**Saint Petersburg state University**

**Department of Mathematical Game Theory and Statistical Decisions**

Methods of Constructing a Characteristic Function in a Network Game

Master's thesis

**Semenov Valery Gennadievich**

Specialization 01.04.02

Applied Mathematics and Informatics

Master’s Program Game Theory and Operations Research

Scientific adviser,

doctor of physical-Mat.sciences',

professor

Petrosjan L. A.

Saint Petersburg

2019

Content

[Introduction 3](#_Toc7002921)

[Problem statement 4](#_Toc7002922)

[Literature review 5](#_Toc7002923)

[Chapter 1 A two-stage network game 6](#_Toc7002924)

[1.2 The first stage of the game 6](#_Toc7002925)

[1.3 The second stage of the game 7](#_Toc7002926)

[1.4 Two problems of maximization 8](#_Toc7002927)

[Chapter 2 The problem of maximization with the selected controls 9](#_Toc7002928)

[2.1 Algorithm trying all possible assignments of connections 9](#_Toc7002929)

[2.2 The idea of the algorithm trying all possible assignments of connections 12](#_Toc7002930)

[2.3 Results of implementation the algorithm which tries all possible assignments of connections 17](#_Toc7002931)

[Chapter 3 The problem of maximizing when the network is selected 22](#_Toc7002932)

[3.1 The method of maximizing the payoff functions of players with unspecified strategies 22](#_Toc7002933)

[3.2 The applicability of the method of maximizing the payoff functions of players with unspecified strategies 28](#_Toc7002934)

[3.3 The method of maximizing the payoff functions of players with preset strategies 28](#_Toc7002935)

[3.4 Implementation of methods of maximizing the payoff functions of players with unspecified and specified strategies 32](#_Toc7002936)

[3.5 A modification of the methods of maximizing the payoff functions of players with specified and unspecified strategies 34](#_Toc7002937)

[3.6 Implementation of the modification of methods of maximizing the payoff functions of players with specified and unspecified strategies 36](#_Toc7002938)

[Summary 38](#_Toc7002939)

[Conclusion 40](#_Toc7002940)

[List of references 41](#_Toc7002941)

[Application 42](#_Toc7002942)

[Chapter 2 brute force algorithm 42](#_Toc7002943)

[Chapter 3 brute force algorithm 52](#_Toc7002944)

[Chapter 3 method of maximizing the payoff functions of players with unspecified strategies 59](#_Toc7002945)

# Introduction

Games are strictly defined mathematical objects. The game is formed by players, a set of strategies for each player and instructions of payoffs, or payments, players for each combination of strategies. Network games (games with a network structure) are a section of game theory, which studies both methods of forming relationships between players in conflict-controlled systems, and the rules for determining the payoffs of players taking into account these relationships. In this paper, we consider a cooperative version of the network game, in which the main problem is the choice of the rules of distribution of the total payoffs of the players among themselves at some predetermined or formed by the players network structure *(as one of these rules of distribution, we can consider the Shapley value)*. In cooperative games with the ability to transfer funds from one player to another, is used the so-called characteristic function that determines the payoff of each coalition of players. Since in these games the construction of the characteristic function is necessary for different coalitions of players to be able to distribute the total payoffs of players, the importance of the possibility of rapid successful construction of the characteristic function can not be overestimated.

In this paper, we consider a cooperative two-stage network game in which the first stage players together form a network in which each player is associated with a certain set of other players, and in the second stage choose controls so that the total gain of all players was the maximum. Here we consider two options of constructing the characteristic function: in the first embodiment, the characteristic function is based on finding the maximum of the sum of the payoffs of the players only by selection controls of the players, and in the second embodiment, only by the selection of a suitable network of interaction between players. The paper discusses algorithms and approaches for both options that can be used to construct a characteristic function, as well as to reduce the time of it’s construction.

# Problem statement

* Immersion in the theory of network games, the study of their features and properties, including also for a two-stage network game in which players can choose strategies on both stages together.
* Development and implementation of an algorithm by which it is possible to build a characteristic function in a two-stage network game in which players at both stages act together, only by selecting a suitable network .
* Development and implementation of algorithms that enable the development of an alternative more effective approach to the construction of a characteristic function in a two-stage network game compared to the method of searching all possible combinations of player’s controls. At the same time, the construction of the characteristic function is carried out in a two-stage network game in which players at both stages act together, only by choosing the appropriate controls of the players.

# Literature review

To dive into the theory of network games are very useful articles [2], [5], [6] which studies the impact of links between players on decision-making about the choice of strategies. A detailed description of the process of modeling a two-stage network game made it possible to get a view of all stages of the game, their features for the case of non-cooperative two-stage network game, for the case of a two-stage game with the possibility of cooperation in the second stage, as well as a cooperative version of a two-stage network game. Articles [1], [7], [8] also proved to be useful because they supplemented articles [2], [5], [6] with various explanations and examples, enabling a deeper understanding of the area under consideration in this work. Paper [3] also provided an opportunity for a deeper study of the material. The article [4] also discusses the theory of cooperative network games, the influence of player’s links with each other.

# Chapter 1 A two-stage network game

Let a finite set of players be given .

A set – the ultimate set of connections between players. *(The connection between two players means the possibility of interaction between these players)*.

A network means an object . Further, we will identify the set of connections with the network. If the element , it means that there is a connection between player and player in the network .

Consider a two-stage network game in the first stage where players form a network . At the second stage of this game, players choose strategies in the already formed network, which subsequently determine their payoffs. It is worth noting that in this game each player from the set can form connections only with certain players from the set. Herewith the maximum number of connections with other players for each player from the set is limited.

## 1.2 The first stage of the game

Consider the first stage of the game on which the network is formed.

Let is the set of players to which player can offer a link formation.

Value maximum number of connections for player .

In the first stage, all players from the set choose strategies that then form connections between players in the network. The strategy of player at this stage is the vector , which is formed according to the principle (1.1) and satisfies the restrictions (1.2) and (1.3)

(1.1)

(1.2)

(1.3)

Denote the set of strategies of player as . In the network, a connection is established between two players, for example between players and , only if . Thus, after the first stage we get the network .

## 1.3 The second stage of the game

In the second stage, each player from the set chooses a strategy expressed by a pair .

Vectors form a new network and are determined by the rule (1.4)

(1.4)

Vectors are applied to the network , changing it’s structure and turning into a new network , which is obtained from the network except for those links , for which either , or .

some control selected by player from some finite set of controls of player .

Thus, the new network and the selected controls define the payoff function .

Since, due to the properties of the functions, mentioned in the article [2], the removal of any connection does not increase the total payoff of all players in the network, and the problem considered in this paragraph is related to the maximization of the sum of payoffs of all players, then at the second stage the vectors of all players are given containing the maximum number of ones.

## 1.4 Two problems of maximization

Consider the maximization problem (1.5)

(1.5)

where – is the set of neighbors of player .

In this problem, at both stages of the game, players choose strategies together. Acting as one player and choosing strategies , at the first stage a network is formed, and at the second stage controls are selected, at which the problem (1.5) is solved.

In order to distribute the maximum total payoff among all players, an auxiliary cooperative game is built, the characteristic function which is determined for any subset of , called a coalition, according to the rule (1.6)

(1.6)

In this paper are considered solutions of two problems. The first problem is to construct characteristic function by selecting a suitable network with the already given controls for all players from the set i.e. to construct a characteristic function given by the rule (1.7)

(1.7)

The second problem is to construct the characteristic function by selecting the appropriate controls in the already formed network for all players i.e. to construct the characteristic function given by the rule (1.8)

(1.8)

# Chapter 2 The problem of maximization with the selected controls

This chapter discusses the construction of the characteristic function given by the rule (1.7). Consider this problem in more detail for some sets of players in the amount of . Further, the described algorithms can also be used for any coalition of players .

Let

be the number of players

– maximum number of connections with other players for the player

a subset of players with whom player can be connected.

income matrix of player with player. In this statement of problem, the matrix is symmetric because player has the same gain from the connection with player as player from the connection with player .

It is necessary to implement the connections between the players so that the sum of payoffs from the links was the maximum.

Further in the following paragraphs it is considered the developed and implemented by me algorithm for solving this problem by the method of applying all possible sequences of assignments of connections between the players, i.e. by searching all possible options of connections between the players.

## 2.1 Algorithm trying all possible assignments of connections

Describe formally developed algorithm:

**1**. Select all nonzero elements in the matrix and create a set C – a set of pairs (indices from the matrix ) corresponding to nonzero elements in the matrix .

We also create a set the set of sequence numbers of pairs in C. ( where is the number of nonzero elements in the matrix .).

It is important to note that in the sets and the order of all elements is important.

**2**. For all two pairs of indices from the set , for which the conditions and are met, make the corresponding sequence numbers in the set equal. Thus, we obtain in the set the same sequence numbers for the corresponding pairs from the set , reflecting the same connection. (the relationship between and players and the to ).

Algebraically:

For all take a pair from ( is the number of nonzero elements in the matrix ). For each pair take a pair for all from the set .

Check whether the conditions are right: and

If both conditions are met, then check the sequence numbers from corresponding to these two pairs:

* If the numbers are not equal, then replace the sequence number from the set corresponding to the pair from the set C with the sequence number from the set corresponding to the pair from the set C.
* If the numbers are equal, we do not change the set .

**2.2** Create variables , and empty set M, which will subsequently be a set of pairs.

**3**. Take the line , which is the line in the matrix of all possible sequences . ( if the point 3 has never been passed).

* If the line was the last line in the matrix , then stop the algorithm.
* Otherwise: . and set such as were obtained in point 2. Move to point 4.

**4**. From the line we take the element having the sequence number . ( if line is a new line from which elements have not yet been selected). If the value of is greater than the number of elements in the line (we can not take a new element from the line ), or this element is not in the set then return to step 3 (herewith we take the next line ). Herewith, compare with . If more , replace on . value zeroize: .

* If the element is contained in the set , then go to step 5. Herewith, increase the value of the variable by .

**5**. Find two elements from the set equal to the value and find the pairs corresponding to these two elements from the set and .

At this stage we have chosen some connection between two players and remove from the set of remaining possible for selecting links pairs and . Since these pairs correspond to one selected connection between two players. From the set we remove ordinal numbers corresponding to pairs and .

Since we have chosen some connection between two players, the gain from this connection must be added to the variable , reflecting the total gain from the choice of this purpose of connections between players. To do this, find in the matrix value corresponding to the pair and add this value to the variable := .

Since we have chosen some connection between two players and , the maximum number of connections for both players must be reduced by one:

move to point 6

**6**. Check the condition:. If the condition is met, then, therefore, the player corresponding to the index can no longer have a connection with other players because the maximum number of connections of this player with other players becomes zero (the player can no longer be associated with anyone and no one can be associated with the player). Therefore, we remove all pairs from the set , the first or second index of which is equal to the number of the player . From we delete sequence numbers corresponding to deleted pairs.

Similarly, check the condition . If the condition is met, then from the set we remove all pairs, the first or second indexes of which are equal to the number of the player .

move to point 4.

## 2.2 The idea of the algorithm trying all possible assignments of connections

The idea of the algorithm is to iterate through all possible combinations of assignments of connections between players. One combination of all possible combinations will be a combination of assignments, in which the total gain from the links will be the maximum. To do this, we create a matrix matrix of all possible combinations of connections between the players. Each row of the matrix is a unique sequence of assignments of connections between players in the matrix .

Since the set contains the numbers of all possible connections between players, and the same connections have the same numbers (player with player and player with player ), then creating a new set , containing all the numbers from the set , which are not repeated, we get a set of numbers of all possible unique connections between players. Thus, the matrix is the matrix of all possible combinations of elements from the set . *(it is important to note that the value of is halved: , is the number of elements in the set )*

All possible combinations can be represented as a tree in Fig. 2.1

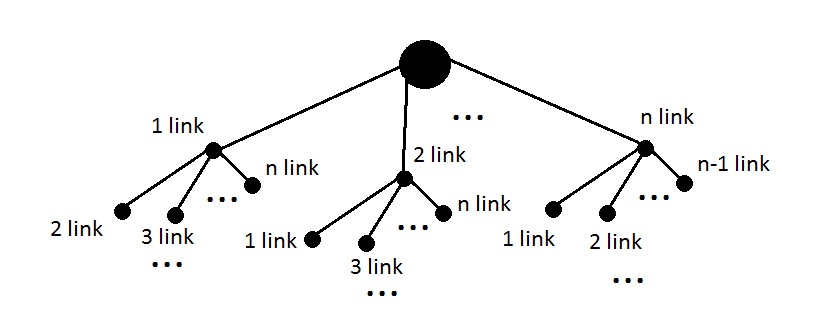


Fig. 2.1

This tree shows all possible combinations of connection assignments between players. In figure. 2.1 it is seen that in the first step we can choose one of different links, where is the number of elements in the set . In the second step we can choose one of the links and so on… One of these combinations and contains the assignment of connections between players, in which the total gain from the connections between players maximum.

However, due to the fact that each player has a limit on the maximum number of links, some links can be discarded before we choose them. That is, some paths from the tree in Fig. 2.1 may not exist (only the first links can be selected). In this case, choosing the first links, we no longer use the current combination of links and choose the next combination, which was reflected in the above algorithm. *(In this case, for the obtained incomplete path, in any case, we compare the total gain from the links contained in this incomplete path with the maximum gain.)*

We will create a matrix of all possible combinations of values from the set matrix using the following algorithm:

1. Create sets as follows:

Set is given equal to set . Other sets are given by the following rule:

For all the set is obtained from the set by adding an entire group of subsets for each subset of the set . Each subset of a group of subsets is obtained from the original set (subset in ) by removing one of the elements of the original set. (subset in ). In total, there are subsets in the subgroup, each of which does not coincide with another subset from this group.

Example:

For the set we get two groups of subsets:

* For a subset from we obtain a group:

,,

* For a subset we obtain a group:

, , ,

It is worth noting that the order of groups of subsets, subsets in groups and elements in subsets is important in all sets . This item is performed strictly in accordance with the order of all elements in the sets .

move to point 2.

1. Set the variable and set the variable . Next for all :
2. Set the value of the variable equal to the number of elements in the set .

Create a set , in which new elements will be added that make up the next possible combination.

Set the value of the variable .

1. The value of the variable make equal to the element with the sequence number from the set . Set the value of the variable .
2. Choose from the set a subset having the sequence number . (Herewith in the set all subsets have the sequence numbers which take into account other subsets from other groups of subsets . For example, in the set ,, ,, , , }, received from the subset with sequence number 0 will be , and with sequence number 3 will be )

Set the element equal to the difference between the sets and .

Add the element to the set . .

* If , then add the set to the matrix *(a new line in the matrix )*. Set define as an empty set.

. If the value of is equal , then stop the execution of the algorithm. Otherwise, go to step 4.

* Otherwise, the value of the variable is reduced by 1: .

Repeat point 5.

Example

Let given number of players

Set the maximum number of connections with other players for each player:

For each player, we define a subset of players with which the player can be associated , , . That is, each player can be associated with any other player.

The income matrix is defined as follows:

1. Create a set of pairs

Create a set of sequence numbers .

1. Get the same sequence numbers in the set for the corresponding pairs from the set , reflecting the same connection:

**2.2** Create variables , and empty set , which later will be a set of pairs.

1. From the matrix of all possible values

Take the line

1. Take the element
2. Delete two elements from . Obtain .

Remove the corresponding pairs from .

Obtain .

We add to the value of the variable the value of the payoff from the connection between two players

Since we have chosen the connection between players and , we reduce the maximum number of connections for both players by :

1. Check condition and condition

Both conditions are not met, so go to item 4.

**4**. Take an element

1. Remove two elements from . Obtain 5, 5}.

Remove the corresponding pairs from . Obtain (2,3), (3,2)}

We add to the value of the variable the value of the gain from the connection between the two players. Thus the value of .

Since we have chosen the connection between players and , we reduce the maximum number of connections for both players by :

The maximum values of possible connections for players are equal:

1. Check condition and condition .

Condition is true, but in set no elements equal to .

Go to item 4.

**4**. Take the element

1. Delete two elements from . Obtain - empty set.

Remove the corresponding pairs from . Obtain } – empty set.

We add to the value of the variable the value of the gain from the connection between the two players. Thus the value of .

Since we have chosen the connection between players and , we reduce the maximum number of connections for both players by :

The maximum values of possible connections for players are equal:

1. Check condition and condition .

Both conditions are satisfied, but the set is empty.

Move to point 4.

**4**. Since the element in the line was the last one, we take a new line from the matrix : .

Performing the same actions for all other lines as for the line we get the value of the variable . This value will be the maximum payoff value that can be obtained when distributing links between players.

## 2.3 Results of implementation the algorithm which tries all possible assignments of connections

I have implemented the algorithm described in paragraphs 2.1 and 2.2 in the Python programming language. Consider the specific examples of the work of this program.

Example 1

Let the number of players

Maximum number of connections with other players for each player

, ,

For each player, we will define a subset of players that the player can be associated with

, ,

That is, each player can be associated with any other player.

Let's define a symmetric matrix of values of connections between players

As a result of the program, the following assignment of links between the players was obtained

– the connection of the and players

– the connection of the and players

– the connection of the and players

The value of the payoff received from this distribution of links between players

Since the maximum number of connections for all players is set in such a way that each player can make a connection with each other, all connections were involved in this situation.

Next, change the maximum number of links for the player .

For a new example, we get the following assignment of connections between players

– the connection of the and players

– the connection of the and players

The value of the payoff received from this distribution of links between players

Since the player can only have a connection with one other player, the player has a connection with the player, because this connection brings a greater gain compared to the connection between the and the player. Another connection between the and players remained as the maximum number of connections for other players did not change.

Now set the maximum number of connections for all players to one: , ,

For a new example, we get the following assignment of connections between players

– the connection of the and players

The value of the payoff received from this distribution of links between players

Since in this case each player can make only one connection, then from all possible connections players can make only one connection, the cost of which will be the total payoff of the players. The maximum payoff has a connection between the and players which was selected.

Example 2

Let the number of players

Maximum number of connections with other players for each player , , , ,

For each player, we will define a subset of players that the player can be associated with

, , , ,

That is, each player can be associated with any other player.

Let's define a symmetric matrix of values of connections between players

As a result of the program, the following assignment of links between the players was obtained

– the connection of the and players

– the connection of the and players

– the connection of the and players

– the connection of the and players

The value of the payoff received from this distribution of links between players

From the results of the program, it can be noted that the fourth player used absolutely all possible connections for him. At the same time, all the other players, having the opportunity to establish one connection, used these connections, and established those connections that bring the maximum benefit. This distribution is optimal and brings the maximum total gain.

Next, change the above example by setting the payoff from establishing a connection between and players to . This value is significantly larger than the total payoff therefore, the link (1,2) must be in any case involved. Let's rewrite the cost matrix

As a result of the program, the following assignment of links between the players was obtained

– the connection of the and players

– the connection of the and players

– the connection of the and players

The value of the payoff received from this distribution of links between players

The result of the program shows that in comparison with the previous example, the distribution of connections remained the same, but the and players instead of making a connection with the fourth player made a connection with each other. This result is logical because this connection has a much greater contribution to the total payoff than the sum of the payoffs from the connections between the and players and the and players.

Next, we reduce the maximum number of connections with other players for the fourth player to one: and change the matrix of values of connections

Now the gain from the connection of the and players is and not .

Between the players was received the following assignment of links

– the connection of the and players

– the connection of the and players

The value of the payoff received from this distribution of links between players

Since the maximum number of connections for the fourth player was reduced to one, and the gain from the connection between the and players is less than the gain between the and as well as the gain from the connection between the and players, then, as a consequence, the connections between the and players and the and players were left.

Thus, running the program for different examples, containing players and for different examples, containing players, we get solutions, the correctness of which can be easily verified by ourselves. However, since the complexity of this algorithm is , where is the number of all possible connections between all players, then running the program for any example containing more than connections turns out to be a very time-consuming task for the computer, which can not be solved without the resources of a supercomputer. To sum up, for examples containing more than five players, this program can be run only if the number of possible connections between players is not more than i.e. with a small number of potential connections between players.

# Chapter 3 The problem of maximizing when the network is selected

This Chapter discusses the construction of the characteristic function given by the rule (1.8). Consider this problem in more detail.

Let the network be already given. For each player in network there is a set of controls: , as well as the payoff functions . The payoff function for each player depends only on the selected control by player in the network and on the set of controls selected by the neighbors of player : , where is the set of neighbors of player in network .

Thus, for each player, the function is given by the values, the number of which is equal to the product of the number of controls in the set of player : by the numbers of controls of all neighbors of player . For example, if player in the network is connected to players and , then the number of values of the function is .

It is necessary to find such set of controls , where is the number of players in the network , that the sum of the payoff functions of all players is maximum

(3.1)

Of course, the above problem can be solved by going through all possible sets of control values of all players. However, for large networks, trying all possible combinations of controls can be very time-consuming process. Therefore, to solve this problem requires a more intelligent approach.

Further in the following paragraphs we will consider developed and implemented by me algorithms of solving this problem by maximizing the payoff functions of players with undefined strategies, by maximizing the payoff functions of the players with specified strategies, by method of modification of the algorithms of maximization the payoff functions.

## 3.1 The method of maximizing the payoff functions of players with unspecified strategies

The idea of the presented algorithm is to find first some so-called reference plan of the maximum values, which will help to significantly reduce the set of various possible sets of controls (denote as a set of ), and then, using the resulting reference plan, try all possible combinations of controls from the set , one of which will be a set, in which the sum of the values of the payoff functions of players will be maximum i.e. solving the problem .

As mentioned above, first we find a reference plan. To do this, from the set of all players in the network in some way select one player who is denoted as . From the set of all possible values of the function , select the maximum value: and take the set of controls corresponding to this maximum value: . Set , where – is the number of neighbors of player , contains the controls selected by player and neighbors of player from the sets of controls . For each neighbor of player , there is also a set of neighbors with unspecified strategies, the functions of which: are also maximized at the controls only of such players, for which controls are not defined, then get a new set of controls. Further, the payoff functions for the neighbours of the neighbours of the players with unspecified strategies are also maximized also at the controls of only those players for which controls are not defined. The process continues until all players are assigned some control. Then the resulting set of controls will be the reference plan .

Consider the example of the choice of the reference plan. In let's represent a network , where for player and for neighbors of player controls are already selected, corresponding to the maximum value of the function : .

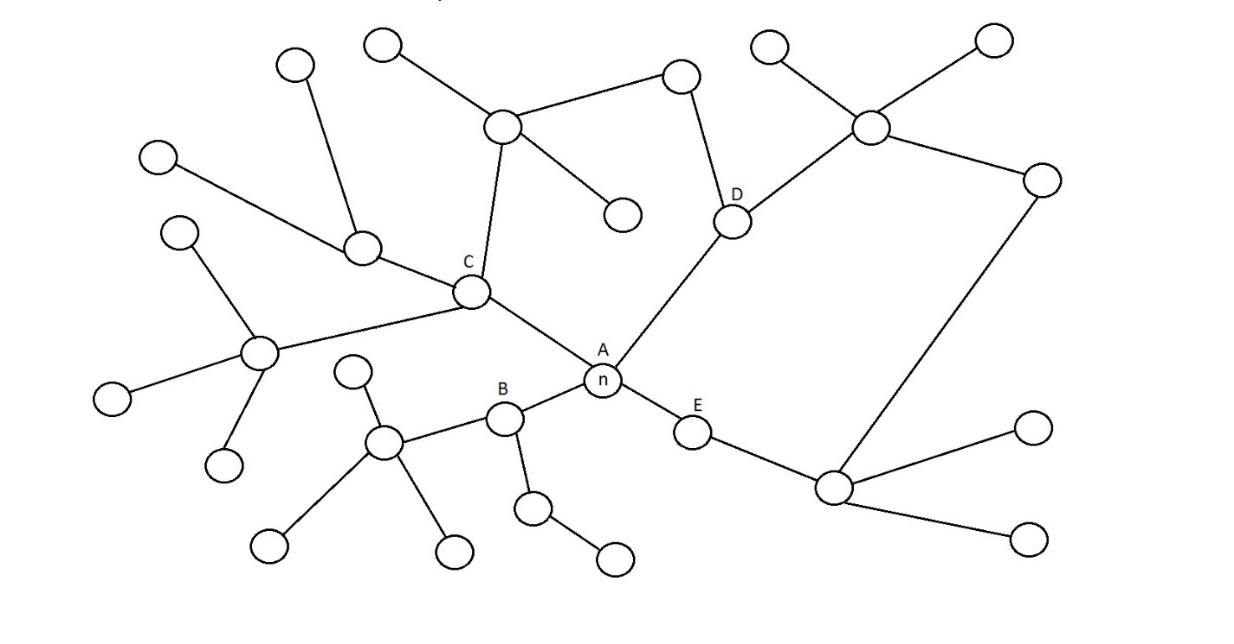


Fig.1

Next, select all players in the network , for which controls are not defined, but which have neighbors with selected controls and designate these players as .

Select sets of controls corresponding to the maximum values of the functions : where andcontrols of the neighbors of the player for which controls are not defined. Thus, for all players , and also for all neighbors of players with undefined controls, we also set controls. Herewith the order of consideration of the functions i.e. the order of strategy selection for players and their neighbors occurs in any order. We should note some moments:

* If during the process of defining the strategies for players and their neighbors, set of neighbors denote as , some element turned out to be a neighbour to the element and also neighbor to the element , where players from set , herewith the function was considered to maximize before the function , the function is maximized across a set of controls , which does not include the control for player .
* Also, if in set one player is a neighbor for another, say a player is a neighbor for player , herewith function is considered to maximize before the function , the function is maximized by the set of controls , which does not include the control for a player .

On this display is presented. In this case

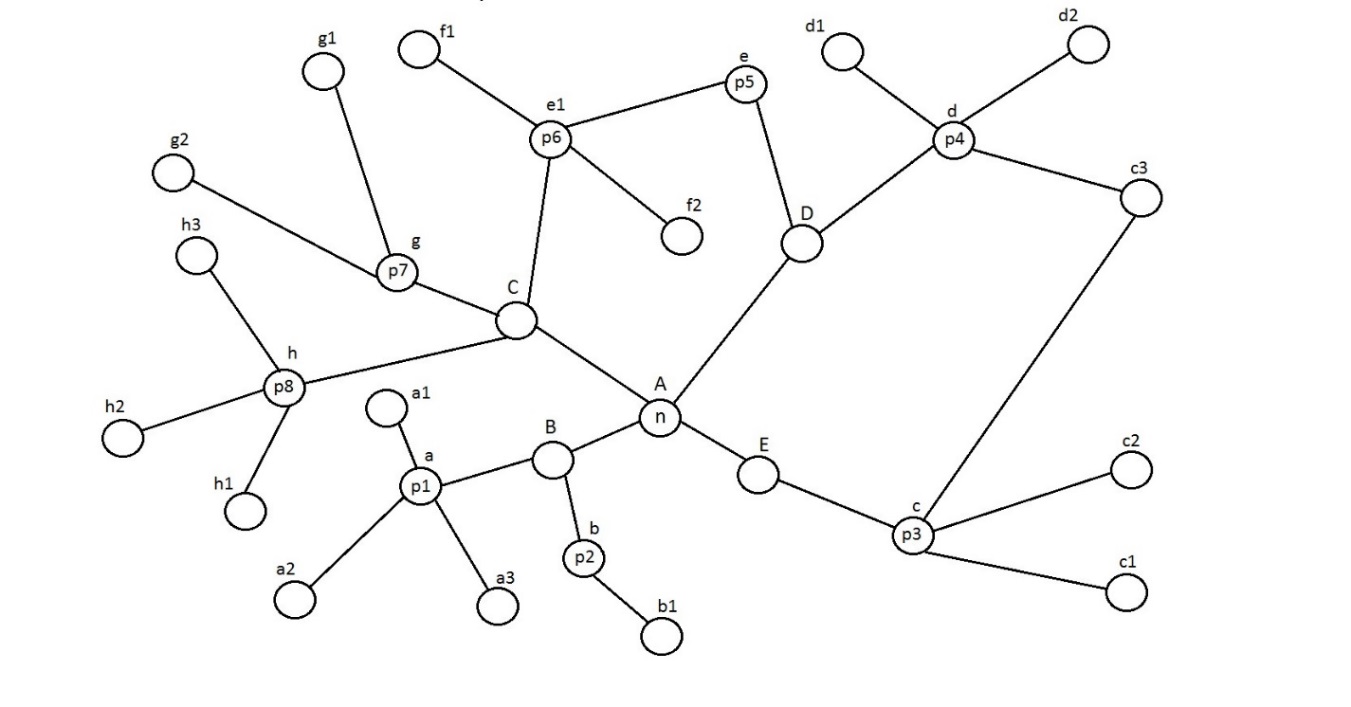


Fig.2

* *Here, the corresponding strategies are selected for player and his neighbors, maximizing the function .*
* *Strategies are selected for player and his neighbor. For another neighbor of player , strategy B is already selected, so the maximum possible value of the function is selected only by the strategies of player and his one neighbor.*
* *Players and have a common neighbor. In this case, the function is maximized earlier than the function , so the maximum of the function is found only by the controls of the player and his two neighbors, for which, after maximization, the strategies and are set.*
* *For the player , the player is a neighbor, so the function is maximized by the set of controls that do not contain the control i.e. is maximized taking into account the fact that the strategy for the player is already known.*

If there were still some players in the network with unselected controls, then for all players in the network for which controls are not selected, but who have neighbors with the selected controls, we designate these players as , we do the same actions as for players . If after this there are also players in the network with unselected controls, then we find such players and do the same actions as for the players and so on until the players with unselected controls remain in the network.

After finding the reference plan , expressed by a set of controls for all players in the network , to each player except player correspond two sets and , which are disjoint subsets of the set of all possible values of the payoff function of the current player and when combined equal to the set and . The first subset contains all values of the payoff function of player which are greater than the value achieved by the function by substituting the controls from the reference plan , and the second subset contains all values of the payoff function of the player which are smaller than the value achieved by the function by substituting the controls from the reference plan . This situation is presented graphically on

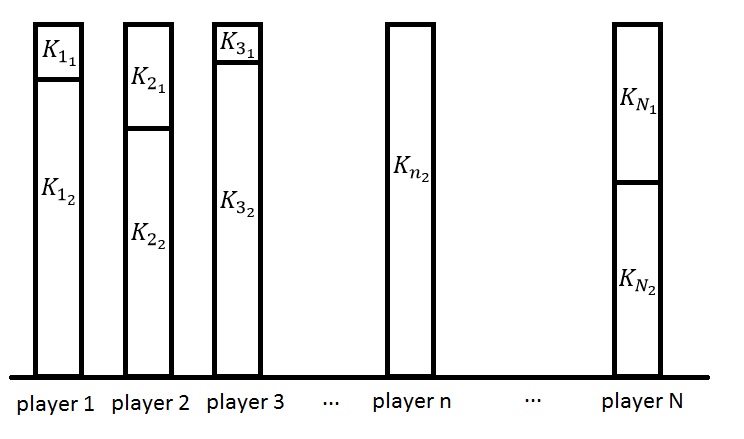
**

Fig.3

For player there is only one set because the reference plan was given taking into account the fact that the maximum of the payoff function is found on the controls of player and the controls of absolutely all his neighbors.

The set of controls that solves the problem may be the reference plan , but there may be another set of controls that solves the problem , by which the value is greater than the value achieved by the set of controls . If such set exists, then for such set in the set of functions the value of at least one of the functions, which we denote as , corresponding to the control values from the set will be greater than the value of the same function , corresponding to the controls from the set .

Let – be the number of the player for which the value of the corresponding function at the values of the set of controls from is greater than at the set of controls from . Denote the set of control values at which the function reaches the value by the set , which is a subset of the set . Denote because , hence .

Thus, the values of some controls from the set need to be equal in the aggregate to a single set of controls, which corresponds to one value of the function from sets of values . More detail, each set contains some set of function values ((( ( (, and each value of the function from all the above values corresponds to a set of control values: ( (( ((. Some of the controls in the control set are collectively equal to one of the above control sets. Therefore, to find the set for each set , where , and , it is sufficient to iterate through all possible sets in which the corresponding controls are equal to the controls from the set .

Let the set contains all sets of controls which correspond to all the values of the functions from each set .

It is necessary to do the following for each set :

Let set is the set of players for which in the set are defined the values of controls, a set a set of the rest other players, then for each set of controls is created a set containing all possible sets of controls : , where , and , in which the values of the controls for players from the set equal to the values of controls from the set (for the rest of the players all possible combinations of controls). In the set are added all sets , where , and , and in set are added all values for all corresponding to all sets of controls from the set .

After all sets have been processed, in the set we find the maximum value , and in the set we find the set corresponding to this value . If the value is greater than the value i.e. value achieved at a set of controls , then the set will correspond to the set of controls which will solve the problem , otherwise to solve the problem will a set of controls .

## 3.2 The applicability of the method of maximizing the payoff functions of players with unspecified strategies

This algorithm is suitable for a network in which each player has quite a lot of neighbors in the network because the greater the number of neighbors for each player the smaller the number of players which are not neighbors for the current player, and, consequently, fewer options for selection from possible strategies. For example, let there are players in some network, each with possible controls. Let the player has neighbors and players, then for each set of controls corresponding to the values from the set it is necessary to iterate over options, where the number of options is obtained based on the fact that the , and players have different controls each. However, if the network is too dense, then the number of variants in the sets (where , are players in the network with too many neighbors) can have a significant impact on the total number of iterations in the algorithm, which may also make the algorithm inefficient.

If there is only one player in the network that can have a significant impact on the number of search options, or some player has a much greater impact on the number of search options than the others, for example, a player who has too few neighbors, then this player should be taken as the player.

Thus, this algorithm is suitable for a network in which:

* Absolutely each player has a lot of neighbors (dense network), but the number of players who are not neighbors for each player is a comparable number.
* Players have a large number of strategies.

## 3.3 The method of maximizing the payoff functions of players with preset strategies

As mentioned above, the choice of the reference plan affects the number of iterations in the algorithm presented in paragraph . A better choice of plan reduces the number of variants of sets of controls in the set , used in the search, thereby reducing the search area for a set of controls , solving the problem .

In the algorithm presented in paragraph , one player was selected, which function was maximized by the controls of player and by controls of absolutely all neighbors of player , and then for all players with unselected controls having neighbors with selected controls, their corresponding functions were maximized. The process continued until all players were given a certain control.

This paragraph discusses the algorithm for solving the problem , which is different from the algorithm presented in paragraph in the method of selecting a reference plan. The difference lies in the fact that when choosing a reference plan for each step are maximized not functions of players with unspecified strategies that have neighbors with specified strategies: , but functions of players with specified strategies that have neighbors with unspecified strategies.

For most networks, at such approach of constructing a reference plan will be found a maximum of more payoff functions players, which can significantly reduce the number of elements in some of the sets . In this case, similar to the algorithm presented in paragraph , if some element for which the strategy is not set, at step is a neighbor to more than one element of the set , for example to players and , then only one payoff function of the functions is maximized by the set of controls of the player .

Let's take a closer look at the above approach with an example. In , we present a network ,where for the player and his neighbors controls are already selected,corresponding to the maximum value of the function .

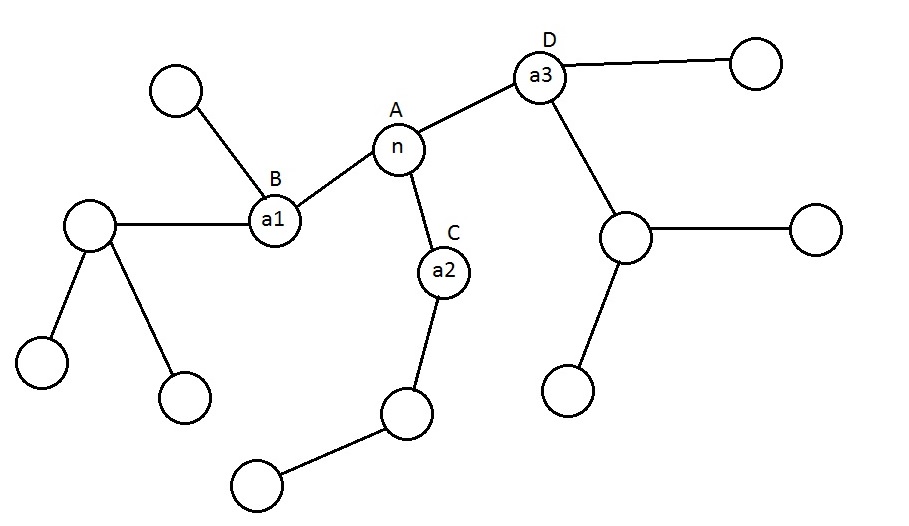


Fig.4

Next, maximize functions at the controls of neighbors of players , for which controls are not defined. This representation is shown on

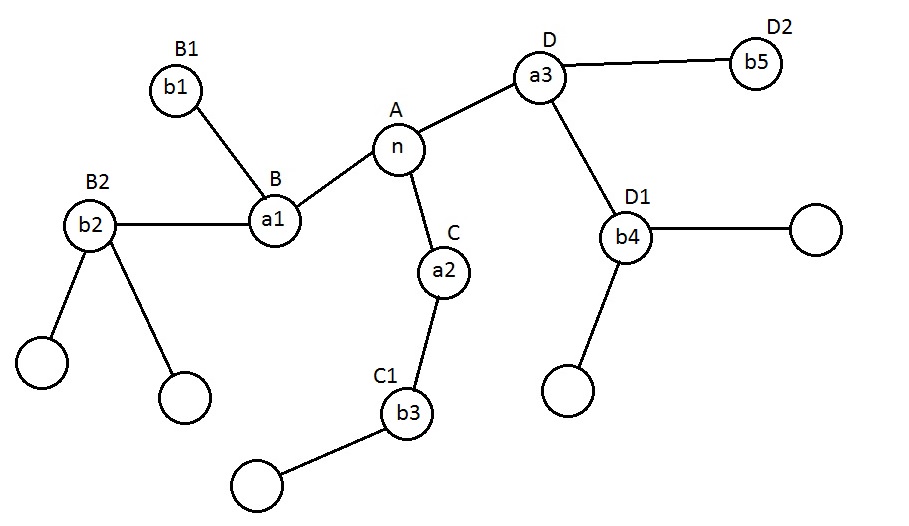
**

Fig.5

*For player – function is maximized at controls of two his neighbors , for player function is maximized at control of one of his neighbor , for player function is maximized at the controls of two his neighbors .*

At the last stage, maximizing the functions , we obtain in setting for each player in the network a certain control. That is, we obtain a reference plan .

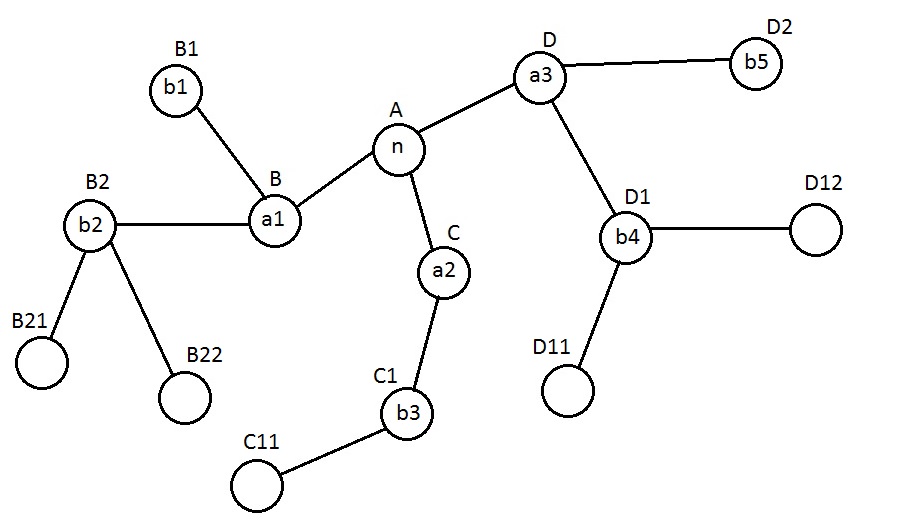


Fig.6

This approach has a drawback in the sense that when maximizing the functions of the players, the maximum is found at the set of controls, in which there is obviously no control of player , the function of which is maximized because the control for player is already set. Thus, in this approach, in most cases, for maximized functions, the maximum will be found at fewer number of controls than for functions in the algorithm presented in .

Another disadvantage of this approach of building a reference plan is that for some networks, fewer functions can be maximized than for the algorithm presented in . A striking example is the network shown on

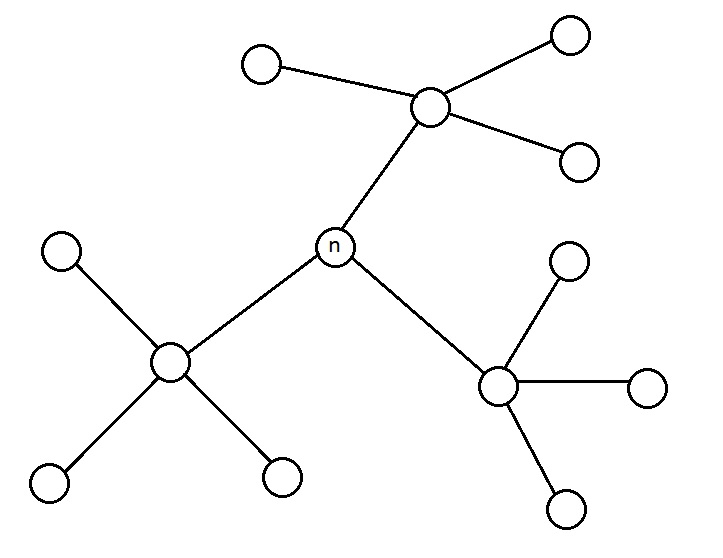


Fig.7

In this network, the use of the algorithm presented in is more likely to be more efficient than the use of this algorithm.

## 3.4 Implementation of methods of maximizing the payoff functions of players with unspecified and specified strategies

Take a network containing players, in which each player can choose one of possible strategies: . A total of various possible combinations of sets selected by players controls: .

Set the links between the players in the form of a table on

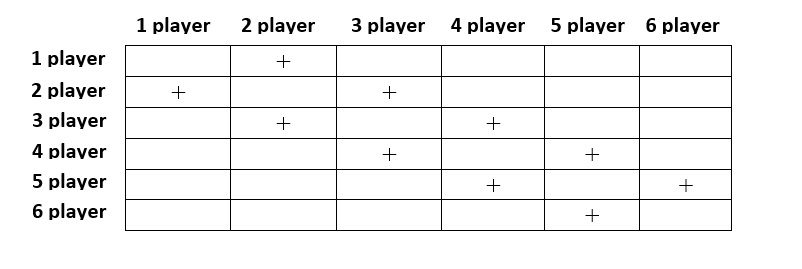


Fig.8

The payoff functions for each player will be set randomly. The functions of the and players depend on the control choice of the given player and his one neighbor, therefore are expressed as different values. The payoff functions of the , and players depend on the choice of the player's control and the choice of the control of his two neighbors, so are expressed as different values. Graphically, this network is shown in

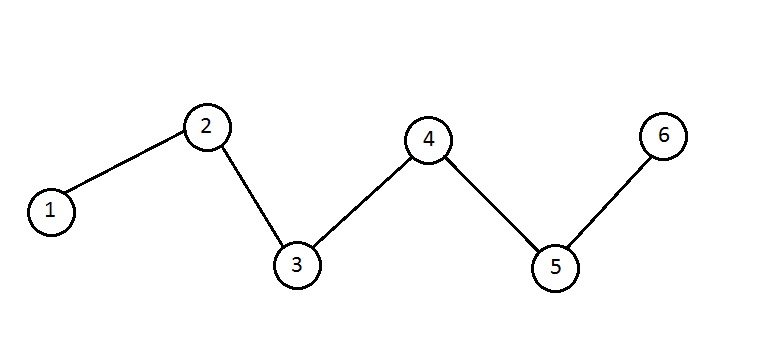


Fig.9

I have implemented in the Python programming language for the above network the algorithm presented in and the algorithm presented in i.e. methods for maximizing the payoff functions of players with unspecified and preset strategies. According to results of the programs was obtained the following set of controls that solves the problem

(3.4.1)

To check the correctness of the algorithms, a program was written that solves the problem by searching absolutely all possible variants of control sets. The result was similar to the result .

Since in this network there are only six players and a set of controls of each player contains different controls: , then the number of iterations for the program that solves the problem by searching absolutely all possible options of sets of controls is .

The number of iterations of the program solving the problem using the algorithm presented in , i.e. using the **method of maximizing the payoff functions of players with given strategies** was obtained : .

The number of iterations of the program that solves the problem using the algorithm presented in , i.e. using the **method of maximizing the payoff functions of players with unspecified strategies** was obtained: .

In this case, the algorithm presented in section , is more effective than the algorithm presented in section . This is due to the fact that despite the fact that for each function the maximum was found only by a set of controls of one neighbor of player , the maximum was found for each function in contrast to the algorithm .

Thus, the approach, in which the maximum of functions is found by the set of controls, which does not contain the control of the player, in this case is more effective due to the fact that the maximum was found for a greater number of payoff functions i.e. to maximize a greater number of functions is more effective than to maximize a smaller number of functions by greater number of controls.

## 3.5 A modification of the methods of maximizing the payoff functions of players with specified and unspecified strategies

The above two approaches of finding the reference plan are based on maximizing the payoff functions of players, taking into account the fact that for some players strategies are already defined. Herewith only for player , the payoff function is maximized by the controls of absolutely all players on which it depends. However, it can be much more effective approach, in which at the initial stage not one player with the number is selected, function of which we maximize, but several players with numbers , which functions are maximized. Herewith these players are selected in such way that regardless of the order of consideration of the functions for any function the maximum is found by controls of absolutely all players on which it depends. Next, the reference plan is found according to one of the above methods.

Consider the example of the selection of the reference plan by modifying the algorithm presented in . we present the network , in which for player and neighbors of player , for player and neighbors of player , for player and neighbors of player are already defined strategies, i.e., maximized payoff functions by strategies of the players , respectively, and by strategies of all the neighbors of players .

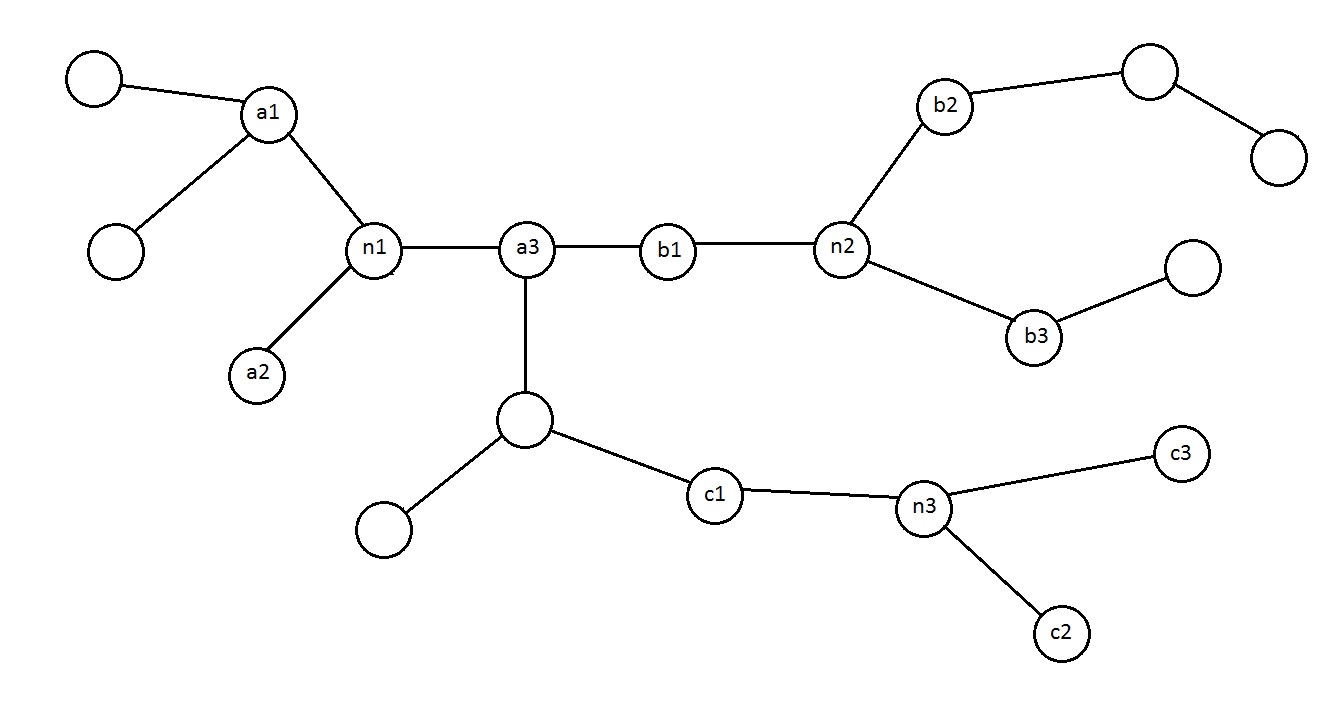


Fig.10

Then, as well as in the algorithm in the network we find all the players for which the strategies are defined, but which have neighbors with unspecified strategies. In this case, these are the players . Maximizing functions (payoff function is not considered for maximization since the maximum of the function is found earlier) by the controls of players with unspecified strategies: , define for this players strategies on

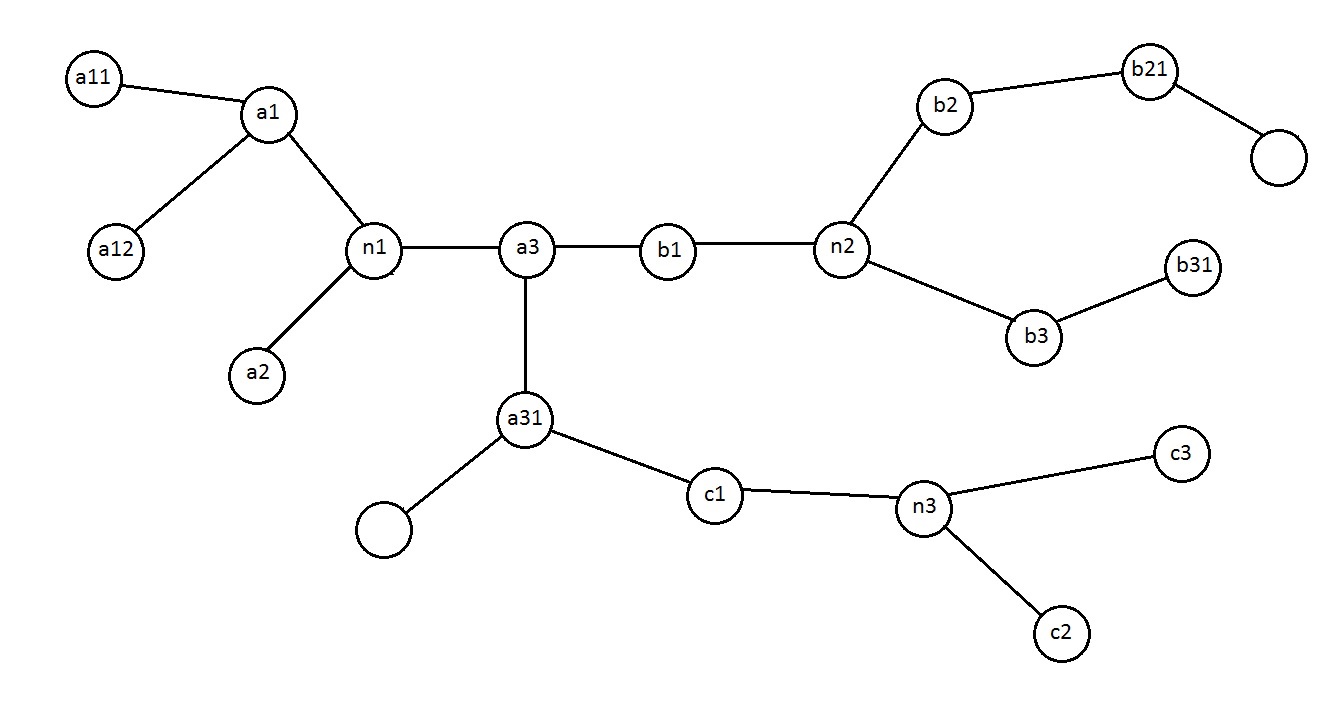


Fig.11

At the last stage, we again find in the network players with defined strategies having neighbors with unspecified strategies, in this case, players and set them strategies respectively, finally getting a reference plan , as shown in

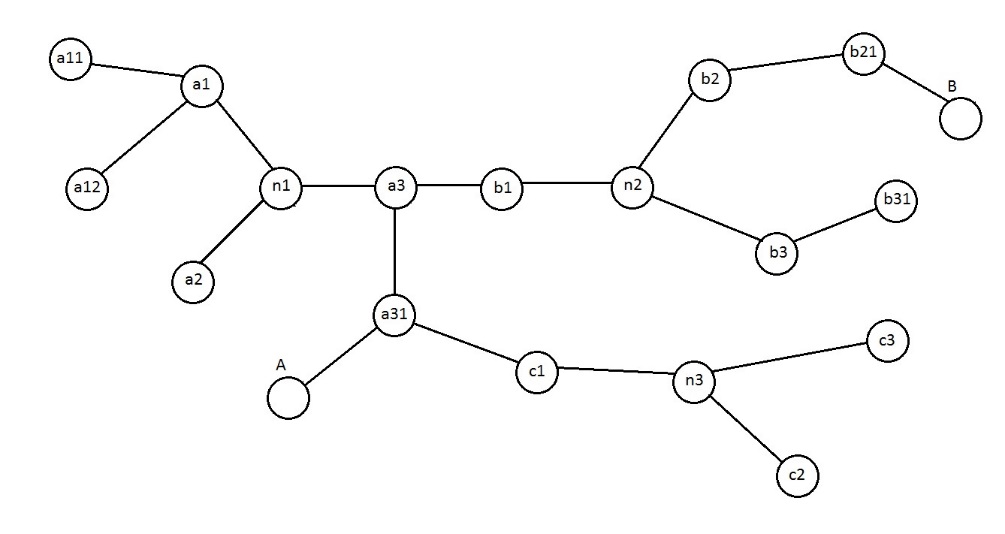


Fig.12

## 3.6 Implementation of the modification of methods of maximizing the payoff functions of players with specified and unspecified strategies

I have implemented a modification of the method of maximizing of the payoff functions of players with defined strategies in the Python programming language. At the initial stage, for the example presented in , shown in , players and were selected. The maximum of the function has been found by the controls of players and , maximum of function has been found by the controls of players and , as shown in

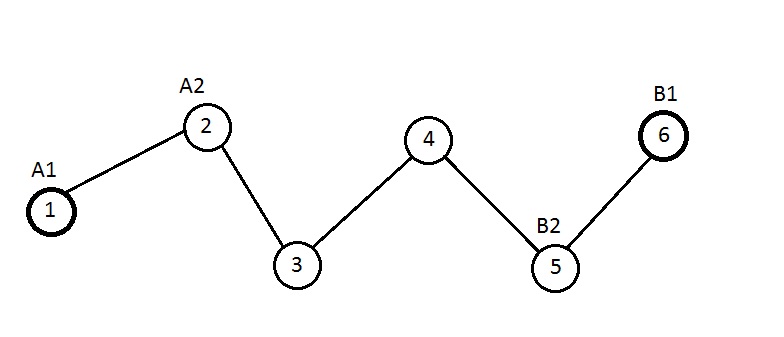


Fig.13

Ultimately, by maximizing the functions and , we define the controls for the remaining and players.

The number of iterations of the program solving the problem by modification of the algorithm presented in ,i.e. by **modification of the method of maximizing the payoff functions of players with defined strategies** is equal to: .

This result indicates that for this particular example, the application of the algorithm modification is less effective than the application of the algorithm . Most likely this is due to the fact that in this modification the maximum is found for fewer number of functions than in the original algorithm. This makes possible to conclude that the increase in the number of maximized functions when finding the reference plan, perhaps in most cases, has a positive effect, reducing the number of iterations in the program.

# Summary

Summing up the work done, it is worth noting that all defined tasks were completed. The algorithm was developed, which gives the possibility to find a characteristic function in two-stage network game in which players in the two phases act together, maximizing the total gain, according to the rule i.e., by construction of the characteristic function only by selection of a suitable network . The correct operation of the so-called brute-force algorithm was tested by substituting various examples for three players into my implementation of this algorithm, as well as various examples for five players. The results of the program coincided with the expected results, easily confirmed logically. However, due to the complexity of the algorithm equal to , running the implementation of this algorithm for the case when the network has more than five players, turn out impossible due to lack of power of a simple computer.

An algorithm has been developed and implemented that makes it possible to find a characteristic function in a two-stage network game according to the rule . that is, by constructing a characteristic function only by selecting the appropriate controls for each player, and another version of this algorithm has been developed and implemented, in which the reference plan was selected by maximizing the payoff functions not of players with unspecified strategies, but payoff functions of players with specified strategies. The correctness of the algorithms has been tested by implementing another algorithm that solves the same problem by iterating through all possible combinations of controls of players. An algorithm was developed and implemented, which is a modification of the algorithms of methods for maximizing the payoff functions of players with defined and unspecified strategies, in which at the initial stage was chosen not one player, but several. On a concrete example has been verified the possible applicability and efficiency of all implemented algorithms, which find the characteristic function according to the rule .

From all of the above, we can conclude that the algorithm developed and implemented by me, which makes it possible to build a characteristic function according to the rule , works and can be successfully used, however, for a large number of players to run the implementation of this program requires a supercomputer. Therefore, to improve the efficiency of this algorithm it’s modernization is required in which there is a more intelligent approach of solution. The work of the developed and implemented series of algorithms that make it possible to find a characteristic function according to the rule is much more effective in comparison with the algorithm that solves the same problem by a method of a simple search of all possible combinations, which makes it possible to draw a conclusion about the possibility of further development of the described approach of finding the characteristic function reflected in a series of algorithms by means of modernization of finding the reference plan.

# Conclusion

According to the results of the work, we can conclude that for the variant of constructing the characteristic function, in which the characteristic function is built on the basis of finding the maximum of the sum of payoff of players only by controls of the players, a whole approach of finding the function was developed, in which it is built much faster than when using the usual method of search of all possible combinations. Based on the results of the complex of program implementations of the algorithms, we can conclude that the described and implemented approach has the potential for development and the direction of development is shown in the results of the work. For the option of constructing a characteristic function, in which it is constructed by finding the optimal network, an algorithm was developed, which also has the potential for development.

# List of references

1. **Petrosyan L., Sedakov A.**One-way flow two-stage network games. Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика, информатика, процессы управления, 2014, № 4, c. 72–81
2. **Петросян Л.А., Седаков А.А., Бочкарев А.О.**Двухступенчатые сетевые игры // Математическая теория игр и ее приложения, 2013. – T. 5. – № 4. – с. 84–104
3. **Petrosyan L., Sedakov A.** The Subgame-Consistent Shapley Value for Dynamic Network Games with Shock // Dynamic Games and Applications, 2015.
4. **Petrosyan, L., Sedakov, A.**The Subgame-Consistent Shapley Value for Dynamic Network Games with Shock // Dynamic Games and Applications, 2016. — Vol. 6, — №4. — p. 520-537
5. **Петросян Л.А., Зенкевич Н.А., Шевкопляс Е.В. *Теория игр.* СПб: БХВ-Петербург, 2012.**
6. **Goyal S., Vega-Redondo F. *Network formation and social coordination* // Games and Economic Behavior. 2005. V. 50. P. 178-207.**
7. **Jackson M., Watts A. On the formation of interaction networks in social coordination games // Games and Economic Behavior. 2002. Vol. 41, N 2. P. 265-291.**
8. **Bala V., Goyal S. A., Noncooperative Model of Network Formation. Econometrica, 2000, vol. 68, no. 5. P. 1181-1229.**

# Application

In this application, the program code related to the defining payoff functions of players have been omitted because of it’s large amount.

Full program code is in the repository created by me, located at the link:

https://github.com/Valera14/Methods-of-constructing-a-characteristic-function-in-a-network-game

## Chapter 2 brute force algorithm

import numpy as np

N = 5

L = list()

N\_links = np.zeros([N, N])

c\_cost = np.zeros([N, N])

N\_links[0][0] = 0

N\_links[1][0] = 1

N\_links[2][0] = 1

#N\_links[3][0] = 1

N\_links[0][1] = 1

N\_links[1][1] = 0

N\_links[2][1] = 1

#N\_links[3][1] = 1

N\_links[0][2] = 1

N\_links[1][2] = 1

N\_links[2][2] = 0

#N\_links[3][2] = 0

#N\_links[0][3] = 1

#N\_links[1][3] = 1

#N\_links[2][3] = 0

#N\_links[3][3] = 0

c\_cost[0][0] = 0

c\_cost[1][0] = 1

c\_cost[2][0] = 3

c\_cost[3][0] = 4

c\_cost[4][0] = 3

#c\_cost[5][0] = 8

#c\_cost[3][0] = 5

c\_cost[0][1] = 1

c\_cost[1][1] = 0

c\_cost[2][1] = 5

c\_cost[3][1] = 7

c\_cost[4][1] = 5

#c\_cost[5][1] = 32

#c\_cost[3][1] = 19

c\_cost[0][2] = 3

c\_cost[1][2] = 5

c\_cost[2][2] = 0

c\_cost[3][2] = 8

c\_cost[4][2] = 7

#c\_cost[5][2] = 3

c\_cost[0][3] = 4

c\_cost[1][3] = 7

c\_cost[2][3] = 8

c\_cost[3][3] = 0

c\_cost[4][3] = 8

#c\_cost[5][3] = 8

c\_cost[0][4] = 3

c\_cost[1][4] = 5

c\_cost[2][4] = 7

c\_cost[3][4] = 8

c\_cost[4][4] = 0

#c\_cost[5][4] = 7

#c\_cost[3][2] = 0

#c\_cost[0][3] = 5

#c\_cost[1][3] = 19

#c\_cost[2][3] = 0

#c\_cost[3][3] = 0

L.append(1)

L.append(1)

L.append(1)

L.append(4)

L.append(1)

#L.append(3)

#L.append(2)

C = list()

C\_n = list()

for i in range(0, N):

for k in range(0, N):

if(c\_cost[k][i] != 0):

C.append([k, i])

n = len(C)

for u in range(0,n):

C\_n.append(u)

for i in range (0, n):

for k in range(0, n):

if(C[i][0] == C[k][1] and C[i][1] == C[k][0]):

if(C\_n[i] != C\_n[k]):

C\_n[k] = C\_n[i]

myset = set(C\_n)

C\_n\_unique = list(myset)

level = list()

level.append(C\_n\_unique)

Main\_matrix = list()

Main\_matrix.append(level)

level = list()

for t\_l in range(0, len(C\_n\_unique)-1):

for y\_l in range(0, len(Main\_matrix[t\_l])):

for k\_l in range(0, len(Main\_matrix[t\_l][y\_l])-1):

temple\_y = Main\_matrix[t\_l][y\_l][0:k\_l+1] + Main\_matrix[t\_l][y\_l][k\_l+2:]

level.append(temple\_y)

level.append(Main\_matrix[t\_l][y\_l][1:])

Main\_matrix.append(level)

level = list()

T = list()

T.append(2)

for o in range(1, len(C\_n\_unique)-1):

T.append((T[0]+o)\*(T[o-1]))

counter\_f = 0

w = 0

Variants\_matrix = list()

for index\_o in Main\_matrix[len(C\_n\_unique)-1]:

path\_list = list()

path\_list.append(index\_o[0])

h = len(C\_n\_unique)-1

temple\_ui = list()

for p\_i in range(1, len(C\_n\_unique)):

Main\_matrix[h-p\_i][0]

number = 0

number = counter\_f // T[p\_i-1]

temple\_ui.append(Main\_matrix[h-p\_i][number])

value\_gotten = list(set(temple\_ui[0]) - set(index\_o))

path\_list.append(int(value\_gotten[0]))

for j in range(1, len(temple\_ui)):

value\_gotten = list(set(temple\_ui[j]) - set(temple\_ui[j-1]))

path\_list.append(int(value\_gotten[0]))

path\_list.reverse()

Variants\_matrix.append(path\_list)

counter\_f = counter\_f + 1

#print(Variants\_matrix)

Final\_sum = list()

Final\_paths = list()

C\_sum\_list = list()

import random

C\_temple = list()

C\_n\_temple = list()

L\_temple = list()

for main\_counter in range(0, len(Variants\_matrix)):

#print(main\_counter)

C\_temple = list()

C\_n\_temple = list()

L\_temple = list()

L\_temple = list()

for t1 in range(0, len(C)):

C\_temple.append(C[t1])

for t2 in range(0, len(C\_n)):

C\_n\_temple.append(C\_n[t2])

for t3 in range(0, len(L)):

L\_temple.append(L[t3])

C\_sum = 0

k\_temple\_counter = 0

index\_path = list()

while(len(C\_n\_temple) != 0):

temple\_list\_variants\_matrix = Variants\_matrix[main\_counter]

po\_counter = 0

for gl in C\_n\_temple:

if(gl == temple\_list\_variants\_matrix[k\_temple\_counter]):

po\_counter = po\_counter + 1

if(po\_counter == 0):

break

random\_number = C\_n\_temple.index(temple\_list\_variants\_matrix[k\_temple\_counter])

k\_temple\_counter = k\_temple\_counter + 1

row\_index = C\_temple[random\_number][0]

column\_index = C\_temple[random\_number][1]

C\_n\_temple\_removed = C\_n\_temple[random\_number]

i = C\_temple[random\_number][0]

j = C\_temple[random\_number][1]

index\_path.append([i,j])

del C\_n\_temple[random\_number]

C\_sum = C\_sum + c\_cost[C\_temple[random\_number][0]][C\_temple[random\_number][1]]

del C\_temple[random\_number]

temple\_index = C\_n\_temple.index(C\_n\_temple\_removed)

del C\_n\_temple[C\_n\_temple.index(C\_n\_temple\_removed)]

del C\_temple[temple\_index]

L\_temple[i] = L\_temple[i] - 1

L\_temple[j] = L\_temple[j] - 1

if (L\_temple[i] == 0 and len(L\_temple)!=0):

o = 0

if (len(C\_temple) != 0):

while(o< len(C\_temple)):

if(C\_temple[o][0]==i or C\_temple[o][1]==i):

del C\_temple[o]

del C\_n\_temple[o]

o = 0

else:

o = o + 1

if (L\_temple[j] == 0 and len(L\_temple)!=0):

o = 0

if (len(C\_temple) != 0):

while(o< len(C\_temple)):

if(C\_temple[o][0]==j or C\_temple[o][1]==j):

del C\_temple[o]

del C\_n\_temple[o]

o = 0

else:

o = o + 1

Final\_paths.append(index\_path)

Final\_sum.append(C\_sum)

C\_sum\_list.append(C\_sum)

print("maximum payoff = " + str(max(Final\_sum)))

index\_final = Final\_sum.index(max(Final\_sum))

print(Final\_paths[index\_final])

## Chapter 3 brute force algorithm

import numpy as np

N = 6

g = [[0, 1, 0, 0, 0, 0],

[1, 0, 1, 0, 0, 0],

[0, 1, 0, 1, 0, 0],

[0, 0, 1, 0, 1, 0],

[0, 0, 0, 1, 0, 1],

[0, 0, 0, 0, 1, 0],]

first\_player\_strategies = list()

second\_player\_strategies = list()

third\_player\_strategies = list()

forth\_player\_strategies = list()

fifth\_player\_strategies = list()

sixth\_player\_strategies = list()

first\_player\_strategies.append('A')

first\_player\_strategies.append('B')

first\_player\_strategies.append('C')

first\_player\_strategies.append('D')

first\_player\_strategies.append('E')

first\_player\_strategies.append('F')

first\_player\_strategies.append('G')

first\_player\_strategies.append('H')

first\_player\_strategies.append('I')

first\_player\_strategies.append('J')

second\_player\_strategies.append('A')

second\_player\_strategies.append('B')

second\_player\_strategies.append('C')

second\_player\_strategies.append('D')

second\_player\_strategies.append('E')

second\_player\_strategies.append('F')

second\_player\_strategies.append('G')

second\_player\_strategies.append('H')

second\_player\_strategies.append('I')

second\_player\_strategies.append('J')

third\_player\_strategies.append('A')

third\_player\_strategies.append('B')

third\_player\_strategies.append('C')

third\_player\_strategies.append('D')

third\_player\_strategies.append('E')

third\_player\_strategies.append('F')

third\_player\_strategies.append('G')

third\_player\_strategies.append('H')

third\_player\_strategies.append('I')

third\_player\_strategies.append('J')

forth\_player\_strategies.append('A')

forth\_player\_strategies.append('B')

forth\_player\_strategies.append('C')

forth\_player\_strategies.append('D')

forth\_player\_strategies.append('E')

forth\_player\_strategies.append('F')

forth\_player\_strategies.append('G')

forth\_player\_strategies.append('H')

forth\_player\_strategies.append('I')

forth\_player\_strategies.append('J')

fifth\_player\_strategies.append('A')

fifth\_player\_strategies.append('B')

fifth\_player\_strategies.append('C')

fifth\_player\_strategies.append('D')

fifth\_player\_strategies.append('E')

fifth\_player\_strategies.append('F')

fifth\_player\_strategies.append('G')

fifth\_player\_strategies.append('H')

fifth\_player\_strategies.append('I')

fifth\_player\_strategies.append('J')

sixth\_player\_strategies.append('A')

sixth\_player\_strategies.append('B')

sixth\_player\_strategies.append('C')

sixth\_player\_strategies.append('D')

sixth\_player\_strategies.append('E')

sixth\_player\_strategies.append('F')

sixth\_player\_strategies.append('G')

sixth\_player\_strategies.append('H')

sixth\_player\_strategies.append('I')

sixth\_player\_strategies.append('J')

K\_function\_1 = np.zeros((10, 10))

K\_function\_2 = np.zeros((10, 10, 10))

K\_function\_3 = np.zeros((10, 10, 10))

K\_function\_4 = np.zeros((10, 10, 10))

K\_function\_5 = np.zeros((10, 10, 10))

K\_function\_6 = np.zeros((10, 10))

consistency = list()

i = 0

g = [[0, 1, 0, 0, 0, 0],

[1, 0, 1, 0, 0, 0],

[0, 1, 0, 1, 0, 0],

[0, 0, 1, 0, 1, 0],

[0, 0, 0, 1, 0, 1],

[0, 0, 0, 0, 1, 0],]

g = g + np.eye(N)

K\_max = 0

max\_strategy = list()

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for second\_player\_counter in range(0, len(second\_player\_strategies)):

for third\_player\_counter in range(0, len(third\_player\_strategies)):

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

consistency = list()

consistency.append(first\_player\_counter)

consistency.append(second\_player\_counter)

consistency.append(third\_player\_counter)

consistency.append(forth\_player\_counter)

consistency.append(fifth\_player\_counter)

consistency.append(sixth\_player\_counter)

K\_current = 0

strategies\_matrix = list()

#for t in range(0, N):

#print(g[t]\*consistency)

#print("\n")

for u in range(0, N):

temple\_strategies\_matrix = list()

for y in range(0, N):

if(g[u][y] == 1):

temple\_strategies\_matrix.append(consistency[y])

strategies\_matrix.append(temple\_strategies\_matrix)

#print(i)

#print(consistency)

#print(strategies\_matrix)

K\_current += K\_function\_1[strategies\_matrix[0][0]][strategies\_matrix[0][1]]

K\_current += K\_function\_2[strategies\_matrix[1][0]][strategies\_matrix[1][1]][strategies\_matrix[1][2]]

K\_current += K\_function\_3[strategies\_matrix[2][0]][strategies\_matrix[2][1]][strategies\_matrix[2][2]]

K\_current += K\_function\_4[strategies\_matrix[3][0]][strategies\_matrix[3][1]][strategies\_matrix[3][2]]

K\_current += K\_function\_5[strategies\_matrix[4][0]][strategies\_matrix[4][1]][strategies\_matrix[4][2]]

K\_current += K\_function\_6[strategies\_matrix[5][0]][strategies\_matrix[5][1]]

i = i + 1

if(K\_max < K\_current):

K\_max = K\_current

max\_strategy = list()

max\_strategy.append(consistency)

#print(first\_player\_counter)

#print(first\_player\_strategies[first\_player\_counter])

print("max sum: ")

print(K\_max)

print("strategy: ")

print(max\_strategy)

## Chapter 3 method of maximizing the payoff functions of players with unspecified strategies

import numpy as np

N = 6

g = [[0, 1, 0, 0, 0, 0],

[1, 0, 1, 0, 0, 0],

[0, 1, 0, 1, 0, 0],

[0, 0, 1, 0, 1, 0],

[0, 0, 0, 1, 0, 1],

[0, 0, 0, 0, 1, 0],]

first\_player\_strategies = list()

second\_player\_strategies = list()

third\_player\_strategies = list()

forth\_player\_strategies = list()

fifth\_player\_strategies = list()

sixth\_player\_strategies = list()

first\_player\_strategies.append('A')

first\_player\_strategies.append('B')

first\_player\_strategies.append('C')

first\_player\_strategies.append('D')

first\_player\_strategies.append('E')

first\_player\_strategies.append('F')

first\_player\_strategies.append('G')

first\_player\_strategies.append('H')

first\_player\_strategies.append('I')

first\_player\_strategies.append('J')

second\_player\_strategies.append('A')

second\_player\_strategies.append('B')

second\_player\_strategies.append('C')

second\_player\_strategies.append('D')

second\_player\_strategies.append('E')

second\_player\_strategies.append('F')

second\_player\_strategies.append('G')

second\_player\_strategies.append('H')

second\_player\_strategies.append('I')

second\_player\_strategies.append('J')

third\_player\_strategies.append('A')

third\_player\_strategies.append('B')

third\_player\_strategies.append('C')

third\_player\_strategies.append('D')

third\_player\_strategies.append('E')

third\_player\_strategies.append('F')

third\_player\_strategies.append('G')

third\_player\_strategies.append('H')

third\_player\_strategies.append('I')

third\_player\_strategies.append('J')

forth\_player\_strategies.append('A')

forth\_player\_strategies.append('B')

forth\_player\_strategies.append('C')

forth\_player\_strategies.append('D')

forth\_player\_strategies.append('E')

forth\_player\_strategies.append('F')

forth\_player\_strategies.append('G')

forth\_player\_strategies.append('H')

forth\_player\_strategies.append('I')

forth\_player\_strategies.append('J')

fifth\_player\_strategies.append('A')

fifth\_player\_strategies.append('B')

fifth\_player\_strategies.append('C')

fifth\_player\_strategies.append('D')

fifth\_player\_strategies.append('E')

fifth\_player\_strategies.append('F')

fifth\_player\_strategies.append('G')

fifth\_player\_strategies.append('H')

fifth\_player\_strategies.append('I')

fifth\_player\_strategies.append('J')

sixth\_player\_strategies.append('A')

sixth\_player\_strategies.append('B')

sixth\_player\_strategies.append('C')

sixth\_player\_strategies.append('D')

sixth\_player\_strategies.append('E')

sixth\_player\_strategies.append('F')

sixth\_player\_strategies.append('G')

sixth\_player\_strategies.append('H')

sixth\_player\_strategies.append('I')

sixth\_player\_strategies.append('J')

K\_function\_1 = np.zeros((10, 10))

K\_function\_2 = np.zeros((10, 10, 10))

K\_function\_3 = np.zeros((10, 10, 10))

K\_function\_4 = np.zeros((10, 10, 10))

K\_function\_5 = np.zeros((10, 10, 10))

K\_function\_6 = np.zeros((10, 10))

first\_strategy\_list = np.zeros(N)

max\_element = K\_function\_1.max()

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for second\_player\_counter in range(0, len(second\_player\_strategies)):

if(K\_function\_1[first\_player\_counter][second\_player\_counter] == max\_element):

first\_strategy\_index = first\_player\_counter

second\_strategy\_index = second\_player\_counter

first\_strategy\_list[0] = first\_strategy\_index

first\_strategy\_list[1] = second\_strategy\_index

K\_max\_3 = 0

second\_index = int(first\_strategy\_list[1])

for third\_player\_counter in range(0, len(third\_player\_strategies)):

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

w = int(K\_function\_3[second\_index][int(third\_player\_counter)][int(forth\_player\_counter)])

if(w > int(K\_max\_3)):

K\_max\_3 = w

forth\_strategy\_index = forth\_player\_counter

third\_strategy\_index = third\_player\_counter

first\_strategy\_list[2] = third\_strategy\_index

first\_strategy\_list[3] = forth\_strategy\_index

K\_max\_5 = 0

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

forth\_index = int(first\_strategy\_list[3])

w = int(K\_function\_5[forth\_index][int(fifth\_player\_counter)][int(sixth\_player\_counter)])

if(w > int(K\_max\_5)):

K\_max\_5 = w

fifth\_strategy\_index = fifth\_player\_counter

sixth\_strategy\_index = sixth\_player\_counter

first\_strategy\_list[4] = fifth\_strategy\_index

first\_strategy\_list[5] = sixth\_strategy\_index

K\_max\_2 = K\_function\_2[int(first\_strategy\_list[0])][int(first\_strategy\_list[1])][int(first\_strategy\_list[2])]

K\_max\_3 = K\_function\_3[int(first\_strategy\_list[1])][int(first\_strategy\_list[2])][int(first\_strategy\_list[3])]

K\_max\_4 = K\_function\_4[int(first\_strategy\_list[2])][int(first\_strategy\_list[3])][int(first\_strategy\_list[4])]

K\_max\_5 = K\_function\_5[int(first\_strategy\_list[3])][int(first\_strategy\_list[4])][int(first\_strategy\_list[5])]

K\_max\_6 = K\_function\_6[int(first\_strategy\_list[4])][int(first\_strategy\_list[5])]

index\_first\_player = int(first\_strategy\_list[0])

index\_second\_player = int(first\_strategy\_list[1])

index\_third\_player = int(first\_strategy\_list[2])

K\_2 = K\_function\_2[index\_first\_player][index\_second\_player][index\_third\_player]

u = 0

player\_2\_pairs = list()

player\_2\_meanings = list()

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for second\_player\_counter in range(0, len(second\_player\_strategies)):

for third\_player\_counter in range(0, len(third\_player\_strategies)):

if(K\_function\_2[first\_player\_counter][second\_player\_counter][third\_player\_counter] > K\_2):

player\_2\_meanings.append(K\_function\_2[first\_player\_counter][second\_player\_counter][third\_player\_counter])

temple\_list\_player\_2 = list()

temple\_list\_player\_2.append(first\_player\_counter)

temple\_list\_player\_2.append(second\_player\_counter)

temple\_list\_player\_2.append(third\_player\_counter)

u = u + 1

player\_2\_pairs.append(temple\_list\_player\_2)

index\_second\_player = int(first\_strategy\_list[1])

index\_third\_player = int(first\_strategy\_list[2])

index\_forth\_player = int(first\_strategy\_list[3])

K\_3 = K\_function\_3[index\_second\_player][index\_third\_player][index\_forth\_player]

u = 0

player\_3\_pairs = list()

player\_3\_meanings = list()

for second\_player\_counter in range(0, len(second\_player\_strategies)):

for third\_player\_counter in range(0, len(third\_player\_strategies)):

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

if(K\_function\_3[second\_player\_counter][third\_player\_counter][forth\_player\_counter] > K\_3):

player\_3\_meanings.append(K\_function\_3[second\_player\_counter][third\_player\_counter][forth\_player\_counter])

temple\_list\_player\_2 = list()

temple\_list\_player\_2.append(second\_player\_counter)

temple\_list\_player\_2.append(third\_player\_counter)

temple\_list\_player\_2.append(forth\_player\_counter)

u = u + 1

player\_3\_pairs.append(temple\_list\_player\_2)

index\_third\_player = int(first\_strategy\_list[2])

index\_forth\_player = int(first\_strategy\_list[3])

index\_fifth\_player = int(first\_strategy\_list[4])

K\_4 = K\_function\_4[index\_third\_player][index\_forth\_player][index\_fifth\_player]

u = 0

player\_4\_pairs = list()

player\_4\_meanings = list()

for third\_player\_counter in range(0, len(third\_player\_strategies)):

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

if(K\_function\_4[third\_player\_counter][forth\_player\_counter][fifth\_player\_counter] > K\_4):

player\_4\_meanings.append(K\_function\_4[third\_player\_counter][forth\_player\_counter][fifth\_player\_counter])

temple\_list\_player\_2 = list()

temple\_list\_player\_2.append(third\_player\_counter)

temple\_list\_player\_2.append(forth\_player\_counter)

temple\_list\_player\_2.append(fifth\_player\_counter)

u = u + 1

player\_4\_pairs.append(temple\_list\_player\_2)

index\_forth\_player = int(first\_strategy\_list[3])

index\_fifth\_player = int(first\_strategy\_list[4])

index\_sixth\_player = int(first\_strategy\_list[5])

K\_5 = K\_function\_5[index\_forth\_player][index\_fifth\_player][index\_sixth\_player]

u = 0

player\_5\_pairs = list()

player\_5\_meanings = list()

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

if(K\_function\_5[forth\_player\_counter][fifth\_player\_counter][sixth\_player\_counter] > K\_5):

player\_5\_meanings.append(K\_function\_5[forth\_player\_counter][fifth\_player\_counter][sixth\_player\_counter])

temple\_list\_player\_2 = list()

temple\_list\_player\_2.append(forth\_player\_counter)

temple\_list\_player\_2.append(fifth\_player\_counter)

temple\_list\_player\_2.append(sixth\_player\_counter)

u = u + 1

player\_5\_pairs.append(temple\_list\_player\_2)

index\_forth\_player = int(first\_strategy\_list[3])

index\_fifth\_player = int(first\_strategy\_list[4])

index\_sixth\_player = int(first\_strategy\_list[5])

K\_6 = K\_function\_6[index\_fifth\_player][index\_sixth\_player]

u = 0

player\_6\_pairs = list()

player\_6\_meanings = list()

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

if(K\_function\_6[fifth\_player\_counter][sixth\_player\_counter] > K\_6):

player\_6\_meanings.append(K\_function\_6[fifth\_player\_counter][sixth\_player\_counter])

temple\_list\_player\_2 = list()

temple\_list\_player\_2.append(fifth\_player\_counter)

temple\_list\_player\_2.append(sixth\_player\_counter)

u = u + 1

player\_6\_pairs.append(temple\_list\_player\_2)

K\_max\_6 = K\_function\_6[int(first\_strategy\_list[4])][int(first\_strategy\_list[5])]

#print(u)

K\_first\_strategy\_max = max\_element + K\_max\_2 + K\_max\_3 + K\_max\_4 + K\_max\_5 + K\_max\_6

final\_strategies\_list = list()

K\_max\_final = list()

counter\_for\_second\_player = 0

for main\_counter\_j in range (0, len(player\_2\_pairs)):

first\_strategy\_i = int(player\_2\_pairs[main\_counter\_j][0])

second\_strategy\_i = int(player\_2\_pairs[main\_counter\_j][1])

third\_strategy\_i = int(player\_2\_pairs[main\_counter\_j][2])

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

K\_all = K\_function\_1[first\_strategy\_i][second\_strategy\_i] + K\_function\_2[first\_strategy\_i][second\_strategy\_i][third\_strategy\_i] + K\_function\_3[second\_strategy\_i][third\_strategy\_i][forth\_player\_counter] + K\_function\_4[third\_strategy\_i][forth\_player\_counter][fifth\_player\_counter] + K\_function\_5[forth\_player\_counter][fifth\_player\_counter][sixth\_player\_counter] + K\_function\_6[fifth\_player\_counter][sixth\_player\_counter]

counter\_for\_second\_player += 1

if(K\_all > K\_first\_strategy\_max):

temple\_strategies\_list = list()

K\_max\_final.append(K\_all)

temple\_strategies\_list.append(first\_strategy\_i)

temple\_strategies\_list.append(second\_strategy\_i)

temple\_strategies\_list.append(third\_strategy\_i)

temple\_strategies\_list.append(forth\_player\_counter)

temple\_strategies\_list.append(fifth\_player\_counter)

temple\_strategies\_list.append(sixth\_player\_counter)

final\_strategies\_list.append(temple\_strategies\_list)

for main\_counter\_j in range (0, len(player\_3\_pairs)):

second\_strategy\_i = int(player\_3\_pairs[main\_counter\_j][0])

third\_strategy\_i = int(player\_3\_pairs[main\_counter\_j][1])

forth\_strategy\_i = int(player\_3\_pairs[main\_counter\_j][2])

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for fifth\_player\_counter in range(0, len(fifth\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

K\_all = K\_function\_1[first\_player\_counter][second\_strategy\_i] + K\_function\_2[first\_player\_counter][second\_strategy\_i][third\_strategy\_i] + K\_function\_3[second\_strategy\_i][third\_strategy\_i][forth\_strategy\_i] + K\_function\_4[third\_strategy\_i][forth\_strategy\_i][fifth\_player\_counter] + K\_function\_5[forth\_strategy\_i][fifth\_player\_counter][sixth\_player\_counter] + K\_function\_6[fifth\_player\_counter][sixth\_player\_counter]

counter\_for\_second\_player += 1

if(K\_all > K\_first\_strategy\_max):

temple\_strategies\_list = list()

K\_max\_final.append(K\_all)

temple\_strategies\_list.append(first\_player\_counter)

temple\_strategies\_list.append(second\_strategy\_i)

temple\_strategies\_list.append(third\_strategy\_i)

temple\_strategies\_list.append(forth\_strategy\_i)

temple\_strategies\_list.append(fifth\_player\_counter)

temple\_strategies\_list.append(sixth\_player\_counter)

final\_strategies\_list.append(temple\_strategies\_list)

print(counter\_for\_second\_player)

for main\_counter\_j in range (0, len(player\_4\_pairs)):

third\_strategy\_i = int(player\_4\_pairs[main\_counter\_j][0])

forth\_strategy\_i = int(player\_4\_pairs[main\_counter\_j][1])

fifth\_strategy\_i = int(player\_4\_pairs[main\_counter\_j][2])

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for second\_player\_counter in range(0, len(second\_player\_strategies)):

for sixth\_player\_counter in range(0, len(sixth\_player\_strategies)):

K\_all = K\_function\_1[first\_player\_counter][second\_player\_counter] + K\_function\_2[first\_player\_counter][second\_player\_counter][third\_strategy\_i] + K\_function\_3[second\_player\_counter][third\_strategy\_i][forth\_strategy\_i] + K\_function\_4[third\_strategy\_i][forth\_strategy\_i][fifth\_strategy\_i] + K\_function\_5[forth\_strategy\_i][fifth\_strategy\_i][sixth\_player\_counter] + K\_function\_6[fifth\_strategy\_i][sixth\_player\_counter]

counter\_for\_second\_player += 1

if(K\_all > K\_first\_strategy\_max):

temple\_strategies\_list = list()

K\_max\_final.append(K\_all)

temple\_strategies\_list.append(first\_player\_counter)

temple\_strategies\_list.append(second\_player\_counter)

temple\_strategies\_list.append(third\_strategy\_i)

temple\_strategies\_list.append(forth\_strategy\_i)

temple\_strategies\_list.append(fifth\_strategy\_i)

temple\_strategies\_list.append(sixth\_player\_counter)

final\_strategies\_list.append(temple\_strategies\_list)

for main\_counter\_j in range (0, len(player\_5\_pairs)):

forth\_strategy\_i = int(player\_5\_pairs[main\_counter\_j][0])

fifth\_strategy\_i = int(player\_5\_pairs[main\_counter\_j][1])

sixth\_strategy\_i = int(player\_5\_pairs[main\_counter\_j][2])

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for second\_player\_counter in range(0, len(second\_player\_strategies)):

for third\_player\_counter in range(0, len(third\_player\_strategies)):

K\_all = K\_function\_1[first\_player\_counter][second\_player\_counter] + K\_function\_2[first\_player\_counter][second\_player\_counter][third\_player\_counter] + K\_function\_3[second\_player\_counter][third\_player\_counter][forth\_strategy\_i] + K\_function\_4[third\_player\_counter][forth\_strategy\_i][fifth\_strategy\_i] + K\_function\_5[forth\_strategy\_i][fifth\_strategy\_i][sixth\_strategy\_i] + K\_function\_6[fifth\_strategy\_i][sixth\_strategy\_i]

counter\_for\_second\_player += 1

if(K\_all > K\_first\_strategy\_max):

temple\_strategies\_list = list()

K\_max\_final.append(K\_all)

temple\_strategies\_list.append(first\_player\_counter)

temple\_strategies\_list.append(second\_player\_counter)

temple\_strategies\_list.append(third\_player\_counter)

temple\_strategies\_list.append(forth\_strategy\_i)

temple\_strategies\_list.append(fifth\_strategy\_i)

temple\_strategies\_list.append(sixth\_strategy\_i)

final\_strategies\_list.append(temple\_strategies\_list)

print(counter\_for\_second\_player)

for main\_counter\_j in range (0, len(player\_6\_pairs)):

fifth\_strategy\_i = int(player\_6\_pairs[main\_counter\_j][0])

sixth\_strategy\_i = int(player\_6\_pairs[main\_counter\_j][1])

for first\_player\_counter in range(0, len(first\_player\_strategies)):

for second\_player\_counter in range(0, len(second\_player\_strategies)):

for third\_player\_counter in range(0, len(third\_player\_strategies)):

for forth\_player\_counter in range(0, len(forth\_player\_strategies)):

K\_all = K\_function\_1[first\_player\_counter][second\_player\_counter] + K\_function\_2[first\_player\_counter][second\_player\_counter][third\_player\_counter] + K\_function\_3[second\_player\_counter][third\_player\_counter][forth\_player\_counter] + K\_function\_4[third\_player\_counter][forth\_player\_counter][fifth\_strategy\_i] + K\_function\_5[forth\_player\_counter][fifth\_strategy\_i][sixth\_player\_counter]

counter\_for\_second\_player += 1

if(K\_all > K\_first\_strategy\_max):

temple\_strategies\_list = list()

K\_max\_final.append(K\_all)

temple\_strategies\_list.append(first\_player\_counter)

temple\_strategies\_list.append(second\_player\_counter)

temple\_strategies\_list.append(third\_player\_counter)

temple\_strategies\_list.append(forth\_player\_counter)

temple\_strategies\_list.append(fifth\_strategy\_i)

temple\_strategies\_list.append(sixth\_strategy\_i)

final\_strategies\_list.append(temple\_strategies\_list)

max\_element\_final = int(max(K\_max\_final))

max\_element\_final\_index = K\_max\_final.index(max\_element\_final)

print("max sum: ")

print(max\_element\_final)

print("strategy: ")

#print(max\_element\_final\_index)

print(final\_strategies\_list[max\_element\_final\_index])

print("number of iterations: ")

print(counter\_for\_second\_player)

Omit the program code for the method “of maximizing the payoff functions of the players with defined strategies” and for method “modification of the method of maximizing the payoff functions of the players with defined strategies” due to their large volume, similarity with the code of the previous method and due to the possibility to find the full code of all methods described in this article on the above link.