tuple with two elements. Throughout the whole game, it is necessary to follow unclaimed i.e. free edges as well as those previously claimed by each player. All of these were stored in different lists. We also developed a new class that we needed to store and manipulate objects called bi-bunches. The class consists of two lists that have no common elements and also have the same number of elements from the beginning until the end of the game. After creating this class, we also needed a list of all the bi-bunches created during the game.

To begin with, we needed a function that is going to tell us whether the Avoider's graph is bipartite or not. More details can be found in [10]. We proved earlier that this is equivalent to whether a graph contains no odd cycles. For the purpose of implementation, things written in [20] were very helpful. A round starts if the condition is satisfied, if it is not satisfied the game ends at that point. Pseudocode for the function that checks if a graph is bipartite follows under Algorithm 4.1.

```
INPUT: Graph G
OUTPUT: Boolean value - True if the given graph is bipartite, False
otherwise
1. V = {} -> dictionary where key is node index and value is in which
                              step the node was visited
2. Algo = []
3. FOR node in the list of nodes:
4.    IF node not in V:
5.          put in Algo (node,0)
6.    WHILE (there are elements in Algo):
7.          remove the first item from the list Algo
8.          r = removed item
9.          IF r[0] in V:
10.               IF (r[1] - V[r]) % 2 == 1 -> if this is true than a node
                                                has been revisited and we
                                                have a cycle
11.                     RETURN False
12.         ELSE:
13.               V[r[0]] = r[1]
14.               FOR node in list of nodes connected with removed node:
15.                     put in Algo (node, r[1]+1)
16.   RETURN True
```

ALGORITHM 4.1 FUNCTION THAT ANSWERS THE QUESTION OF WHETHER A GRAPH IS BIPARTITE OR NOT

Avoider's strategy is always the same. He wants to claim an edge across an existing bi-bunch. So, the first thing is to check whether a list with all the bi-bunches is empty or contains some elements. If it is not empty, we create all possible edges from existing bi-bunches and find one that is not claimed previously – if such edge exists. To create these edges, we wrote a function that is combining the elements from the two sets of a particular bi-bunch. Otherwise, if there is no matching edge or the list with bi-bunches is empty, the next step is to find two untouched vertices and to claim an edge incident with them. A vertex is untouched if it does not belong to any bi-bunch and all the edges incident with it are unclaimed. For checking whether a vertex is untouched, we also created a function that returns a Boolean value. If none of these two options were available, then we were checking if there are at least two bi-bunches and also whether there is at least one unclaimed edge that can connect two different bi-bunches. This pretty much summarizes the whole of Avoider's strategy. It is important to emphasize that after almost every move, it was necessary to make some changes to the existing bi-bunches or to create a new one. For that purpose, we created a few functions that merge the bi-bunches in a way defined in the proof of Theorem 3.4.

Let us now move on to Enforcer's strategies. In each of the three strategies we presented, his primary goal is the same. To recall, he needs to find an edge that connects two opposite sides of the bipartition of one of the Avoider's connected components. We created the function that returns a Boolean value that answers the question of whether such edges exist together with a list of all the edges that satisfy the condition. In Algorithm 4.2 we created pseudocode for the solution of this step in the strategy.

In Random strategy, if he was unable to claim such edge described in the previous paragraph, he will randomly choose one. For that, we used the choice() method that returns a randomly selected element from the specified sequence. That sequence in our case was a list of all unclaimed edges by that point. Changes on the bi-bunches had to be made depending on which edge was chosen. If the chosen edge is such that neither of its vertices belongs to any bi-bunch, a new bi-bunch was created and in that case, we also needed two untouched vertices to put in that bunch. To check these two things, we have defined functions. The first one that is answering the question of whether the chosen edge does not belong to any bi-bunch is called 'free'. And the other is called 'e2u' that stands for exist 2 untouched. Next, if Enforcer claims an edge such that one of its vertices belongs to some bi-bunch and the other one is untouched, that edge is named 'semi free'. In that case, some alterations are done on the existing bi-bunch from which the mentioned vertex is. And the last case is that both vertices of the edge belong to some bi-bunch. That edge we named 'occupied'.

In the Twin strategy, we used similar functions and approaches as in Avoider's strategy. And for The smallest strategy, instead of the choice() method, we introduce a function that we named 'smallest' that is returning the edge whose vertex degrees give the smallest value when added together. This degree refers to a subgraph of claimed edges.

```
INPUT: Graph G
OUTPUT: Boolean value - True if there exists at least one edge that
satisfies the condition (False otherwise)
      together with a list of all the edges that satisfy the condition
1. colors_of_nodes = {}
2. colors = ['red','blue']
3. DEF coloring (node, color):
4.    FOR neighbor IN the list of neighbors of G:
5.          color_of_neighbor = colors_of_nodes.get(neighbor, None)
6.          IF color_of_neighbor == color:
7.                REUTURN False
8.    RETURN True
9. DEF get_color_for_node(node):
10.   FOR color IN colors:
11.         IF coloring(node,color):
12.               REUTRN color
13. FOR node IN list of nodes in a DFS pre-ordering starting at source:
14.   colors_of_nodes[node] = get_color_for_node(node)
15. result = False
16. good_edges = []
17. con_com = connected_components of G
18. FOR i IN range(len(con_com)):
19.   component = con_com[i]
20.         FOR j in range(i+1, len(component)):
21.               IF (colors_of_nodes[component[i]]=='red' AND
      colors_of_nodes[component[j]]=='blue'
                     AND component[i],component[j] NOT in list of edges of
      G (or the opposite case of colors):
22.                    result = True
23.                    in list good_edges put (component[i],component[j])
24. IF (result):
25.   RETURN (result,good_edges)
26. ELSE:
27.   RETURN (result,'edges don't exist')
```

ALGORITHM 4.2 FUNCTION THAT ANSWERS THE QUESTION OF WHETHER THE PRIMARY STEP IN ENFORCER'S STRATEGY CAN BE DONE

5. Results

We have dealt with the analysis of this Non-bipartite game and within this chapter, we will talk about the results obtained in many different experiments. We are opposing strategies used to obtain these two theoretical limitations defined and proven in Theorem 3.4. We observe the duration of the game. The duration of the game is presented as the number of rounds required for Enforcer to win. The number of rounds if we follow the strategies explained before, should be at least $\frac{n^2}{8}+\frac{n-2}{12}$ and not more than $\frac{n^2}{8}+\frac{n}{2}+1$ where n is the number of vertices. Avoider's strategy is unique and unchanged in all experiments. On the other hand, trying to enhance Enforcer's play, we introduced three different strategies. After implementing these strategies and running many experiments, obtained results are presented in the following chapters.

5.1 'Random strategy'

The first strategy that we are going to analyze is the Random strategy. The first few experiments were done by following the steps defined below. Numbering indicates the priority in selecting an edge. The players will always try to make the first move defined in their strategies. If they are not able to do that, they will move to the next option and so on.

AVOIDER'S STRATEGY:

1. Claim an edge across an existing bi-bunch.
2. Claim an edge that will join two untouched vertices.
3. Clam an edge that will connect two bi-bunches.

ENFORCER'S STRATEGY:

1. Claim an edge that connects two opposite sides of the bipartition of one of the Avoider's connected components.
2. Randomly choose and claim an edge.

5.1.1 Experiments

These experiments were done on the complete graphs starting from the graph with 6 vertices and ending with the graph that contains 50 vertices, but only on the even ones. For every number of vertices, the experiment was run 20 times. The most important thing was to calculate the upper and lower bound for each number of vertices and to follow the number of rounds for every played game and how it is changed within the boundaries. In other words, the idea was to see what result the opposing strategies of these two players give.

Some general observations can be done on the obtained data. Firstly, we can tell that all the obtained numbers are between boundaries and that is something that was expected.

Secondly, it can be seen that for the smaller graphs number of rounds are almost the same in every experiment. For example, for the graph with 6 vertices, the number of rounds was always 7. As the graph was getting larger, the number of rounds started to change more drastically.

Some statistical measures were needed. Mean, mode and, median were calculated.

- *Mean* refers to the arithmetic mean – the sum of numbers divided by how many numbers are being summed.
- *Mode od modus* gives us the most frequently occurring or repetitive value in a range of given data.
- *Median* represents the value that separates the set and divides it into lover and higher half. It is a number in the middle of the ordered set of values.

For better visualization of the statistical measures and how they are different one from another, look at Figure 5.1.

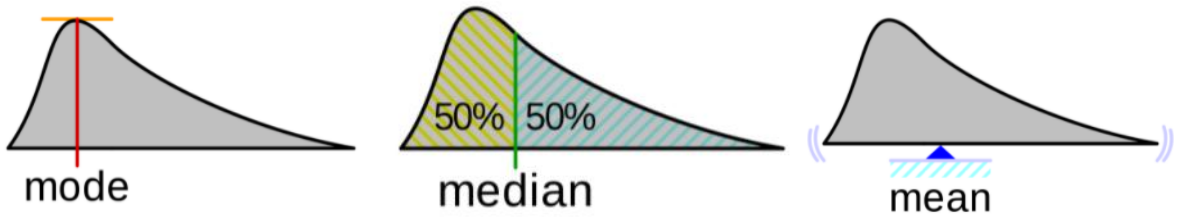

FIGURE 5.1 STATISTICAL MEASURES [18]

Now, let us take a look at Figure 5.2 to see our results. As it was already mentioned, for every graph size 20 games were played on it. We took the mode values to represent them on the plot. In Figure 5.2 in both the upper and lower bound and results we subtracted $\frac{n^2}{8}$ in order to emphasize differences between the values in the second-order term. Later in the paper, this way of presenting the results was always used for a clearer picture of the differences in results for different strategies.

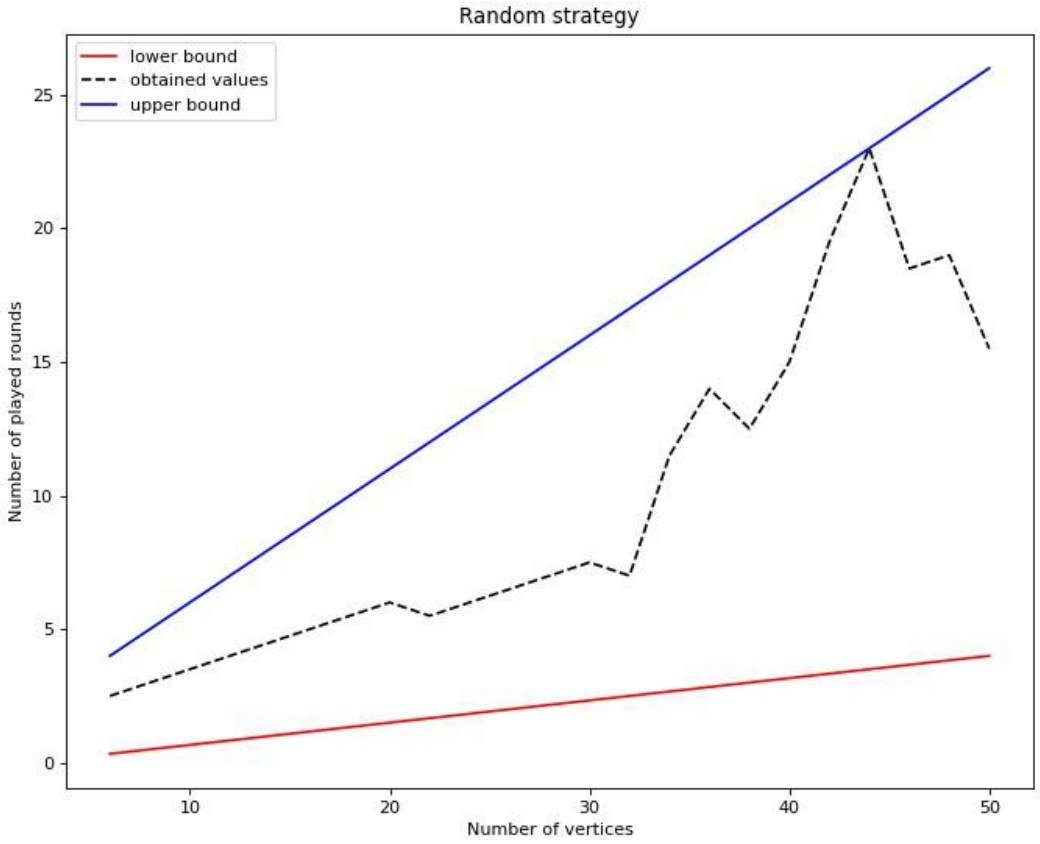

FIGURE 5.2 RESULTS FOR 'RANDOM STRATEGY' – 20 LAUNCHES ON GRAPHS FROM N=6 TO N=50 SUBTRACTED BY THE VALUE OF $\frac{n^2}{8}$

What can be concluded is that for the smaller n, the number of rounds was somewhere near the middle – between the upper and lower bound. As n was getting larger we can see that number of rounds was leaning towards the upper bound. That means that Avoider played the game better than Enforcer.

It seemed like Enforcer was at its best in the games played on the graphs from $n = 20$ to $n = 30$ because the curve is lowest there. Somewhere after $n = 30$, all obtained values are closer to the upper bound in comparison to the lower one. That leaves us with a question of whether and how we can change Enforcer's strategy to improve these results.

In Figure 5.2 mode values were represented, but we also calculated mean and median. In Table 5.1 you can find precise values for the upper and lower bounds.

Number of vertices	6	8	10	12	14	16	18	20	22	24	26
Lower bound	4.83	8.50	13.17	18.83	25.50	33.17	41.83	51.50	62.17	73.83	86.5
Upper bound	8.5	13	18.5	25	32.5	41	50.5	61	72.5	85	98.5
Mean	7	11	16	21.95	28.6	36.65	45.6	55.65	66.25	78.48	91.1
Mode	7	11	16	22	29	37	46	56	66	78	91
Median	7	11	16	22	29	37	46	56	66	78	91

28	30	32	34	36	38	40	42	44	46	48	50
100.17	114.83	130.5	147.17	164.83	183.5	203.17	223.83	245.5	268.17	291.83	316.5
113	128.5	145	162.5	181	200.5	221	242.5	265	288.5	313	338.5
105.15	120.1	135.85	156.4	174.9	194.85	216.05	238.3	260.55	282.5	306.85	331.4
105	120	135	156	176	193	215	240	265	283	307	328
105	120	136	156.5	175	195	215.5	238.5	260	283	307	330.5

TABLE 5.1 STATISTICAL MEASURES

From the given data, we can conclude that mean, modus, and median are not very different from each other. In fact, in numerous cases, they are the same.

At the end of this case, let us take a look at the plots in the following pages that are showing obtained results from done experiments for $n = 10$, $n = 30$, and $n = 50$.

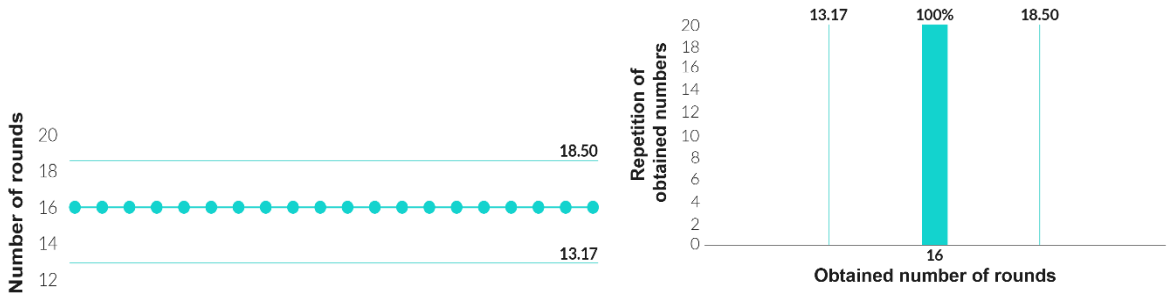

FIGURE 5.3 OBTAINED RESULTS APPLYING THE RANDOM STRATEGY FOR N = 10

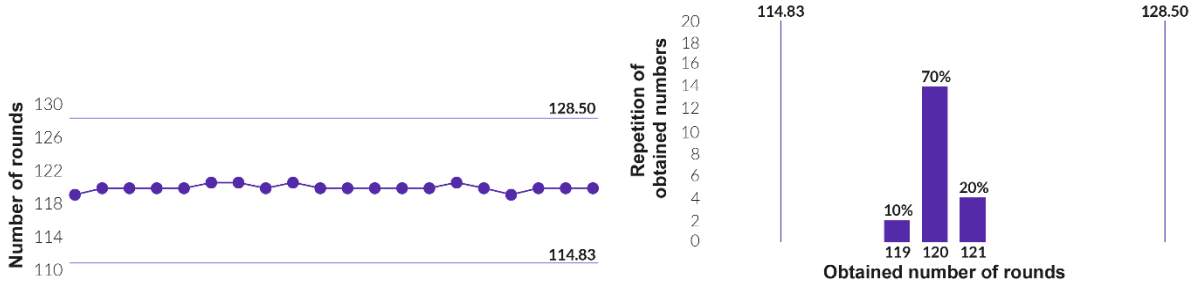

FIGURE 5.4 OBTAINED RESULTS APPLYING THE RANDOM STRATEGY FOR N = 30

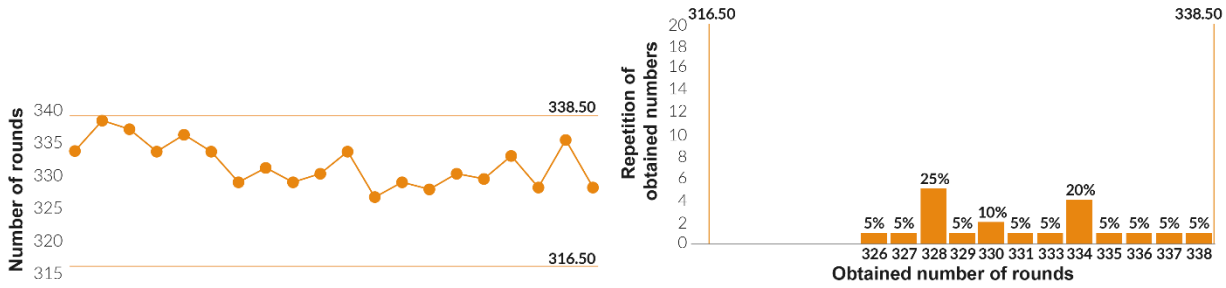

FIGURE 5.5 OBTAINED RESULTS APPLYING THE RANDOM STRATEGY FOR N = 50

One more thing that would also be interesting to see and can be helpful later when comparing the strategies is how far the average number of rounds was from lower and how far from the upper bound. We wanted to get a percentage of the interval in which the obtained value is located. We have done that by calculating the value of this formula $\frac{LG-LB}{UB-LB}$ where LG stands for the length of the game presented as the number of rounds, LB for the lower bound, and UB for the upper bound.

Number of vertices	The percentage of the interval in which the obtained value is located	Number of vertices	The percentage of the interval in which the obtained value is located
6	*59.09%*	*28*	*38.83%*
8	*55.56%*	*30*	*38.54%*
10	*53.13%*	*32*	*36.90%*
12	*50.54%*	*34*	*60.22%*
14	*44.29%*	*36*	*62.27%*
16	*44.47%*	*38*	*66.76%*
18	*43.46%*	*40*	*72.24%*
20	*43.68%*	*42*	*77.50%*
22	*39.52%*	*44*	*77.18%*
24	*41.34%*	*46*	*70.49%*
26	*38.33%*	*48*	*70.94%*
		50	*67.73%*

TABLE 5.2 WHERE IN THE INTERVAL ARE OBTAINED RESULTS

We can draw some conclusions by looking at the percentages in the table. The lower the percentage, the faster the Enforcer managed to win within the proven boundaries. The lowest percentage is 36.90% obtained for the graph with 32 vertices. Observing these 23 values for different graph sizes, for this '32-node' graph, the Enforcer achieved the fastest victory and played the best. The highest percentage was obtained for the '42-node' graph with the value of 77.50% and that is the best result in favor of Avoider. The average number of rounds is located at 54.48% of the interval. The number of rounds was never too close to the lower bound. We wanted to change that by improving Enforcer's strategy. Later in the paper, we will see if that was possible by comparing in detail obtained results.

5.1.2 Experiments on larger graphs

We saw how the number of rounds changed within the boundaries and how for the smaller graphs we got pretty similar values for every game that has been played, and for larger ones how these results differ one from another. The next idea was to make sure that these strategies work for the bigger graphs also – the ones with a larger number of vertices. Previously it has also been seen that for larger graphs, the number of rounds

was closer to the upper bounds, but the largest graph had 50 vertices. So, we thought it might be interesting to see how these numbers will behave for even larger graphs. Games were simulated for every even graph from 50 vertices to 100 vertices. In Figure 5.6 we can see the results. The same 'trick' was used – subtraction by the value of $\frac{n^2}{8}$ to get a better visualization of the data.

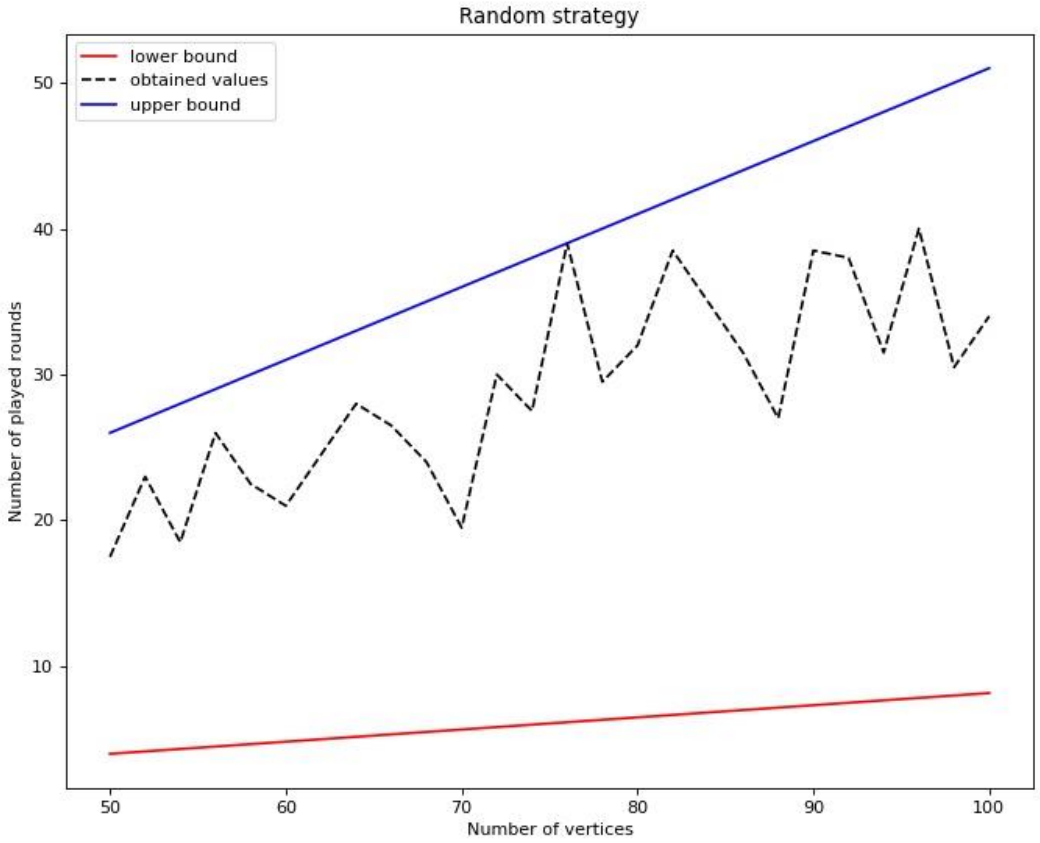

FIGURE 5.6 RESULTS FOR 'RANDOM STRATEGY' - GRAPHS FROM N=50 TO N=100 SUBTRACTED BY THE VALUE OF $\frac{n^2}{8}$

Firstly, we can see the curve is very close to the upper bound for almost every size of a graph starting from $n = 50$ and ending with $n = 100$. This is telling us that Avoider's strategy opposed to his opponent's strategy gives much better results. He was able to prolong the game almost to the maximum number of rounds in some cases.

To confirm what can be seen in Figure 5.6 just by looking at it, we created Table 5.3 which will tell us where in the interval obtained values are located. We used the same formula as before and obtained the following figures.

Number of vertices	The percentage of the interval in which the obtained value is located	Number of vertices	The percentage of the interval in which the obtained value is located
50	61.36%	*76*	100%
52	82.48%	*78*	68.81%
54	59.86%	*80*	73.91%
56	87.76%	*82*	90.09%
58	70.39%	*84*	77.88%
60	61.78%	*86*	66.22%
62	72.22%	*88*	52.42%
64	82.04%	*90*	80.60%
66	73.84%	*92*	77.22%
68	62.71%	*94*	59.09%
70	45.60%	*96*	78.14%
72	80.75%	*98*	53.57%
74	67.19%	*100*	60.31%

TABLE 5.3 WHERE IN THE INTERVAL ARE OBTAINED VALUES

From Table 5.3 we can see that almost all of the values are in the upper part of the interval (percentages greater than 50). The lowest is for the graph with 70 vertices with the value of 45.60%. And the greatest one is for the graph with 76 vertices where it is equated with the upper bound giving the result of 100%. The average position of the game length is at 71.01% of the interval.

5.1.3 Conclusions for 'Random strategy'

By observing the results for graphs starting with $n = 6$ and ending with $n = 50$ and 20 launches for each value of n, we can see that Avoider's strategy was better in some ways than Enforcer's because, within the interval in which the game had to end, Avoider managed to prolong the game for a long time. The average number of rounds was at 54.48% of the interval which is somewhere in the middle, a bit closer to the upper bound. And if we move our attention to graphs with $n = 50$ to $n = 100$ we can see that the average number of rounds was at 71.01% of the interval. We can conclude that for the larger graphs the curve is moving more and more to the upper bound which means that Enforcer was slower and slower in achieving the win. This is not an unexpected result, because if we recall the strategies, we can see that Avoider's strategy was more detailed and not one move was based on an arbitrary choice of an edge. On the other side, Enforcer's strategy consisted of only one well-defined step. He followed only one rule and

if that was not possible, he would randomly choose an edge. These edges were marked as possible bad ones because they were probably going to ruin the strategy. This is a part of the game that seemed like it needs an improvement. Enforcer's strategy must be defined in a more deterministic way in order to get better results. We were hoping that these improvements would shift the curve down. And so the next strategy called Twin was born.

5.2 'Twin strategy'

The first logical improvement of Enforcer's strategy was to define and prioritize his moves in a way that will 'spoil' the Avoider's strategy.

This improvement was based on 'blocking' those moves interesting to Avoider. Recall that Enforcer has the same primary goal in choosing an edge in each of these simulations. The other moves in this particular strategy that we named Twin are similar to Avoider's moves. By following these steps when choosing an edge, he might be able to finish the game earlier than before. New strategy for Enforcer follows:

1. Claim an edge that connects two opposite sides of the bipartition of one of the Avoider's connected components.
2. Claim an edge that connects two untouched vertices. (Just a reminder, he will need two more untouched vertices to put in a bi-bunch based on the rules for creating and manipulating with bi-bunches)
3. Claim an edge that will connect two bi-bunches.
4. If none of these options is possible, choose randomly.

Now, looking at the two strategies, they are pretty similar. It seems like this could help Enforcer to win the game faster. Let us see the results of experiments done in the same way as for 'Random strategy' and try to compare them. The main question is: Was Avoider forced by Enforcer to claim an edge that will end the game sooner than before?

5.2.1 Experiments

After the done changes on Enforcer's strategy, we would like to see whether these numbers of rounds are closer to the lower bounds. That would be a good indicator that we are on a right track. In Figure 5.7 we can see what we got by applying an improved strategy for Enforcer. The mode values were used, the same as for the Random strategy.

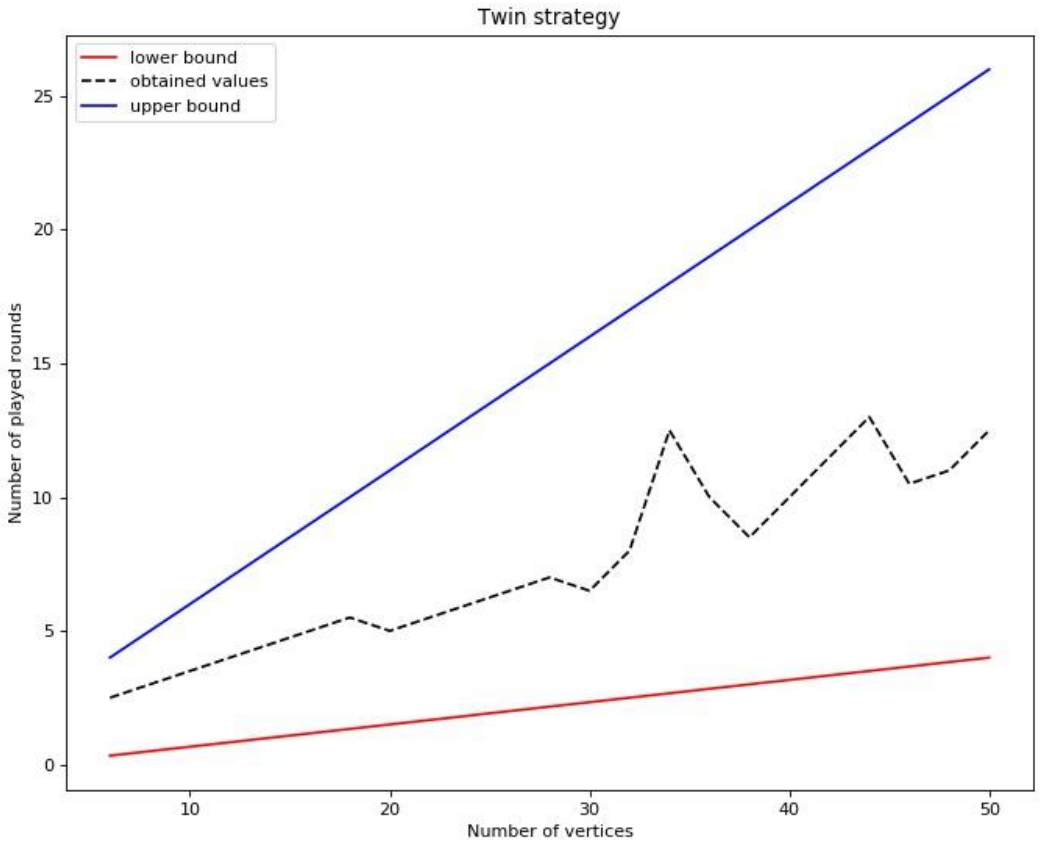

FIGURE 5.7 RESULTS FOR 'TWIN STRATEGY' – 20 LAUNCHES ON GRAPHS FROM N=6 TO N=50 SUBTRACTED BY THE VALUE OF $\frac{n^2}{8}$

By looking at Figure 5.7 we can see that the black curve that represents mode values for done experiments is somewhere in the middle of the interval. Even more, it is leaning towards the lower bound. If we compare it with Figure 5.2 we can definitely see that some improvements for Enforcer have been made.

Based on this figure we can say that the curve, which represents mode values from our experiments, is shifted downward. And if we briefly recall the plots for Random strategy and the cases for n values around 45 when the mode was almost equal to the upper bound and compare it to this case, we can conclude that it is drastically lower and much closer to the lower bound, which goes in favor of Enforcer, than before.

It would be convenient to have a table with statistical measures for these experiments also to compare it with the first one.

Number of vertices	6	8	10	12	14	16	18	20	22	24	26
Lower bound	4.83	8.50	13.17	18.83	25.50	33.17	41.83	51.50	62.17	73.83	86.5
Upper bound	8.5	13	18.5	25	32.5	41	50.5	61	72.5	85	98.5
Mean	7.05	11	16.1	21.85	28.75	36.8	45.35	55.45	66.3	78.3	91.15
Mode	7	11	16	22	29	37	45	55	66	78	91
Median	7	11	16	22	29	37	45	55	66	78	91

28	30	32	34	36	38	40	42	44	46	48	50
100.17	114.83	130.5	147.17	164.83	183.5	203.17	223.83	245.5	268.17	291.83	316.5
113	128.5	145	162.5	181	200.5	221	242.5	265	288.5	313	338.5
105.05	119.55	134.45	154.9	172.8	192.6	213.65	234.65	256.6	278.85	304.7	328
105	119	136	157	172	189	210	232	255	275	299	325
105	119.5	135.5	155	172	192.5	213	233.5	255	278	303	328

TABLE 5.4 STATISTICAL MEASURES FOR TWIN STRATEGY

We can make some conclusions about these experiments, not comparing them with the previous ones. For smaller graphs, it can be concluded that the number of rounds for every played game is almost the same. On the other hand, for the bigger graphs, we can see that numbers are changing. Also, as for the previous strategy, mean, mode, and median are very similar, in a lot of cases they are equivalent. In the pages that follow you can find plots to get a better understanding of where these numbers are and how they are changing.

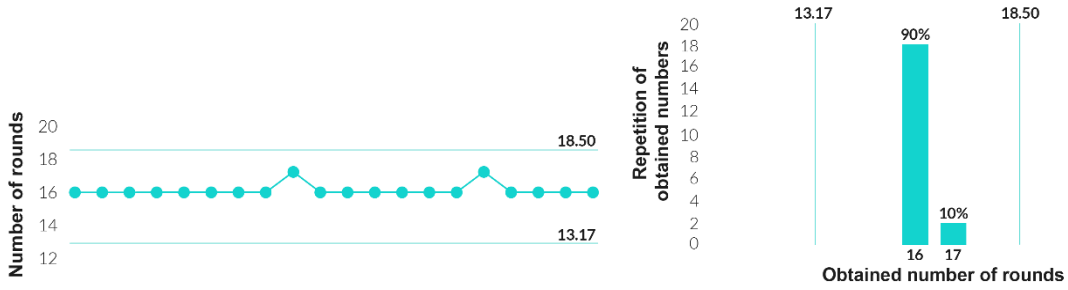

FIGURE 5.8 OBTAINED RESULTS APPLYING THE TWIN STRATEGY FOR N = 10

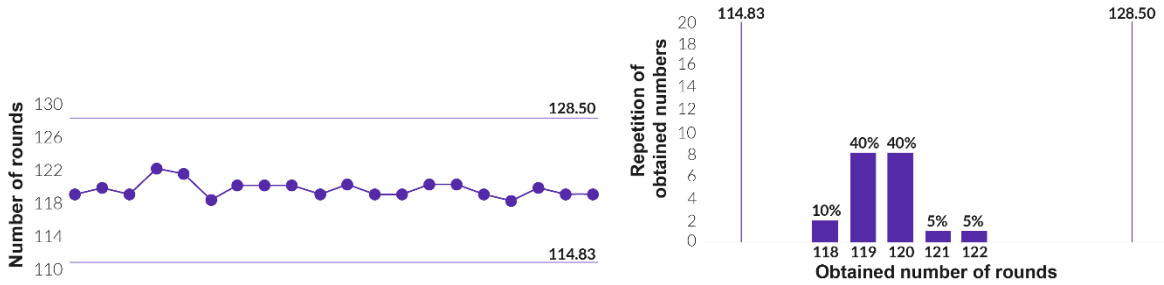

FIGURE 5.9 OBTAINED RESULTS APPLYING THE TWIN STRATEGY FOR N = 30

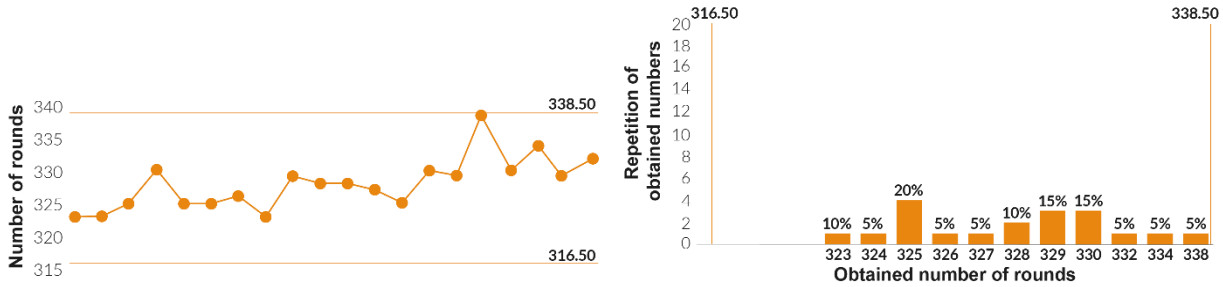

FIGURE 5.10 OBTAINED RESULTS APPLYING THE TWIN STRATEGY FOR N = 50

As we talked about it earlier, by calculating where within the interval obtained values are located and comparing them with the same results for other strategies, we can see the differences and how each change moves the number of rounds whether up or down the interval. In Table 5.5 we calculated the percentage of the interval in which the obtained values for every graph size are located.

Number of vertices	The percentage of the interval in which the obtained value is located	Number of vertices	The percentage of the interval in which the obtained value is located
6	*60.45%*	*28*	*38.05%*
8	*55.56%*	*30*	*34.51%*
10	*55.00%*	*32*	*34.14%*
12	*48.92%*	*34*	*50.43%*
14	*46.43%*	*36*	*49.28%*
16	*46.38%*	*38*	*53.53%*
18	*40.58%*	*40*	*58.79%*
20	*41.58%*	*42*	*57.95%*
22	*40.00%*	*44*	*56.92%*
24	*40.00%*	*46*	*52.54%*
26	*38.75%*	*48*	*60.79%*
		50	*52.27%*

TABLE 5.5 WHERE IN THE INTERVAL ARE OBTAINED RESULTS

Some conclusions can be drawn from these results. Firstly, we can find the smallest value within this interval. That value is 34.14% and it is obtained for the graphs of size 32. It is interesting to notice that for the Random strategy, the lower percentage was also obtained for the graphs of size 32. The highest percentage was obtained for the '48-node' graph and that value was 60.79%. The highest value for the Random strategy was 77.50%. This can be an indicator that we are on the right track because we are moving the curve to the lower bound of the interval and that is what we wanted to achieve by introducing these improvements in Enforcer's strategy. The average number of rounds is located at 48.38% on the interval. That is also an improvement, comparing it with the 54.48% obtained for the Random strategy.

5.2.2 Experiments on larger graphs

Based on the conclusion that more differences have been seen in the larger graphs after the changes on Enforcer's strategy, we were curious to see if improvements are going to be more visible for even larger graphs. Let us start with Figure 5.11 to visualize the obtained figures.

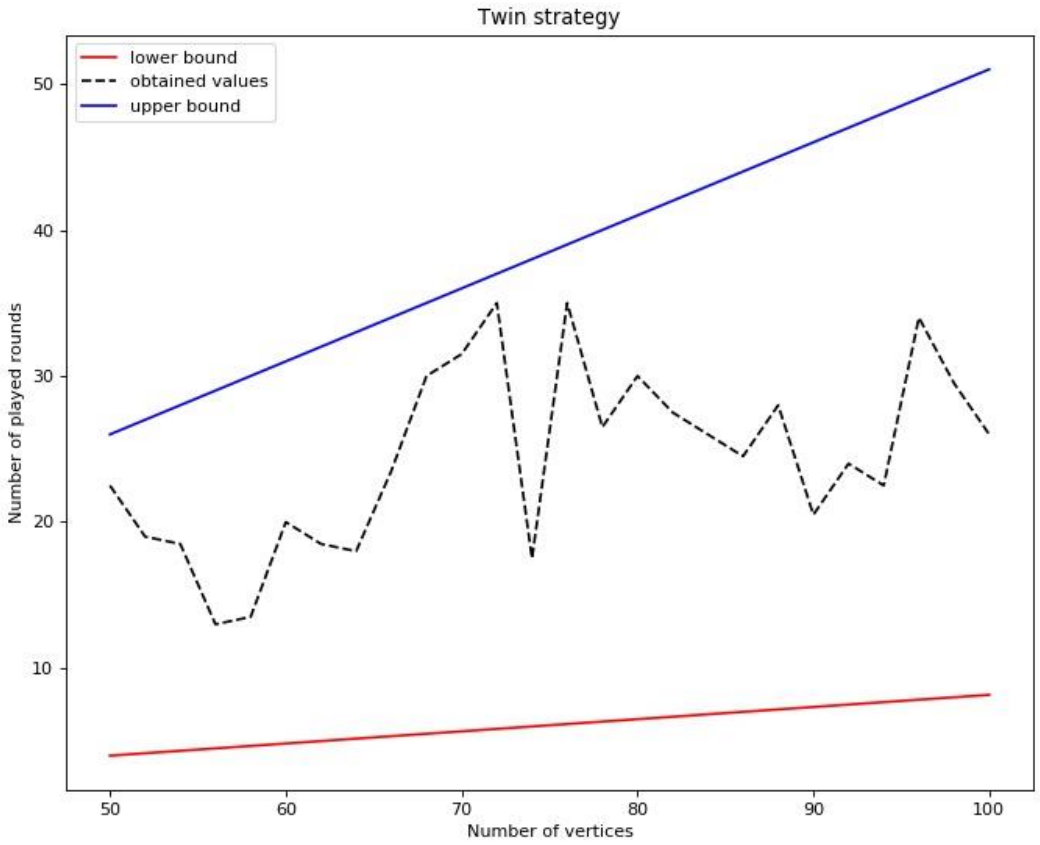

FIGURE 5.11 RESULTS FOR 'TWIN STRATEGY' - GRAPHS FROM N=50 TO N=100 SUBTRACTED BY THE VALUE OF $\frac{n^2}{8}$

From Figure 5.11 we can notice that in some parts, we can see that the curve is leaning towards the lower bound and that was not the case in the Random strategy. Somewhere around $n = 70$ the curve is almost 'glued' to the upper bound and no improvement should be expected there, but for the values of n close to 60 and also 90, it seems that some improvements were achieved. The best result was obtained for $n = 56$ and $n = 58$ and after that $n = 74$. Pretty good results are also visible for $n = 90$, $n = 92$ and $n = 94$. The whole curve is shifted towards the lower bound, for some values of n more, for some less, but in general, it is lower.

The same as for the Random case, to confirm what can be seen in Figures 5.11 we created Table 5.6 which will tell us exactly where within the interval these obtained lengths of the games are located.

Number of vertices	The percentage of the interval in which the obtained value is located	Number of vertices	The percentage of the interval in which the obtained value is located
50	84.09%	76	87.82%
52	64.96%	78	59.90%
54	59.86%	80	62.12%
56	34.69%	82	58.96%
58	34.87%	84	53.00%
60	57.96%	86	47.30%
62	50.00%	88	55.07%
64	46.11%	90	34.05%
66	63.37%	92	41.77%
68	83.05%	94	36.78%
70	85.16%	96	63.56%
72	93.58%	98	51.19%
74	35.94%	100	41.63%

TABLE 5.6 WHERE IN THE INTERVAL ARE OBTAINED RESULTS

In Table 5.6 we can see that there is a lot more than one value located in the lower part of the interval as was the case with the Random strategy. The lowest percentage is obtained for the '90-node' graph with the amazing result of 34.05%. Just to compare, for the graph of the same dimension in the Random strategy, the length of the game was at 80.60% of the interval. This is a great improvement. The average position of the game length is at 57.42% of the interval.

5.2.3 Conclusions for 'Twin strategy'

After various analyzes regarding the Twin strategy, it can be concluded that it gives quite good results. By applying this strategy, Enforcer managed to match Avoider's brilliant strategy, which we know gives great results for him. Compared with the Random strategy, it gives much better results for Enforcer in the form of shortening the game i.e. the speed of his victory. Specific figures speak in favor of this and we see that in the case of 20 runs for graphs up to $n = 50$ the average position of the length of the game within the interval is at 48.38%. In the case of graphs larger than 50 up to those with 100 vertices, this number is 57.42% which is a significant improvement in comparison with 71.01% obtained for the Random strategy.

5.3 'The smallest'

For The smallest strategy we chose to do the following changes in Enforcer's strategy:

1. He tries to find and claim an edge that connects two opposite sides of the bipartition of one of the Avoider's connected components.
2. He claims an edge whose vertex degrees give the smallest value when added together.

This strategy is different from the others because this second option is always possible to achieve and also there is almost no random part as in the previous two. The only case where we have a random part is if there are several edges with the same sum after adding the degrees of their vertices.

Firstly, we run 20 experiments for every value of n, starting from 6 and ending with 50. We have chosen this approach because it would be easy to compare it with the previous two strategies. Figure 5.12 is showing the results of this strategy.

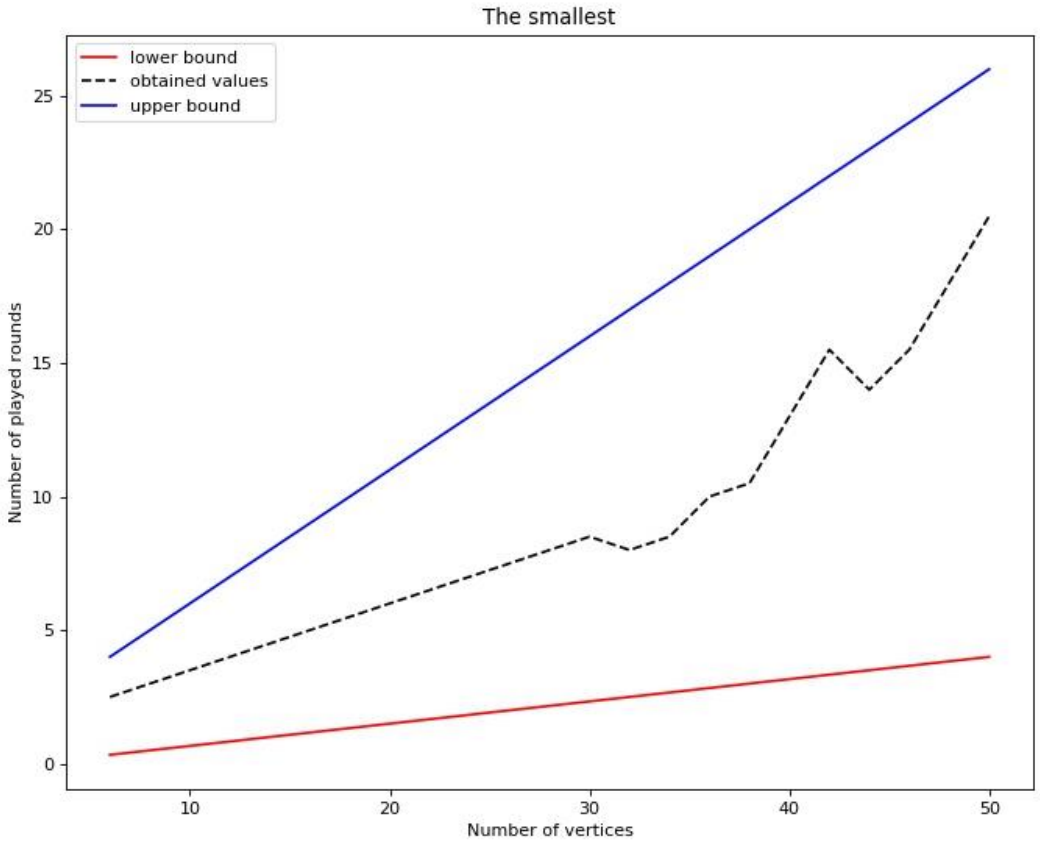

FIGURE 5.12 RESULTS FOR 'THE SMALLEST' STRATEGY – 20 LAUNCHES ON GRAPHS FROM N=6 TO N=50 SUBTRACTED BY THE VALUE OF $\frac{n^2}{8}$

Simply by looking at Figure 5.12 we can say that the black curve, which shows the modes of obtained values, is somewhere in between those boundaries. For n's smaller than 30 the result is stable somewhere in the middle of the interval. After that, it starts to move in the wanted direction but not for long. It seems like it starts to move more to the upper part

of the interval somewhere around $n = 45$. Based on that, we can assume that for higher values of n it will get closer to the upper bound, but it does not have to be that way.

In Table 5.7 we can see more clearly the things that we mentioned and saw in Figure 5.12.

Number of vertices	The percentage of the interval in which the obtained value is located	Number of vertices	The percentage of the interval in which the obtained value is located
6	*52.27%*	*28*	*42.34%*
8	*54.44%*	*30*	*41.83%*
10	*51.25%*	*32*	*40.34%*
12	*52.16%*	*34*	*43.26%*
14	*49.29%*	*36*	*49.28%*
16	*47.02%*	*38*	*57.06%*
18	*46.92%*	*40*	*59.35%*
20	*43.68%*	*42*	*62.23%*
22	*44.35%*	*44*	*60.51%*
24	*43.58%*	*46*	*68.03%*
26	*42.50%*	*48*	*66.46%*
		50	*72.27%*

TABLE 5.7 WHERE IN THE INTERVAL ARE OBTAINED RESULTS

The lowest percentage is 40.34% for $n = 32$, the highest is 72.27% for $n = 50$ and the average is 51.76%. This result is pretty similar to one obtained for the Random strategy.

Some assumptions about the future curve shifting were made, but they do not necessarily need to be accurate. We wanted to be sure, so experiments were done for larger graphs also, but only one game per different value of n.

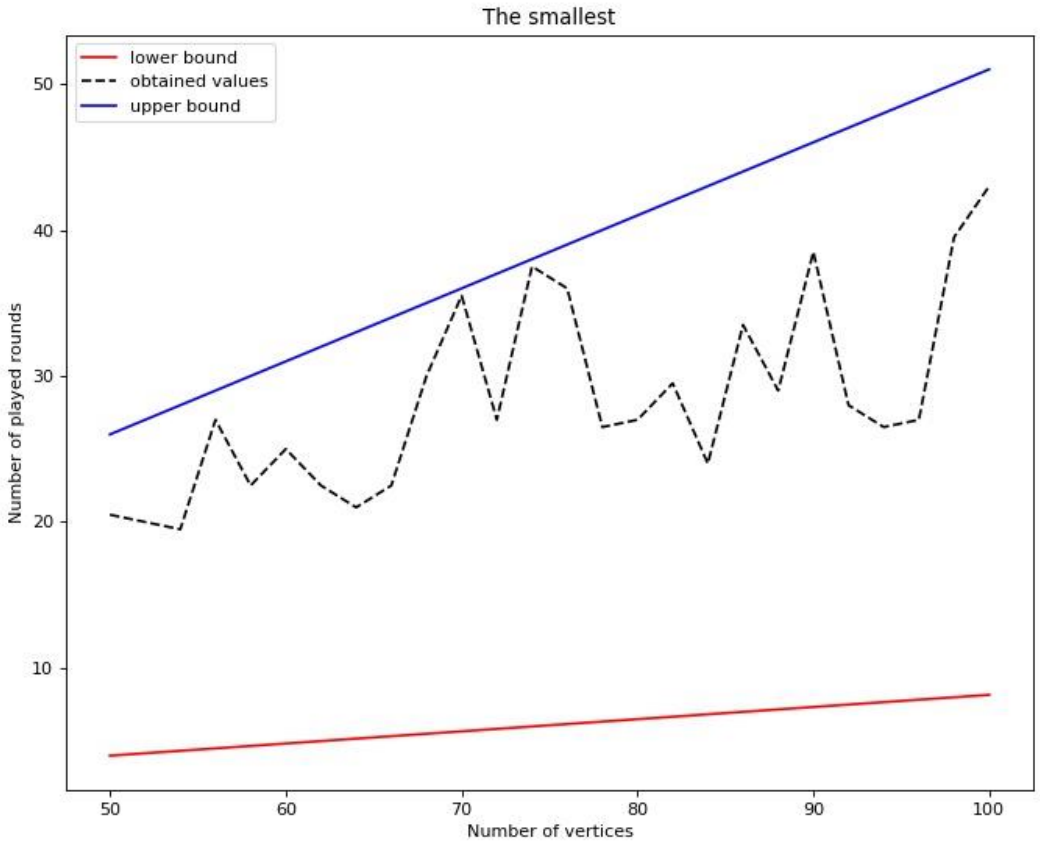

FIGURE 5.13 RESULTS FOR 'THE SMALLEST' STRATEGY - GRAPHS FROM N=50 TO N=100 SUBTRACTED BY THE VALUE OF $\frac{n^2}{8}$

Our assumptions were partially correct. We assumed that the values would go higher and higher towards the upper bound. We did not have constant upward movement after $n = 50$ but almost all the values were in the upper part of the interval. Applying this strategy we failed to move the curve down compared to the Random strategy.

As for all previous strategies, we created Table 5.8 to have an even clearer picture of how the duration of the game moved with increasing graph size.

Number of vertices	The percentage of the interval in which the obtained value is located	Number of vertices	The percentage of the interval in which the obtained value is located
50	75.00%	76	90.86%
52	69.34%	78	59.90%
54	64.08%	80	59.42%
56	91.84%	82	64.62%
58	70.39%	84	47.47%
60	77.07%	86	71.62%
62	64.81%	88	57.71%
64	56.89%	90	80.60%
66	59.88%	92	51.90%
68	83.05%	94	46.69%
70	98.35%	96	46.56%
72	67.91%	98	75.00%
74	98.44%	100	81.32%

TABLE 5.8 WHERE IN THE INTERVAL ARE OBTAINED RESULTS

The average percentage of the interval where the number of rounds is located is 69.64%. All the figures are pretty high. Given that our wish was for these numbers to be as low as possible, it can be said that this strategy is not the best option to oppose Avoider's strategy.

5.4 Comparison of strategies

The previous paragraphs present the results for three different strategies for Enforcer that are opposed to the fixed Avoider's strategy. After analyzing each strategy and doing some comparisons with others it could be concluded which one gave us the best results. Enforcer's goal was to end the game as soon as possible. The only way to do that was to force Avoider to choose an edge that would spoil the bipartition of his subgraph consisting of pre-claimed edges. For each size of a graph, we calculated the upper and lower bounds. By doing so, we got an interval and all obtained values had to be within it. We followed the curve movements of the results representing the length of the game. We took the Random strategy as the initial one and looked at whether the curve shifts downwards or upwards. Moving the curve downwards would mean that we have made some improvements.

It would be convenient to have a figure that will have the curves for all three strategies. In that figure, we could clearly see which strategy gave the best result simply by observing which one is the lowest and closest to the lower bound. So, we created Figure 5.14 for that purpose.

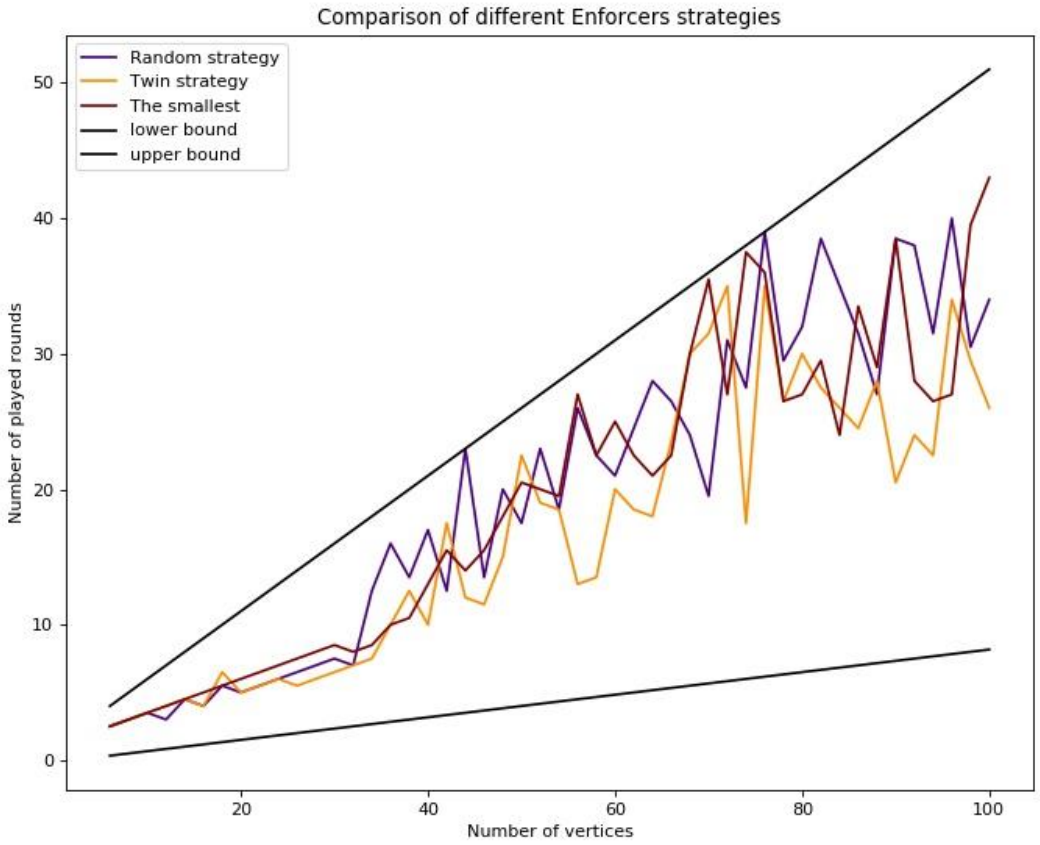

FIGURE 5.14 COMPARISON OF ALL THREE STRATEGIES

From Figure 5.14 we can see that the yellow curve which represents the Twin strategy is the lowest. The Random and The smallest strategies have a lot of overlap and give similar results. Let us take a look at Table 5.9 with a few more parameters for comparison.

Strategies	The location of the shortest game within the interval	The location of the longest game within the interval	The average location within the interval
Random strategy	*31.03%*	*100%*	*62.45%*
Twin strategy	*29.17%*	*93.58%*	*51.47%*
The smallest	*37.93%*	*98.44%*	*60.78%*

TABLE 5.9 COMPARISON OF ALL THREE STRATEGIES

Based on Figure 5.14 and Table 5.9, we can definitely conclude that by applying the Twin strategy, Enforcer managed to win the game the fastest. As previously stated the remaining two strategies give similar results. The fastest victory for Enforcer by applying the Random strategy is faster than the one applying The smallest strategy. However, if we take a look at the average duration of the game and where within the interval it is located, we can see that The smallest strategy gives a slightly better result.

6. Conclusion and future work

6.1 Summary and conclusion

In the Avoider-Enforcer game called the Non-bipartite game, the experiments were done using different strategies for Enforcer and the fixed one for Avoider. All the obtained numbers were inside the boundaries and that was also a piece of evidence in the form of an experiment that the boundaries were well defined as it was also proven. However, it seemed like there is room for improvement in Enforcer's strategy because a lot of his moves were arbitrarily chosen. We tried three different strategies for him and we monitored how they would behave when opposed to Avoider's strategy. The strategy that we called Twin, one that is trying to 'block' desired moves of Avoider, proved to be the best. These improvements that we got made Enforcer's win faster than before which is exactly what we wanted to achieve.

6.2 Future work

This field is not explored enough so a lot of different things can be done in the future. The first thing that seems interesting is to change the priority of the moves for the players. For example, the first desired move for Avoider could be to claim an edge that connects two untouched vertices and after that to claim an edge across existing bi-bunch. These changes might also affect the theorem and its boundaries but it would be interesting to see in what way these number of rounds would go. Will it be improvement or deterioration? Also, because of technical reasons, the biggest graph used for these experiments contains 100 vertices. It would be nice to see how these changes and improvements that were made would behave for larger graphs e.g. graph with 1000 vertices.

Bibliography

[1] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, Fast winning strategies in Avoider-Enforcer games, Graphs and Combinatorics 25 (2009), 533-544.

[2] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, Positional games, Birkhäuser, 2014.

[3] D. Hefetz, M. Krivelevich, and T. Szabó, Avoider-Enforcer games, Journal of Combinatorial Theory, Ser. A, 114 (2007), 840-553.

[4] V. Anuradha, C. Jain, J. Snoeyink, T. Szabó, How long can a graph be kept planar? *Electronic Journal of Combinatorics*, 15 (2008), N14.

[5] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, Avoider-Enforcer: The rules of the game, Journal of Combinatorial Theory, Ser. A Volume 117, Issue 2, February 2010, 152-163.

[6] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, The connectivity game, In: Positional Games. Oberwolfach Seminars, vol 44. Birkhäuser, Basel. May 2014.

[7] J.A.Bondy and U.S.R. Murty, Graph theory with applications, QA166.B67, printed in the USA, 1979, 75-29826.

[8] M.Lutz, Learning Python, fourth edition, Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, September 2009.

[9] J.A.Bondy and U.S.R. Murty, Graph theory, Mathematics Subject Classification (2000): 05C; 68R10, 2008.

[10] Check whether a given graph is Bipartite or not, https://www.geeksforgeeks.org/bipartite-graph/ [accessed: May 2021].

[11] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, Fast winning strategies in Maker-Breaker games, Journal of Combinatorial Theory, Series B, 99 (2009) 39–47.

[12] J. Beck, On positional games, *J. of Combinatorial Theory, Ser. A*, 30 (1981), 117-133.

[13] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, Fast winning strategies in positional games, Electronic Notes in Discrete Mathematics, volume 29, 15 August 2007, 213-217.

[14] J. Beck, Random graphs and positional games on the complete graph, Annals of Discrete Mathematics 28 (1985), 7-13.

[15] D.Hefetz, M.Krivelevich, M.Stojaković and T.Szabó, Planarity, colorability and minor games, *SIAM Journal on Discrete Mathematics*, 22 (2008), 194-212.

[16] P.Danziger, Colouring, January 2008.

[17] Manthel's theorem, https://math.stackexchange.com/questions/628877/mantels-theorem-proof-verification, [accessed: May 2021].

[18] "Wikipedia" Statistical measures, https://en.wikipedia.org/wiki/Mode_(statistics), [accessed: May 2021].

[19] "Wikipedia" Positional games, https://en.wikipedia.org/wiki/Positional_game [accessed: May 2021].

[20] Checking if a graph has a cycle of odd length, https://www.geeksforgeeks.org/check-graphs-cycle-odd-length/, [accessed: May 2021].

[21] "Wikipedia" Python programming language, https://en.wikipedia.org/wiki/Python_(programming_language), [accessed: May 2021].

[22] "Wikipedia" Turán's theorem, https://en.wikipedia.org/wiki/Tur%C3%A1n%27s_theorem [accessed: May 2021].

[23] V. Chvátal and P. Erdös, Biased positional games, Annals of Discrete Mathematics 2 (1978), 221-228.

BIOGRAPHY

Aleksandra Hajder was born on April 15, 1995 in Zrenjanin. She finished elementary school "2. oktobar", music school "Josif Marinković" where she played piano, and the School of economics "Jovan Trajković" in Zrenjanin. She received her Bachelor's degree in Applied Mathematics in 2018 at the Faculty of Sciences, University of Novi Sad. The same year she continued her Master studies in the field of Data Science at the same faculty. Undergraduate studies were mainly oriented toward finances. During her studies, she did an internship in Erste Bank Serbia in the Audit sector. She currently works at the fintech company FIS.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR
Identifikacioni broj:
IBR
Tip dokumentacije: monografska dokumentacija
TD
Tip zapisa: tekstualni štampani materijal
TZ
Vrsta rada: master rad
VR
Autor: Aleksandra Hajder
AU
Mentor: dr Miloš Stojaković
MN
Naslov rada: Analiza strategija za brze pobede u Avoider-Enforcer "Non-bipartite" igri
NR
Jezik publikacije: engleski
JP
Jezik izvoda: e
JI
Zemlja publikovanja: Republika Srbija
ZP
Uže geografsko područje: Vojvodina
UGP
Godina: 2021.
GO
Izdavač: autorski reprint
IZ
Mesto i adresa: Novi Sad, Trg Dositeja Obradovića 4
MA
Fizički opis rada: 6 poglavlja, 57 strana, 23 lit. citata, 22 grafika, 9 tabela
FO
Naučna oblast: matematika
NO
Naučna disciplina: primenjena matematika
ND
Ključne reči: Teorija grafova, Pozicione igre, Avoider-Enforcer igre, Non-bipartite igra
UDK
Čuva se: u biblioteci Departmana za matematiku i informatiku, Prirodno-matematičkog fakulteta, u Novom Sadu
CU
Važna napomena:
VN

Izvod: Pozicione igre se često igraju na grafovima različitih vrsta. U ovim igrama postoje dva igrača – Avoider I Enforcer, čije se strategije suprotstavljaju. Najpopularnija igra ove vrste je Iks-Oks I njeno višedimenzionalno uopštenje. Igra koja je u fokusu ovog rada se zove „Non-bipartite" igra. Ona se igra na kompletnim grafovima. Igrači naizmenično biraju grane iz grafa. Avoider pokušava da izbegne stvaranje nebipartitnog podgrafa. S druge strane, Enforcer pokušava da natera Avoider-a da uradi upravo to. Naš cilj je da potvrdimo da će se igra odigrati unutar dokazanih granica, ali i da vidimo gde se tačno nalazi trajanje igre u okviru dobijenog intervala ukoliko se oba igrača drže svojih optimalnih strategija. Avoider-ova strategija je detaljno izložena, a za Enforcer-a smo pokušali da postignemo poboljšanja menjajući mu deo strategije koji je bio definisan kao proizvoljan. U budućnosti bi bilo zanimljivo videti kako trajanje igre varira prilikom promena prioriteta poteza za svakog igrača. Ostavljeno je mnogo prostora za buduća istraživanja ove igre, ali i uopšteno Pozicionih igara.

IZ
Datum prihvatanja teme od strane NN veća:
DP
Datum odbrane:
DO
Članovi komisije:
KO
Predsednik: dr Dejan Vukobratović, redovni profesor FTN-a
Mentor: dr Miloš Stojaković, redovni profesor PMF-a
Član: dr Dušan Jakovetić, vanredni professor PMF-a

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
KEY WORDS DOCUMENTATION

Accession number:
ANO
Identification number:
INO
Document type: monograph type
DT
Type of record: printed text
TR
Contents code: master thesis
CC
Author: Aleksandra Hajder
AU
Mentor: PhD Miloš Stojaković
MN
Title: Analysis of fast winning strategies in Avoider-Enforcer "Non-bipartite" game
XI
Language of text: English
LT
Language of abstract: e
LA
Country of publication: Republic of Serbia
CP
Locality of publication: Vojvodina
LP
Publication year: 2021.
PY
Publisher: author's reprint
PU
Publ. place: Novi Sad, Trg Dositeja Obradovića 4
PP
Physical description: 6 chapters, 57 pages, 23 references, 22 figures, 9 tables
PD
Scientific field: mathematics
SF
Scientific discipline: applied mathematics
SD
Key words: Graph theory, Positional games, Avoider-Enforcer games, Non-bipartite game
UC
Holding data: Department of Mathematics and Informatics' Library, Faculty of Sciences, Novi Sad
HD
Note:

N

Abstract: Positional games are often played on different types of graphs. They involve two players whose goals oppose. The most popular game of this type is Tic-Tac-Toe and its higher-dimensional generalizations. The game that is the focus of this paper is called the Non-bipartite game, played on the complete graph. Players are called Avoider and Enforcer. Their names say a lot about their goals. Avoider is trying to avoid creating a non-bipartite subgraph while Enforcer is trying to enforce Avoider to do exactly that. Our goal is to verify that the game is going to be played within the proven boundaries and to see where exactly is the duration of the game when both players stick to their optimal strategies. Avoider's strategy was laid out in detail, but for Enforcer, we have tried to make some improvements. In the future, it would be interesting to see how the duration of the game fluctuates when we change the priorities of the moves for each player. There is a lot of space for future investigation of this particular game, but also in general of positional games.

AB

Accepted by the Scientific Board on:

ASB

Defended:

DE

Thesis committee:

DB

Chair: PhD Dejan Vukobratović, full professor
Mentor: PhD Miloš Stojaković, full professor
Member: PhD Dušan Jakovetić, associate professor