Санкт-Петербургский государственный университет
Факультет прикладной математики — процессов управления
Кафедра математической теории игр и статистических решений

Томилина Галина Александровна

Выпускная квалификационная работа бакалавра

Анализ распространения информации в сети

Направление 01.03.02
Прикладная математика и информатика

Заведующий кафедрой,
доктор физ.-мат. наук,
профессор — Петросян Л. А.

Научный руководитель,
кандидат физ.-мат. наук,
старший преподаватель — Кумачёва С. Ш.

Рецензент,
кандидат физ.-мат. наук,
научный сотрудник — Житкова Е. М.

Санкт-Петербург
2018

Содержание

Введение

Одним из основных источников государственного бюджета являются налоги. Все установленные налоги и сборы, правовые нормы их регулирования и взимания образуют налоговую систему страны, а сама процедура установления и уплаты называется налогообложением.

Участниками процесса налогообложения являются фискальные органы и налогоплательщики.

Если формулировать задачи участников налоговой системы с точки зрения теории принятия решений, цели сторон могут быть представлены как задача максимизации выигрышей, представляющих собой личное благосостояние после сбора налогов. В таких условиях фискальные органы стремятся получить максимально возможные сборы, а налогоплательщики — заплатить как можно меньше. Мы рассматриваем рациональных участников процесса, поэтому стремлением некоторых налогоплательщиков к улучшению социального благосостояния в рамках рассматриваемых моделей можно пренебречь.

В рассмотренной ситуации противоречия интересов сторон, очевидно, необходим жесткий контроль выплаты налогов. Традиционные методы этого контроля крайне ресурсоемки, и проведение полного аудита или других стандартных мероприятий может свести на ноль прибыль фискальных органов.

Цель данной работы — построить и проанализировать модели процессов распространения информации, происходящих в сети налогоплательщиков и приводящих к принятию налогоплательщиками решения о выплате налогов или уклонению от них, а также изучить методы влияния на эти процессы.

В качестве возможных вариантов поведения налогоплательщиков будут рассматриваться подражание другим успешным агентам, рациональный анализ распределения выигрышей и отклик на случайное распространение информации.

Сетевая постановка данной задачи позволяет учитывать не только параметры налоговой системы, но и особенности социальной структуры, образованной налогоплательщиками. Изучение социальных связей становится все более возможным и в то же время востребованным в современном мире,

поэтому модели, использующие информацию о них, особенно актуальны.

Для достижения обозначенной цели были сформулированы следующие задачи:

- проанализировать существующие исследования в данной области;
- предложить модели, представляющие описанное распространение информации;
- создать программные реализации моделей для проведения экспериментов;
- провести численные эксперименты и проанализировать полученные результаты.

Основной метод исследования — экспериментальное моделирование.

Постановка задачи

Будем исследовать описанные процессы, рассматривая в качестве налогооблагаемого населения (множества налогоплательщиков) конечное, но большое множество $N = \{1, \ldots, n\}$. Введем в рассмотрение сеть $G = (N, P)$, где P — матрица, характеризующая связи между налогоплательщиками.

Основными характеристиками налогоплательщика являются его уровень дохода и некоторый зависящий от модели параметр, определяющий его стратегию, то есть то, будет ли он уклоняться от уплаты налогов. Также одна из рассматриваемых моделей учитывает психологические свойства налогоплательщиков, влияющие на их склонность к выбору той или иной стратегии.

Вид матрицы P и трактовка распределения стратегий в сети зависит от уточненной постановки подзадачи.

Модель пропорциональной имитации

Как и в работах [1], [2], будем предполагать, что можно разделить налогоплательщиков на две группы по уровню дохода. Эти уровни дохода обозначим через $I = \{L, H\}$, где L — низкий доход, H — высокий. Обозначим заявленный доход экономического агента $R(i) : I \to I$. При уплате налогов налогоплательщик с низким уровнем дохода не имеет возможности уклониться, так как его уровень доходов уже минимально возможный: заявленный доход всегда $R(L) = L$. Налогоплательщик с высоким уровнем дохода может как заявить о низких доходах (уклониться, скрыв часть своих доходов от фискальных органов) $R(H) = L$, так и сообщить истинную информацию $R(H) = H$.

В связи с отсутствием у налогоплательщиков с низким уровнем дохода возможности изменять стратегию, для данной подзадачи можно рассматривать сеть, состоящую только из налогоплательщиков с высоким уровнем дохода.

Пусть налогоплательщики рационально подходят к результатам своих действий, используя для их оценки некоторую матрицу выигрышей A, а также сопоставляют свои доходы с доходами других налогоплательщиков. Так как в данной ситуации нет необходимости учитывать качество отноше-

ний между налогоплательщиками, P — симметричная матрица, в которой $p_{ij} = 0$, если между налогоплательщиками i и j не существует социальной связи, и $p_{ij} = 1$ в противном случае.

Для этой подзадачи имеет место теоретико-игровая постановка, поэтому распределение в сети задано непосредственно в форме стратегий.

Модель случайного распространения информации с двумя уровнями дохода

Пусть I и стратегии, доступные для налогоплательщиков, аналогичны предыдущей модели. Рассматриваем по-прежнему только налогоплательщиков с высоким уровнем дохода.

Пусть налогоплательщики не оценивают последствия своих действий, то есть ожидаемые доходы при выборе той или иной стратегии, и информация распространяется случайным образом. В данном случае P — стохастическая матрица. Элемент этой матрицы $p_{ij} > 0$ в случае, если между налогоплательщиками i и j существует некоторая социальная связь. Значение этого параметра близко к 1, если у i-го агента есть основания предполагать, что агент j обладает экспертными знаниями о вероятности проверки, и близко к 0 в противном случае. Такая матрица позволяет моделировать процесс случайного распространения информации, учитывающий структуру связей в обществе.

Распределение стратегий в сети, представленной таким образом, определяется при помощи «порогового правила», одна из формулировок которого представлена в [3].

Модель случайного распространения информации с тремя уровнями дохода

Предположим теперь, что $I = \{L, M, H\}$, где M — средний уровень дохода. В данной ситуации стратегия налогоплательщика с низким уровнем дохода по-прежнему предопределена. Налогоплательщик со средним уровнем дохода теперь имеет возможность уклониться на одну ступень $R(M) = L$ или не уклониться $R(M) = M$, а налогоплательщик с высоким

уровнем дохода может уклониться на две ступени $R(H) = L$, уклониться на одну ступень $R(H) = M$ или не уклониться $R(H) = H$.

Введем также дополнительную характеристику налогоплательщика — риск-склонность. Под риск-склонностью будем понимать психологические особенности поведения в условиях неопределенности. Будем рассматривать три градации риск-склонности: риск-избегающие налогоплательщики стремятся честно выплачивать налоги вне зависимости от того, мала или велика вероятность проверки, риск-нейтральные ведут себя рационально и принимают решения в зависимости от имеющейся информации, а риск-склонные стремятся избежать выплаты налогов.

В данной ситуации информация распространяется случайным образом, аналогично предыдущей модели, матрица P также сохраняет свои свойства.

Обзор литературы

Для исследования была использована научная и учебно-методическая литература, связанная со случайными процессами, теорией игр и сетевой динамикой.

Для проведения экспериментов использовались данные, приведенные в источниках [4] и [5], а также алгоритмы, описанные T. Kamada, S. Kawai [6] и Y. Koren [7].

Математическое представление характеристик налогоплательщиков опирается на работы P. Chandler, L. Wilde [1], А. Васина, П. Васиной [2], Е. А. Губар, С. Ш. Кумачевой, Е. М. Житковой [8].

Случайное распространение информации

Основными источниками теории по случайным процессам можно назвать книги А. А. Боровкова [9] и М. Де Гроота [10].

Модели случайного распространения информации, представленные в этой работе, опираются прежде всего на модель, исследованную М. Де Гроотом в [11].

Модели влияния на распространение информации в сети изучались, в частности, Д. А. Губановым, Д. А. Новиковым и А. Г. Чхартишвили в [12]. Анализу подобного процесса была также посвящена работа В. М. Буре, В. М. Парилиной и А. А. Седакова [13].

Рациональный анализ распределения выигрышей

В силу того, что мы рассматриваем процесс распространения информации как эволюционный процесс, в данной работе взаимодействие агентов изучается не только с позиции случайных процессов, но и с точки зрения эволюционных игр.

Одним из ключевых источников информации о модели имитации поведения других экономических агентов была книга W. Sandholm [14].

В процессе анализа матриц выигрышей использовался ряд книг по теории игр, в частности, [15], [16], [17], [18], [19], [20].

Анализ специфических эволюционных процессов в сети налогоплательщиков, существенных для данной работы, был проведен, например, в работах [8], [21].

Глава 1. Экспериментальное моделирование

Проблемы визуализации графов и алгоритмы для их решения

Визуализация сетевых структур — нетривиальная задача, так как она ставит перед исследователем ряд вопросов, связанных с индивидуальными особенностями каждого графа. В большинстве случаев невозможно одинаково хорошо отобразить связи между всеми узлами сети, поэтому необходимо оценивать цели визуализации перед её созданием.

Основным вопросом, требующим решения, чаще всего является выбор между хорошо просматриваемым отображением всех имеющихся в сети узлов и корректным отображением структуры связей между ними.

В данной работе для визуализации графов применяются следующие алгоритмы, специфика которых обеспечивает возможность определяться с требованиями к отображению заданной сети во время проведения каждого эксперимента:

- circular, распределяющий узлы сети на эллипсе;
- shell, распределяющий узлы сети на нескольких концентрических эллипсах;
- spectral, использующий в качестве координат узлов сети собственные вектора матричного представления связей между узлами [7];
- spring, моделирующий силы притяжения и отталкивания между узлами и использующий их для определения положения узлов [6].

В рамках создания программного инструмента для работы с рассматриваемыми далее моделями была проведена модификация стандартных реализаций перечисленных алгоритмов для поддержки отображения различных графических элементов. Программный код представлен в приложении.

Детали программной реализации

Для программной реализации модели был использован язык Python 3 (среда JupyterNotebook). Выбор этого языка обусловлен прежде всего гибкими возможностями для визуализации, обеспеченными библиотеками

matplotlib и её оболочкой для графов networkx, а также удобными библиотеками для работы с матрицами и случайными числами. JupyterNotebook дополнительно облегчает представление результатов исследования, интегрируя форматированный текст, код и графики.

С использованием этих инструментов был создан программный код, позволяющий экспериментировать с описанными выше алгоритмами, задавая необходимые параметры. Программа отвечает за запрос и обработку этих параметров, формирование сетевой структуры сети заданной топологии, связей и начального распределения.

Результатом выполнения программы является описание начального и конечного состояний сети и соответствующие им изображения, а также число потребовавшихся итераций. Опционально эти данные автоматически сохраняются в виде отчета.

Программа допускает как индивидуальный ввод параметров для каждого конкретного эксперимента, так и автоматическое формирование отчета по возможным сочетаниям интересующих исследователя параметров.

Рассматриваемые типы сетей

- Случайный граф

  Игроки рассматриваются попарно. Связь между ними образуется с вероятностью $\frac{1}{k}$, где k – натуральное число, заданное при запуске программы. Если связь между агентами i и j есть, p_{ij} и p_{ji} генерируются случайно так, чтобы $p_{ij}, p_{ji} \in [0, 1]$. При этом p_{ij} и p_{ji}, как правило, не совпадают.

  Графы такого типа лучше всего отображаются с помощью алгоритма spring, т.к. здесь нет специфичной структуры связей между узлами и оптимальным можно считать изображение, позволяющее наилучшим образом просматривать наибольшее количество узлов, лишь примерно представляя структуру связей между ними.
- Граф–решетка

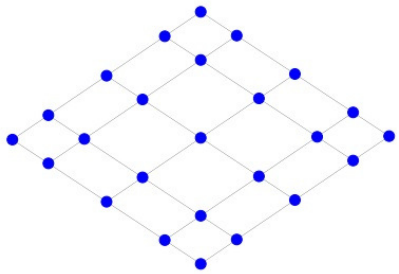

Рис. 1: Пример графа-решетки

Граф–решетка — связный граф, имеющий представление в виде решетки (пример см. рис. 1).

Ребра решетки рассматриваются как двунаправленные, для каждого направления задается свой случайный вес.

Графы такого типа лучше всего отображаются с помощью алгоритма spectral, т.к. он оптимально визуализирует структуру связей между узлами, демонстрируя именно решетку. В случае, когда ребра имеют неодинаковый вес, их форма несколько искажается, но отображение топологии по-прежнему корректно.

- Граф–кольцо

  Для каждого игрока образуется случайная по величине двунаправленная связь с двумя другими игроками.

  Лучший алгоритм визуализации — circular, т.к. он идеально соответствует данному типу графа.

- Граф с почти изолированными игроками

  – На первом этапе формирования генерируется случайный граф, описанный ранее.
  – На втором этапе случайно выбираются m почти изолированных игроков. m задается при запуске программы, $m < \frac{n}{2}$.
  – Для каждого из почти изолированных игроков i выбирается один случайный игрок j, не входящий в число почти изолированных игроков.
  – Все элементы i-ого столбца и i-ой строки матрицы P обнуляются, за исключением p_{ii}, p_{ij} и p_{ji}. Таким образом i-ый агент сообщается только с j-ым.

– Производится поправка на стохастичность матрицы (элементы каждой строки делятся на сумму её элементов).

Выбор алгоритма для отображения такого графа зависит от задачи, поставленной перед исследователем. Для того, чтобы проследить состояние почти изолированных узлов, наилучшим образом подходит алгоритм spring, который отобразит данные узлы на существенном расстоянии от остальных, подчеркивая их свойства. Однако в таком случае очень сложно визуально оценивать динамику остальных узлов, т.к. расстояния между ними, полученные с помощью данного алгоритма, слишком малы, что приводит к наложению узлов друг на друга. Таким образом в случае, когда необходимо оценить динамику всех узлов сети, предпочтительнее алгоритм circular, недостатком которого будет неочевидное отображение структуры связей.

Глава 2. Модель случайного распространения информации

Цель рассмотрения модели, представленной в данном разделе, — изучение процесса распространения информации о вероятности проверки со стороны фискальных органов в сети налогоплательщиков и анализ способов управления этим процессом.

Постановка этой задачи возникает в связи с ресурсоемкостью традиционных методов контроля за уплатой налогов и желанием налоговых органов снизить расходы на проверки, не снижая при этом налоговые сборы. С практической точки зрения выдвигается гипотеза о том, что можно снизить потребность в проведении полного аудита и других затратных мероприятий, заменив их анализом распространения информации о том, что такие мероприятия проводиться будут, и, возможно, влиянием на это распространение.

Модель для популяции с двумя уровнями дохода

Модель для популяции с двумя уровнями дохода рассматривалась в работе [22].

Предположим, что в популяции налогоплательщиков возможны два уровня дохода: низкий и высокий. Под уклонением от уплаты налогов будем понимать поведение налогоплательщика с высоким уровнем дохода, предоставляющего сведения о получении низкого дохода и таким образом укрывающего от налогообложения разницу между реальным доходом и заявленным.

Для упрощения моделирования включим в сеть только налогоплательщиков с высоким уровнем дохода в связи с отсутствием у остальных возможности уклоняться. Пусть в начальный момент времени каждый налогоплательщик из N обладает некоторым представлением о значении вероятности проверки f_i^0. Предположим, что он принимает решение об уклонении от налогов, сопоставляя f_i^0 с параметром p^* — порогом чувствительности всех налогоплательщиков в популяции. В соответствии с «пороговым правилом», одна из формулировок которого представлена в [3], при условии $f_i^0 < p^*$ i-й налогоплательщик уклоняется от уплаты налогов, а

при превышении порога предпочитает не рисковать.

Взаимодействие налогоплательщиков приводит к обновлению их представлений о вероятности проверки на каждой итерации:

$$f_i^k = \sum_{j=1}^{n} p_{ij} f_j^{k-1}. \tag{1}$$

Взаимодействие продолжается бесконечно или до момента, когда при некотором k не будет выполнено $f_i^k \approx f_i^{k-1}$ при $\forall i$.

Предположим теперь, что в естественную сеть могут быть искусственно внедрены информационные центры (аналогично «центрам влияния» в работе [13]) — агенты, стремящиеся «убедить» остальных в некотором значении вероятности проверки.

Роль такого информационного центра может выполнять любой из налогоплательщиков $l \in N$. Пусть S — множество налогоплательщиков, для которых выполняется условие $p_{lj} > 0, l \neq j$. Присвоим информационному центру параметр α_l, выражающий степень его уверенности в значении f_l^0. Тогда после внедрения информационного центра обновленные элементы l-й строки матрицы P будут иметь следующий вид:

$$\begin{gathered} p_{ll} = \alpha_l, \\ p_{lj} = \frac{1-\alpha_l}{|S|}, \quad j \in S, \\ p_{lj} = 0, \quad j \notin S, \quad l \neq j. \end{gathered} \tag{2}$$

С помощью такой модели можно описать, например, стремление налоговых органов завысить представления о вероятности проверки. В таком случае параметры информационного центра $f_l^0 = 1$, $\alpha_l \approx 1$.

Модель для популяции с тремя уровнями дохода

Предположим, что в популяции налогоплательщиков возможны три уровня дохода — низкий (L), средний (M) и высокий (H). В таких условиях уже две категории налогоплательщиков могут уклоняться от уплаты налогов в понимании, представленном в предыдущей модели, причем налогоплательщики с высоким уровнем дохода могут выбирать уже из трех стратегий: не уклоняться, заявить о среднем доходе или заявить о низком доходе. Будем обозначать число налогоплательщиков из определенной ка-

тегории по уклонению через $N(R(i) = j)$, где $j \in I$ — заявленный доход, а $i \in I$ — реальный.

В качестве дополнительного условия введем свойство риск-склонности. Для него будем рассматривать три градации: риск-нейтральные, риск-избегающие и риск-склонные. В соответствии со своим названием, риск-избегающие не уклоняются от уплаты налогов вне зависимости от своего текущего представления о вероятности проверки, риск-нейтральные полностью полагаются на эту вероятность, а риск-склонные предпочитают избегать уплаты налогов до тех пор, пока это возможно.

В такой ситуации будем рассматривать распространение уже двух видов информации: о вероятностях проверки заявивших о низком доходе p_l и проверки заявивших о среднем p_m. Соответственно, каждый налогоплательщик в сети должен иметь представление о значениях каждой из этих вероятностей l_i^k и m_i^k. Процессы информационного обмена аналогичны процессу из предыдущей модели и происходят одновременно.

Информационные центры аналогичным образом распространяют уже два типа информации.

Параметры модели

Приведенные модели позволяют рассматривать процессы случайного распространения информации в сети налогоплательщиков с учетом:

- структуры социальных связей между налогоплательщиками (с оценкой качественных характеристик этих связей)
- распределения уровня доходов между налогоплательщиками
- начального распределения информации в сети
- психологических особенностей налогоплательщиков

Различные комбинации этих параметров могут служить основой для разнообразных экспериментов, моделирующих различные ситуации: как общие и крайние случаи, так и конкретные, рассматриваемые по статистическим данным о некотором сообществе. В частности, можно рассмотреть результаты попытки внедрения в существующую сеть информационного центра и оценить рациональность этого.

Параметры экспериментов

Программная реализация моделей с разными уровнями дохода налогоплательщиков выполнена отдельно, код приведен в приложениях 1 и 2.

Под устойчивым состоянием сети для модели с двумя уровнями дохода подразумевается выполнение условия:

$$\sqrt{\sum_{i=1}^{n}(f_i^k - f_i^{k-1})^2} \leqslant 10^{-3} \tag{3}$$

Аналогичное условие для модели с тремя уровнями дохода имеет следующий вид:

$$\sqrt{\sum_{i=1}^{n}(l_i^k - l_i^{k-1})^2 + \sum_{i=1}^{n}(m_i^k - m_i^{k-1})^2} \leqslant 10^{-3} \tag{4}$$

Запрограммированные алгоритмы поддерживают от нуля до двух информационных центров. Параметры информационных центров задаются вручную в случае использования режима одиночного эксперимента, либо задаются как $f_l^0 = 0{,}99$, $\alpha_l = 0{,}99$ для центра, завышающего вероятность проверки, и $f_l^0 = 0{,}01$, $\alpha_l = 0{,}99$ — для занижающего. Местоположение информационных центров в сети задается случайно.

Значение p^*, используемое во время эксперимента, задается вручную. В приведенных экспериментах были использованы значения 0,4 и 0,6.

Процентное соотношение групп населения с различным уровнем дохода соответствует статистическим данным о доходах в Российской Федерации [4]. Аналогичные данные о риск-склонности получены из [5].

Для каждой из моделей рассматриваются все виды сетей, описанные в разделе «Экспериментальное моделирование».

В модели с двумя уровнями дохода отображаются только налогоплательщики с высоким уровнем дохода: синим — уклоняющиеся от уплаты налогов, желтым — не уклоняющиеся.

Для отображения модели с тремя уровнями дохода и учетом риск-склонности используются следующие обозначения:

- Уровень дохода
  - низкий — круг
  - средний — четырехугольник
  - высокий — треугольник
- Риск-склонность
  - риск-склонные — красный
  - риск-нейтральные — синий
  - риск-избегающие — зеленый
- Уклонение
  - Для среднего уровня дохода
    - неуклоняющиеся — □
    - уклоняющиеся — ◇
  - Для высокого уровня дохода
    - неуклоняющиеся — ▲
    - уклоняющиеся на одну ступень — ◀
    - уклоняющиеся на две ступени — ▼

Примеры

Комбинации параметров рассматриваемых моделей крайне разнообразны. В данном разделе приведены примеры, иллюстрирующие некоторые из сетей с заслуживающей внимания динамикой. Так как модель с тремя уровнями дохода является расширением модели с двумя уровнями, приведенные примеры касаются именно её. Примеры для двух уровней дохода могут быть найдены в работах [22], [23], [24].

Пример 1

Рассмотрим случайную сеть с вероятностью образования связи между попарно рассматриваемыми налогоплательщиками $\frac{1}{3}$ без информационных центров (рис. 2).

В ней 25 налогоплательщиков, из них 6 — с низким уровнем дохода, 14 — со средним, 5 — с высоким. В начальный момент времени все, кроме 9 риск-избегающих налогоплательщиков и налогоплательщиков с низким

уровнем дохода, уклоняются на максимально возможное число ступеней: $N(R(H) = L) = 4, N(R(M) = L) = 12$.

Устойчивое состояние достигается после 7 итераций. В результате обмена информацией все налогоплательщики в данной сети решают не уклоняться от уплаты налогов.

В рассмотренной ситуации использовалось значение $p^* = 0{,}4$. С помощью программного инструмента было также проведено по 50 аналогичных экспериментов с параметрами, перечисленными выше, и $p^* \in \{0{,}4; 0{,}6\}$. Из их анализа можно утверждать, что получаемое в результате число уклоняющихся налогоплательщиков в такой ситуации зависит от соотношения порога чувствительности и средних представлений о вероятности в группе риск-нейтральных налогоплательщиков $\tilde{f}$. Именно риск-нейтральные налогоплательщики имеют такое значение в связи с тем, что их количество в начальный момент наиболее велико. В случае, если $\tilde{f} > p^*$, налогоплательщики в связанной сети приходят к тому, что уклоняющихся от уплаты налогов нет, в противном случае все уклоняются.

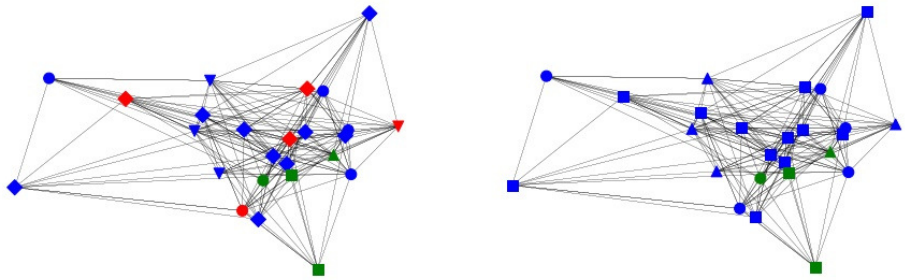

Рис. 2: Начальное и устойчивое распределение для сети без информационных центров

Пример 2

В случаях, когда в сети присутствует только один информационный центр, при условии наличия связей между ним и другими налогоплательщиками, его представление об информации распространяется на другие узлы по определению.

Интерес может представлять динамика сети в случае, когда параметры информационного центра $p_l = 0{,}99, p_m = 0{,}01, \alpha = 0{,}99$. Практическое

значение этих параметров состоит в том, что некоторый центр «утверждает», что налогоплательщиков, заявивших о низком доходе, будут проверять с очень высокой вероятностью, в то время как заявившие о среднем, скорее всего, смогут избежать проверки.

Пусть имеется сеть–решетка из 25 налогоплательщиков (рис. 3). Из тех же соображений, что и в примере 1, изначально имеется 6 экономических агентов с низким уровнем дохода, 14 — со средним, 5 — с высоким. Распределение риск-избегающих в сети задается случайно, поэтому для этого примера оно уже другое: в начальный момент времени 10 налогоплательщиков не уклоняются от уплаты налогов, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.

Для достижения устойчивого состояния такой сети потребовалось 77 итераций, что можно объяснить относительно небольшой связностью между узлами решетки. В устойчивом состоянии не уклоняется от уплаты налогов 21 налогоплательщик, то есть все налогоплательщики с низким и средним уровнями дохода, а также один риск-избегающий, не попавший в их число. Уклоняющиеся же налогоплательщики все относятся к группе $N(R(H) = M) = 4$ — приняв информацию, распространяемую информационным центром, они изменили свою стратегию и уклоняются в результате только на одну ступень.

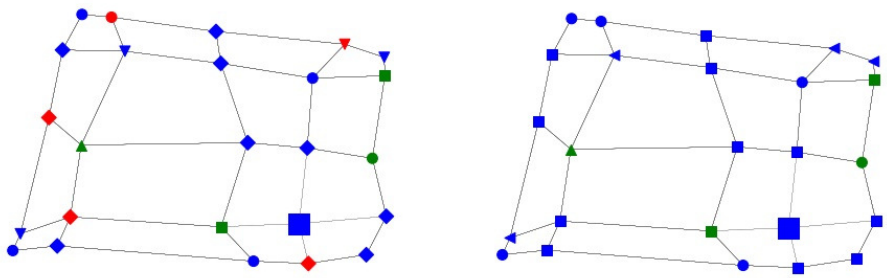

Рис. 3: Начальное и устойчивое распределение для сети с одним информационным центром

В случаях, когда можно утверждать, что в сети нет второго информационного центра (случайно образованного некоторым уверенным в себе налогоплательщиком или созданного намеренно), искусственный информационный центр — мощный способ воздействия на распространение информации, с помощью которого можно получить необходимые фискальным органам результаты.

Пример 3

Особый интерес может представлять взаимодействие двух и более информационных центров. Рассмотрим два случая, в которых действие двух информационных центров прямо противоположно: один распространяет информацию $p_l = 0{,}99, p_m = 0{,}99$, а другой $p_l = 0{,}01, p_m = 0{,}01$.

В качестве первого из двух примеров такого типа рассмотрим случайную сеть с вероятностью образования связи между попарно рассматриваемыми налогоплательщиками $\frac{1}{3}$ (рис. 4). Распределение доходов и уклоняющихся прежнее.

Через 33 итерации достигается устойчивое состояние, в котором 24 налогоплательщика — все, кроме второго информационного центра, — не уклоняются от уплаты налогов. Подобная ситуация выгодна для фискальных органов, поэтому по-прежнему можно утверждать, что внедрение информационного центра эффективно.

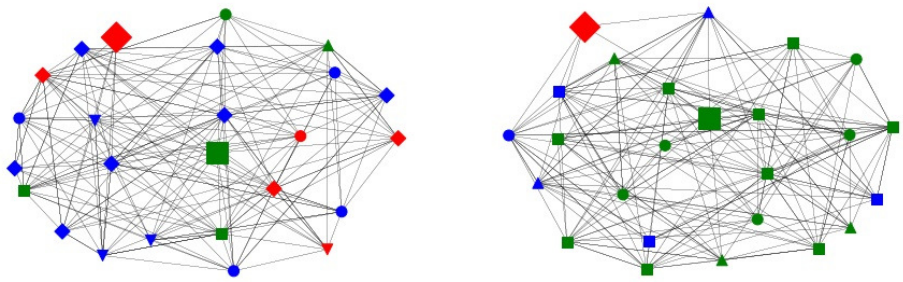

Рис. 4: Начальное и устойчивое распределение для сети с двумя информационными центрами

Пример 4

В похожих условиях рассмотрим сеть–решетку из 25 налогоплательщиков (рис. 5). Распределение доходов среди них сохраняется. В начальный момент времени 10 налогоплательщиков не уклоняются от уплаты налогов, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.

Через 54 итерации достигается устойчивое состояние, в котором не уклоняются 17 налогоплательщиков, $N(R(H) = L) = 2$, $N(R(M) = L) = 6$.

Данный пример демонстрирует, что в случае, когда в сети уже есть каким-то образом сформировавшийся информационный центр (в случае

некоторых типов топологии это не обязательно один налогоплательщик, иногда можно рассматривать целую группу), результаты искуственного создания информационного центра могут быть не такими однозначными, как кажется из его параметров. Из этого следует необходимость анализа сетевой структуры среды, в которой распространяется информация, и психологических особенностей рассматриваемых лиц.

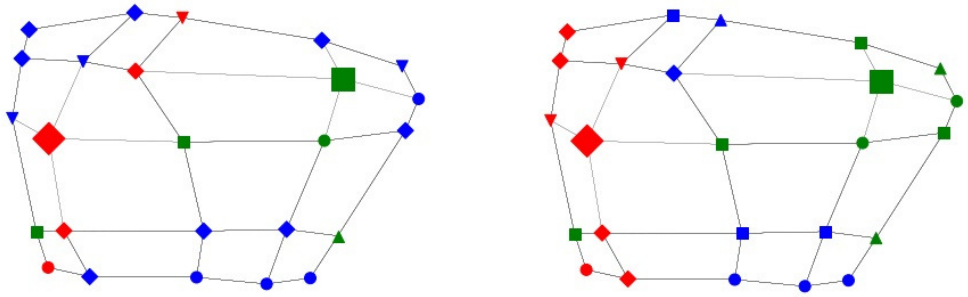

Рис. 5: Начальное и устойчивое распределение для сети с двумя информационными центрами

Иллюстративные примеры

Следующие несколько примеров (рис. 6–18) приводятся в качестве иллюстрации работы программного инструмента и в подтверждение сделанных выше выводов.

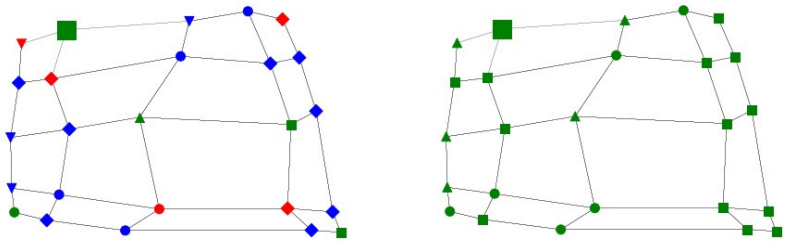

Рис. 6: Дана сеть-решетка из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 114.
Устойчивое состояние: не уклоняются 25

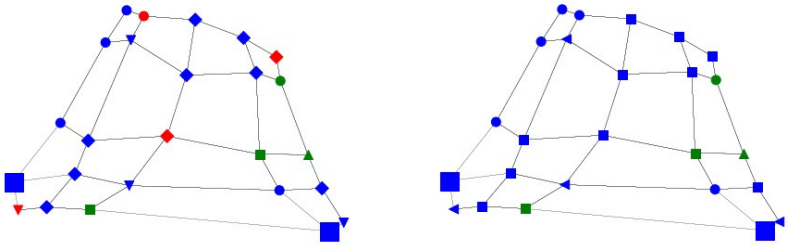

Рис. 7: Дана сеть-решетка из 25 налогоплательщиков.
Начальное состояние: не уклоняются 11, $N(R(H) = L) = 4$, $N(R(M) = L) = 10$.
Итераций: 115.
Устойчивое состояние: не уклоняются 21, $N(R(H) = M) = 4$

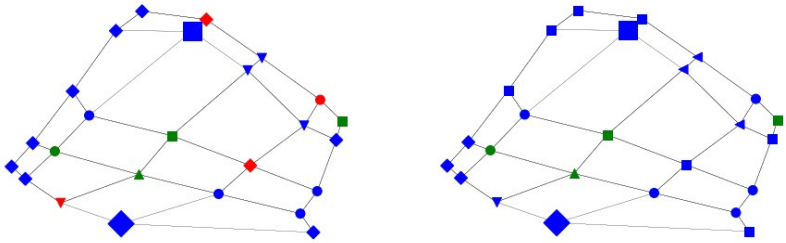

Рис. 8: Дана сеть-решетка из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 103.
Устойчивое состояние: не уклоняются 17, $N(R(H) = L) = 1$, $N(R(M) = L) = 4$, $N(R(H) = M) = 3$

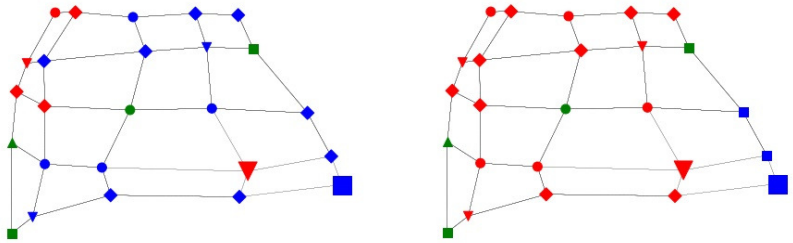

Рис. 9: Дана сеть-решетка из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 475.
Устойчивое состояние: не уклоняются 12, $N(R(H) = L) = 4$, $N(R(M) = L) = 9$

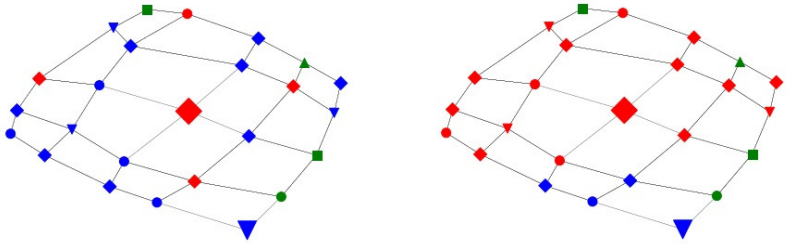

Рис. 10: Дана сеть-решетка из 25 налогоплательщиков.
Начальное состояние: не уклоняются 9, $N(R(H) = L) = 4$, $N(R(M) = L) = 12$.
Итераций: 61.
Устойчивое состояние: не уклоняются 9, $N(R(H) = L) = 4$, $N(R(M) = L) = 12$

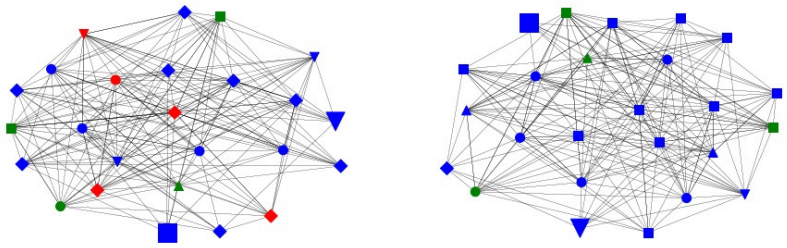

Рис. 11: Дана сильно связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 11.
Устойчивое состояние: не уклоняются 22, $N(R(H) = L) = 2$, $N(R(M) = L) = 1$

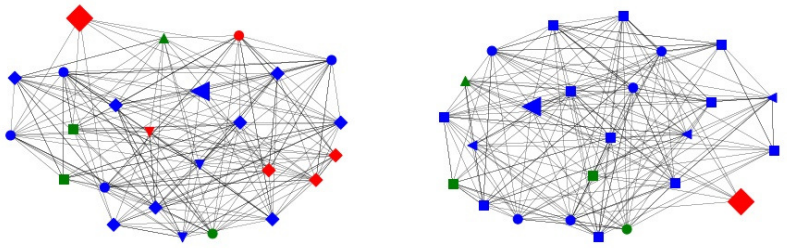

Рис. 12: Дана сильно связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 9, $N(R(H) = L) = 3$, $N(R(M) = L) = 12$.
Итераций: 39.
Устойчивое состояние: не уклоняются 20, $N(R(M) = L) = 1$, $N(R(H) = M) = 4$

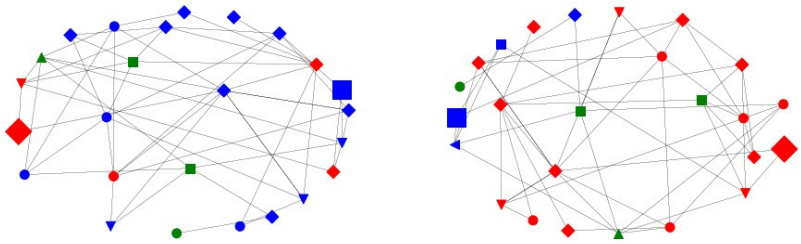

Рис. 13: Дана слабо связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 19.
Устойчивое состояние: не уклоняются 11, $N(R(H) = L) = 3$, $N(R(M) = L) = 10$, $N(R(H) = M) = 1$

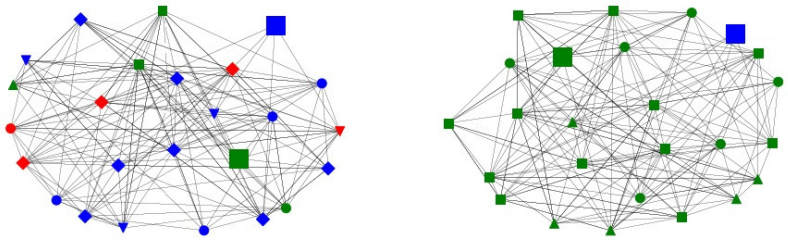

Рис. 14: Дана сильно связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 11, $N(R(H) = L) = 4$, $N(R(M) = L) = 10$.
Итераций: 47.
Устойчивое состояние: не уклоняются 25

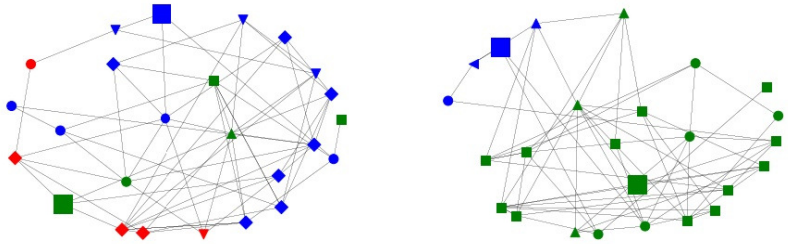

Рис. 15: Дана слабо связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 11, $N(R(H) = L) = 4$, $N(R(M) = L) = 10$.
Итераций: 34.
Устойчивое состояние: не уклоняются 24, $N(R(H) = M) = 1$

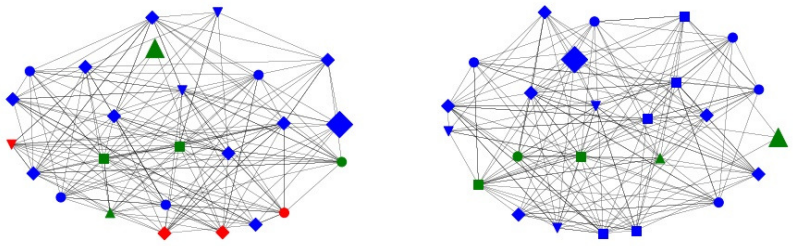

Рис. 16: Дана сильно связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 3$, $N(R(M) = L) = 12$.
Итераций: 39.
Устойчивое состояние: не уклоняются 15, $N(R(H) = L) = 3$, $N(R(M) = L) = 7$

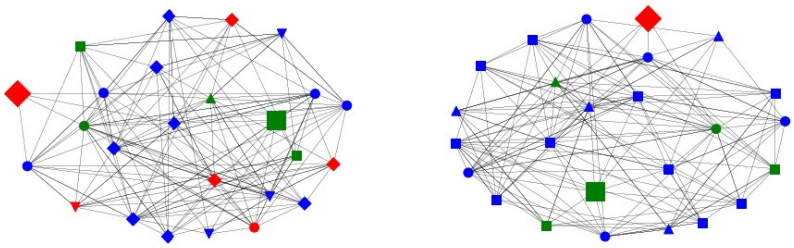

Рис. 17: Дана сильно связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 23.
Устойчивое состояние: не уклоняются 24, $N(R(M) = L) = 1$

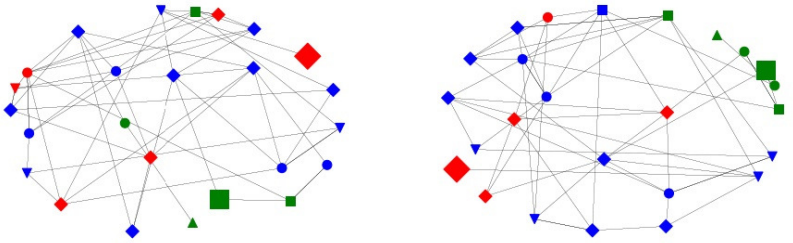

Рис. 18: Дана слабо связанная сеть из 25 налогоплательщиков.
Начальное состояние: не уклоняются 10, $N(R(H) = L) = 4$, $N(R(M) = L) = 11$.
Итераций: 25.
Устойчивое состояние: не уклоняются 11, $N(R(H) = L) = 4$, $N(R(M) = L) = 10$

Результаты

В данном разделе была рассмотрена модель информационного обмена, основаная на естественных социальных предпосылках.

Были описаны свойства случайного распространения информации в сети налогоплательщиков, не подверженной влиянию извне. Обнаружена связь между итоговым распределением стратегий и соотношением представления об информации в группе риск-нейтральных налогоплательщиков и порога чувствительности.

В результате также был предложен метод влияния на процесс информационного обмена со стороны фискальных органов с помощью информационных центров. Проанализированы возможные исходы такого влияния. Показано, что на результаты оказывает существенное влияние структура социальных связей между налогоплательщиками, исходное распределение представлений о распространяемой информации и психологические особенности налогоплательщиков.

В рамках развития данной модели интересными являются, например, оценка благосостояния системы до и после процесса информационного обмена (аналогично работе [8]) и алгоритмизация принятия решения о целесообразности влияния на некоторую сеть при заданных ресурсных ограничениях.

Глава 3. Модель пропорциональной имитации

Цель рассмотрения модели, представленной в данном разделе — в некотором виде ввести в распространение информации между налогоплательщиками рациональное принятие решений.

Постановка этой цели возникает в связи с тем, что случайное распространение информации находится не только под влиянием связей между налогоплательщиками и их психологическими особенностями. Информационный агент может иметь возможность оценить результаты применения стратегии, основанной на полученной информации, и сопоставить, по некоторым социальным признакам или точным данным, её с результатами тех налогоплательщиков, с которыми у него установлены связи. В связи с естественными предпосылками в данной модели предполагается, что существует вероятность того, что в случае, если налогоплательщик видит, что стратегия другого налогоплательщика приносит больший выигрыш, он может сменить свою стратегию.

Модель, лежащая в основе представленного здесь исследования, изначально рассматривалась в [21].

Модель

В рамках данной модели вновь рассматривается процесс распространения информации с учетом социальных связей между участниками.

Пусть допустимы два уровня дохода налогоплательщиков. Аналогично модели случайного распространения информации с двумя уровнями дохода, будем рассматривать только налогоплательщиков с высоким уровнем дохода. В таком случае каждый из рассматриваемых налогоплательщиков имеет возможность выбирать из двух стратегий $X = \{1, 2\}$, где 1 — не уклоняться от уплаты налогов, а 2 — уклоняться. При этом существует некоторая матрица выигрышей A, элементы которой характеризуют результаты выбора той или иной стратегии и взаимодействия с другим налогоплательщиком, также принявшим свое решение. Один из вариантов построения такой матрицы представлен в работе [21].

В каждый момент времени каждый налогоплательщик оценивает выигрыш в результате текущего выбора своей стратегии. Предположим, что эта оценка может быть получена по формуле:

$$u_i = \phi_i \sum_{j \in M_i} a_{x_i(t), x_j(t)} \tag{5}$$

Здесь $x_i(t) \in X$ — стратегия налогоплательщика i в момент t, M_i — множество налогоплательщиков–соседей, то есть тех налогоплательщиков, с которыми у налогоплательщика i есть связи, $a_{x_i(t), x_j(t)}$ — элемент матрицы A, характеризующий выигрыш налогоплательщика i в результате взаимодействия с соседом j, а ϕ_i — весовой коэффициент, определяющий тип подсчета выигрыша. Далее в текущей работе будет рассматриваться $\phi_i = \frac{1}{|M_i|}$ для получения среднего выигрыша и $\phi_i = 1$ — для накопительного.

Таким образом подсчитывается выигрыш каждого игрока, после чего происходит обновление стратегий. Можно рассматривать различные правила, по которым происходит это обновление, подробнее [14]. В данной работе рассматривается правило пропорциональной имитации стратегии случайного соседа, имеющее следующий вид:

$$p(x_i(t+1) = x_j(t)) = \left[\frac{\lambda}{|M_i|} (u_j(t) - u_i(t)) \right]_0^1 \tag{6}$$

Здесь $[z]_0^1 = max(0, min(1, z))$, $j \in M_i$ — случайно выбранный сосед.

Параметры модели

Приведенная модель позволяет рассматривать процесс распространения информации в сети налогоплательщиков с учетом:

- начального распределения информации в сети налогоплательщиков
- структуры связей между налогоплательщиками
- рациональной с точки зрения индивидуального налогоплательщика оценки результатов выбора стратегии

Таким образом мы уходим от возможности учитывать особенности отношений между налогоплательщиками и их личные психологические особенности, но вводим гипотезу об их частичной рациональности, применимую

в реальных ситуациях.

Особенно сложен вопрос построения матрицы выигрыша A, потому что на практике налогоплательщик сопоставляет результаты своего выбора стратегии с результатами выбора соседа по более естественным признакам, создающим затруднения на пути математического моделирования. В данной работе будут рассматриваться несколько вариантов матрицы A, типичных для подобных задач, но на данный момент не имеющих выражения через параметры сети налогоплательщиков.

Значительную роль в результатах эксперимента играет также выбор типа подсчета выигрыша (значения ϕ_i) и значения λ.

Параметры экспериментов

В качестве матрицы выигрыша A в данной работе рассматриваются несколько примеров типовых матриц. Численное представление приведено по [15].

Дилемма заключенного

Матрица игры «Дилемма заключенного» в численном выражении в виде (7) используется на основании того, что в классической экономической интерпретации (здесь по [19]) первая, миролюбивая стратегия агента заключается в кооперации, а вторая — в агрессии: в случае рассматриваемой модели это уклонение от уплаты налогов. Предположим, что при вычислении матрицы выигрышей учитываются некие социальные блага, доступные налогоплательщикам в случае успешного сбора налогов. В таком случае первая стратегия агента выгодна не только фискальным органам, но и налогоплательщикам, так как она увеличивает их доходы. Однако вторая стратегия, уклонение от выплаты налогов, выгоднее (при условии кооперации остальных) в этой ситуации лично для уклоняющегося налогоплательщика, поэтому он неуклонно стремится к ней.

$$\begin{pmatrix} 4 & -1 \\ 5 & 0 \end{pmatrix} \tag{7}$$

Охота на оленя

Матрица игры «Охота на оленя» используется в численном выражении в виде (8). Её использование обосновано классическим описанием конфликта личного и общественного блага [16]. Пусть условия в государстве таковы, что комбинация социальных преимуществ, полученных в результате успешного сбора налогов, может увеличить личное благосостояние налогоплательщиков в среднем на большую сумму, чем та, которую они могли бы получить в результате уклонения от налогов. Однако выигрыш от уклонения для налогоплательщика гарантирован, в то время как успешность сбора налогов в целом зависит от него лишь частично. Поэтому в зависимости от своих психологических особенностей и характера взаимоотношения с окружающими часть налогоплательщиков может уклониться от стратегии уплаты налогов, возможно, лишая выигрыша остальных.

$$\begin{pmatrix} 3 & 0 \\ 1 & 1 \end{pmatrix} \tag{8}$$

Ястребы и голуби

Матрица игры «Ястребы и голуби» представлена в численном выражении в виде (9). Её использование может быть обосновано возможностью представления взаимодействия экономических агентов в виде эволюционной игры, как, например, в [21].

$$\begin{pmatrix} -1 & 4 \\ 0 & 2 \end{pmatrix} \tag{9}$$

Программный инструмент также позволяет ввести матрицу самостоятельно.

Рассматриваемые типы сетей описаны в разделе «Экспериментальное моделирование». Начальное распределение стратегий генерируется одним из двух методов:

- случайно — индивидуально рассматривается каждый налогоплательщик: для него генерируется случайное число из выборки $\{0, 1\}$, в случае выпадения 1 налогоплательщик уклоняется от уплаты налогов,

0 — нет (для больших сетей обеспечивает равномерное распределение стратегий)

- с заданными пропорциями — пользователь указывает долю налогоплательщиков, не уклоняющихся от выплаты налогов. С помощью этого параметра и общего числа налогоплательщиков вычисляется число неуклоняющихся налогоплательщиков, случайно выбирается это число индексов в сети, после чего для узлов, находящихся по выбранным индексам, задается свойство не уклонения от уплаты налогов.

Под устойчивым состоянием в данной модели подразумевается состояние, в котором выполняется условие:

$$\sqrt{\sum_{i=1}^{n}(x_i(t) - x_i(t+1))} \leqslant 10^{-3} \tag{10}$$

В связи со значениями, которые может принимать $x(t)$, данное условие соответствует условию на неизменность стратегий в результате итерации. Оно имеет приведенный вид с целью учета погрешности операций с числами с плавающей точкой на вычислительных системах.

Для выбора λ используется соотношение, приведенное в [21]:

$$\frac{\lambda_i}{|M_i|} = 1$$

Для адаптации этого правила для большого числа рассматриваемых налогоплательщиков, используются либо λ_i, индивидуальные для каждого налогоплательщика, либо $\lambda = \frac{1}{n}\sum_{i=1}^{n}\lambda_i$.

Алгоритм визуализации выбирается в зависимости от типа сети, подробнее в разделе «Рассматриваемые типы сетей». Каждый налогоплательщик отображается в виде узла сети с цветом, зависящим от его стратегии: желтым отображаются не уклоняющиеся от уплаты налогов агенты, синим — уклоняющиеся.

Программный инструмент допускает автоматическое проведение экспериментов с сохранением отчетов.

Код приведен в приложении 3.

Примеры

Автоматическое выполнение экспериментов с различными параметрами выдает отчет о 72 различных ситуациях. Каждая из этих ситуаций — только пример для целого класса возможных сетей. В данном разделе приведены выразительные примеры некоторых из этих ситуаций с различными матрицами выигрыша.

Пример 1.

Первый пример иллюстрирует типичную ситуацию, формирующуюся в случае использования матрицы игры «Дилемма заключенного» (7).

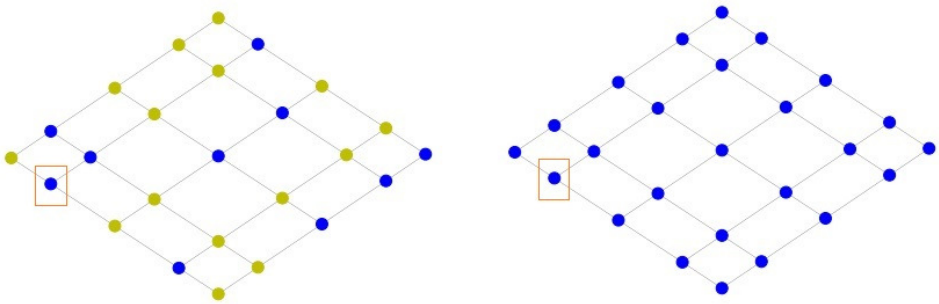

Рис. 19: Начальное и устойчивое распределение для ситуации с матрицей игры «Дилемма заключенного»

Рассматривается сеть–решетка из 25 налогоплательщиков.

В начальный момент времени в сети 10 уклоняющихся налогоплательщиков и 15 — неуклоняющихся. После шести итераций достигается устойчивое состояние (10), в котором все налогоплательщики уклоняются (см. рис. 19).

Используя матрицу выигрышей, сведения о связях между агентами, формулу (5) и значения ϕ_i, соответствующие средним выигрышам, программа подсчитывает выигрыши всех агентов на каждой итерации. Можно видеть, что в начальный момент времени, например, $u_1 = -1$, а $u_{16} = 4$, в то время как в конечный $u_i = 0, \forall i$. Рассматривая выигрыши всех налогоплательщиков, можно увидеть, что налогоплательщик с индексом 1 (на рис. 19 он выделен рамкой) — единственный налогоплательщик с улучшившимся положением.

При использовании данной матрицы ситуация, в которой в конечный момент времени все налогоплательщики уклоняются и получают выигрыш 0, имеет место в случае, если в начальный момент времени в сети есть хоть один уклоняющийся налогоплательщик. Этот результат теоретически обоснован, исходя из свойств дилеммы заключенного, равновесие в которой достигается, в нашем толковании, в случае, когда оба взаимодействующих агента выбирают стратегию уклонения от уплаты налогов [17].

Две другие рассматриваемые матрицы уже не обладают свойством однозначного влияния на исход взаимодействия налогоплательщиков, поэтому в случае с их использованием сетевая структура и начальное распределение стратегий имеют значение. Именно это будет проиллюстрировано в следующих примерах.

Пример 2.

Примером удачного для налогоплательщиков процесса может служить следующая ситуация. Приведена сеть–решетка из 25 налогоплательщиков, используется матрица выигрышей игры «Охота на оленя» (8).

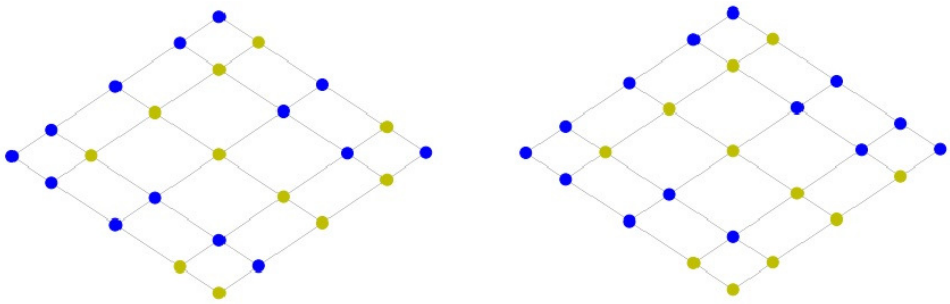

Рис. 20: Начальное и устойчивое распределение для ситуации с матрицей игры «Охота на оленя»

В начальный момент времени, как и в конечный, в сети 14 уклоняющихся и 11 неуклоняющихся налогоплательщиков. В результате всего двух итераций было достигнуто устойчивое состояние (10), в котором изменили свои стратегии только налогоплательщики с индексами 9 и 23 (см. рис. 20). Аналогично примеру 1, можно рассмотреть средние выигрыши, подсчитанные программой. Изменение стратегий всего двух агентов оказало позитивное влияние на следующие средние выигрыши: в начальный

момент $u_5 = 1{,}5$, $u_{10} = 1$, $u_{15} = 2$, $u_{24} = 0$, а в конечный момент $u_5 = 3$, $u_{10} = 2$, $u_{15} = 3$, $u_{24} = 1$. Выигрыши остальных агентов не изменились.

Анализируя аналогичные примеры, можно увидеть, что использование матрицы подобного типа в большинстве случаев приводит к улучшению благосостояния налогоплательщиков, увеличивая их выигрыши в конечный момент времени.

Однако с точки зрения фискальных органов результаты в различных сетях в данном случае существенно отличаются: возможны как подобные промежуточные варианты распределения стратегий, так и ситуации, когда все налогоплательщики в конечный момент времени уклоняются либо не уклоняются от уплаты налогов. Поэтому данная ситуация может представлять интерес для налоговой системы, так как именно анализ сетевой структуры среды налогоплательщиков может помочь ей предсказать результаты естественного развития процесса или смоделировать результаты влияния на систему путем принудительного изменения стратегий некоторых из налогоплательщиков.

Пример 3.

Особой нестабильностью результатов отличается матрица игры «Ястребы и голуби» (9).

Рассмотрим, например, сеть из 25 налогоплательщиков, связи между которыми образованы случайно при вероятности образования связи между попарно рассматриваемыми узлами $\frac{1}{3}$ (см. рис. 21).

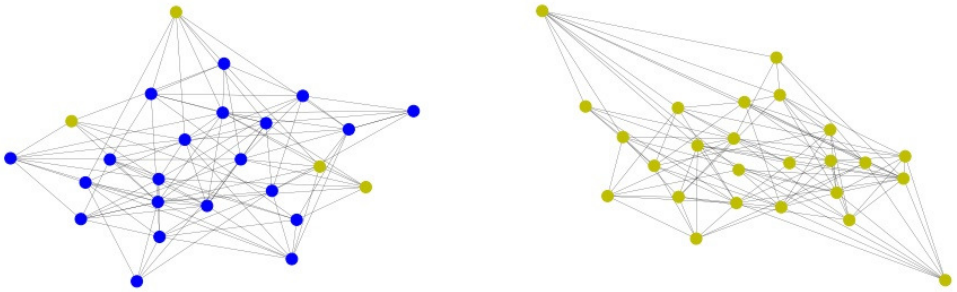

Рис. 21: Начальное и устойчивое распределение для ситуации с матрицей игры «Ястребы и голуби»

В начальный момент в сети 21 уклоняющийся и 4 неуклоняющихся

налогоплательщика.

Число итераций: 18.

Конечное распределение в устойчивом состоянии (10): 25 не уклоняющихся налогоплательщиков.

Подсчетом выигрышей отдельно взятых налогоплательщиков можно получить, что их попытки увеличить свой выигрыш имитацией поведения более успешных соседей значительно ухудшили итоговое положение. Рассмотрим в этот раз накопительный выигрыш агентов, рассчитываемый автоматически по формуле (5) со значением $\phi_i = 1$. Можно увидеть, что выигрыш игрока с индексом 6 в начальный момент времени $u_6 = 28$, а в конечный момент $u_6 = -7$. Аналогично шестому налогоплательщику, в рассматриваемой сети не было ни одного налогоплательщика, результат которого улучшился бы в результате взаимодействия с соседями.

Для ситуаций с подобными матрицами имеется позитивная с точки зрения фискальных органов тенденция к увеличению числа не уклоняющихся от уплаты налогов агентов. Однако для налогоплательщиков результаты процесса взаимодействия при данной матрице выигрышей чаще всего не являются желательными.

Аналогично с матрицей игры «Охота на оленя» (8) данный вариант поддается влиянию со стороны фискальных органов, причем здесь очень велика вероятность аккуратно проанализированным воздействием получить близкое к идеальному с их точки зрения. Стоит, однако, отметить, что в более сложных моделях необходимо также накладывать ограничения на минимальный выигрыш налогоплательщиков, так как в случае его неограниченного убывания социальные последствия не будут выгодны для системы.

Иллюстративные примеры

Следующие несколько примеров (рис. 22–34) приводятся в качестве иллюстрации работы программного инструмента и в подтверждение сделанных выше выводов.

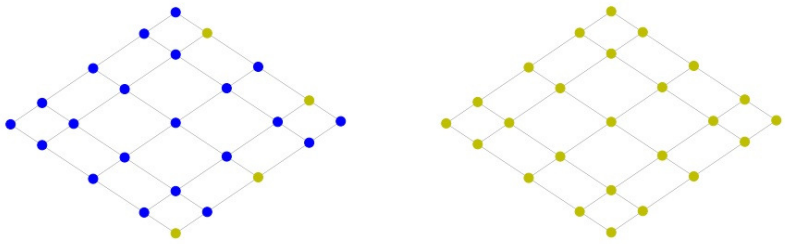

Рис. 22: Дана сеть-решетка из 25 налогоплательщиков. Используется матрица игры «Ястребы и голуби».
Начальное распределение: 21 уклоняющихся, 4 неуклоняющихся.
Число итераций: 25.
Конечное распределение: 0 уклоняющихся, 25 неуклоняющихся

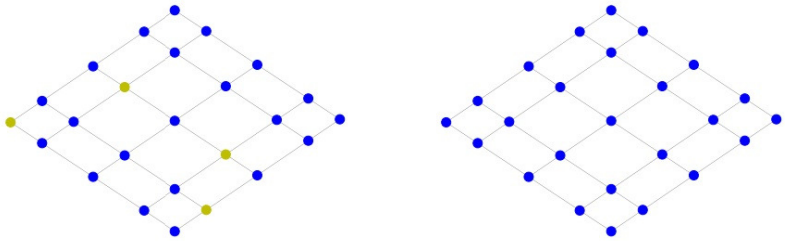

Рис. 23: Дана сеть-решетка из 25 налогоплательщиков. Используется матрица игры «Охота на оленя».
Начальное распределение: 21 уклоняющихся, 4 неуклоняющихся.
Число итераций: 2.
Конечное распределение: 25 уклоняющихся, 0 неуклоняющихся

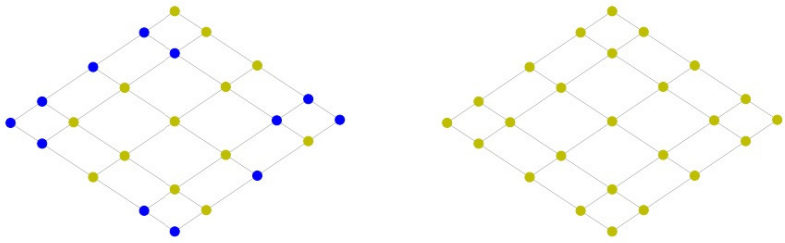

Рис. 24: Дана сеть-решетка из 25 налогоплательщиков. Используется матрица игры «Охота на оленя».
Начальное распределение: 12 уклоняющихся, 13 неуклоняющихся.
Число итераций: 4.
Конечное распределение: 0 уклоняющихся, 25 неуклоняющихся

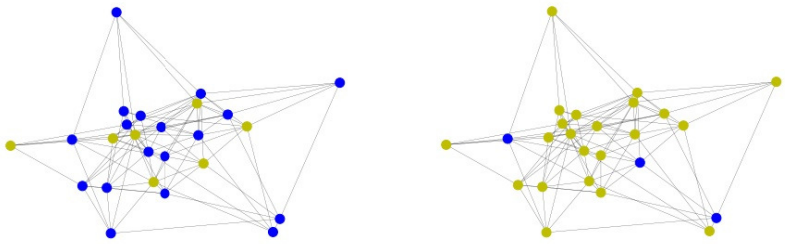

Рис. 25: Дана сильно связанная сеть из 25 налогоплательщиков. Используется матрица игры «Ястребы и голуби».
Начальное распределение: 18 уклоняющихся, 7 неуклоняющихся.
Число итераций: 11.
Конечное распределение: 3 уклоняющихся, 22 неуклоняющихся

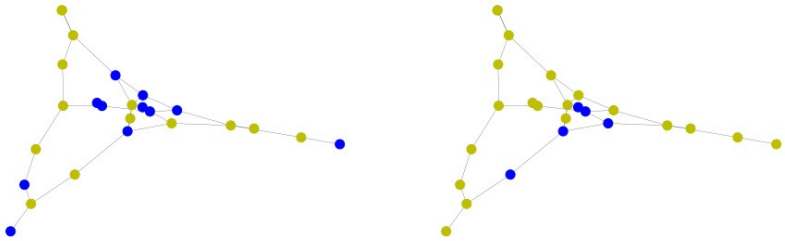

Рис. 26: Дана слабо связанная сеть из 25 налогоплательщиков. Используется матрица игры «Охота на оленя».
Начальное распределение: 11 уклоняющихся, 14 не уклоняющихся.
Число итераций: 4.
Конечное распределение: 5 уклоняющихся, 20 не уклоняющихся

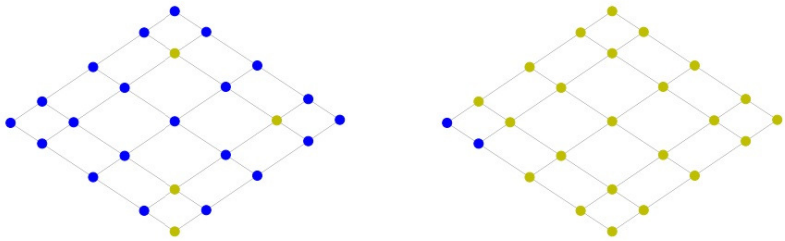

Рис. 27: Дана сеть-решетка из 25 налогоплательщиков. Используется матрица игры «Дилемма заключенного».
Начальное распределение: 9 уклоняющихся, 16 неуклоняющихся.
Число итераций: 15.
Конечное распределение: 25 уклоняющихся, 0 неуклоняющихся

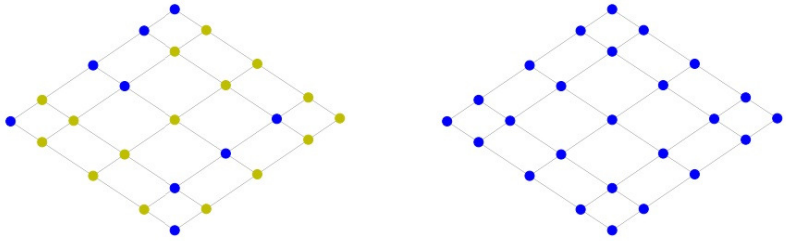

Рис. 28: Дана сеть-решетка из 25 налогоплательщиков. Используется матрица игры «Ястребы и голуби».
Начальное распределение: 21 уклоняющихся, 4 неуклоняющихся.
Число итераций: 25.
Конечное распределение: 2 уклоняющихся, 23 неуклоняющихся

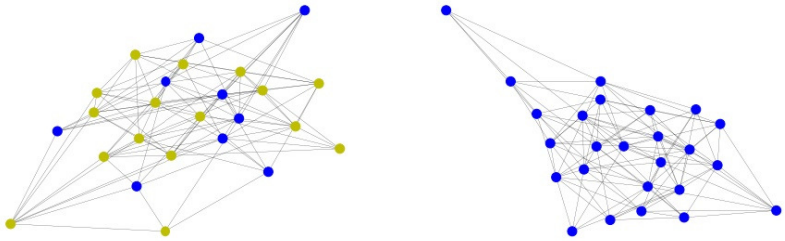

Рис. 29: Дана сильно связанная сеть из 25 налогоплательщиков. Используется матрица игры «Дилемма заключенного».
Начальное распределение: 9 уклоняющихся, 16 неуклоняющихся.
Число итераций: 8.
Конечное распределение: 25 уклоняющихся, 0 неуклоняющихся

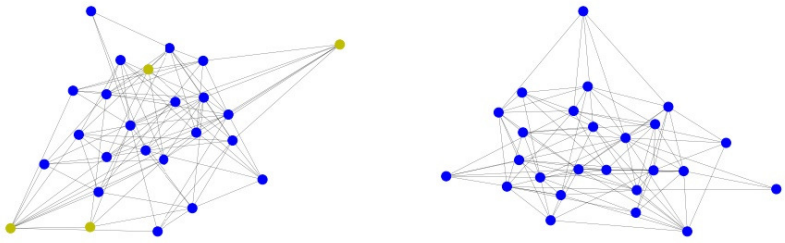

Рис. 30: Дана сильно связанная сеть из 25 налогоплательщиков. Используется матрица игры «Охота на оленя».
Начальное распределение: 21 уклоняющихся, 4 неуклоняющихся.
Число итераций: 3.
Конечное распределение: 25 уклоняющихся, 0 неуклоняющихся

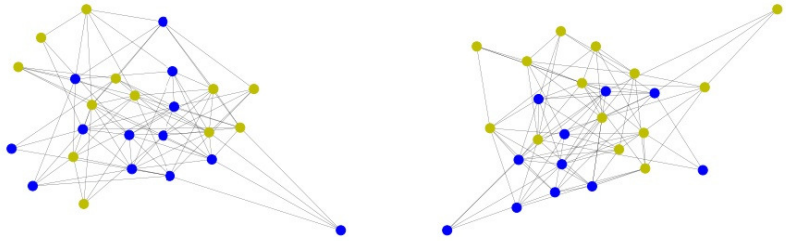

Рис. 31: Дана сильно связанная сеть из 25 налогоплательщиков. Используется матрица игры «Охота на оленя».
Начальное распределение: 13 уклоняющихся, 12 неуклоняющихся.
Число итераций: 6.
Конечное распределение: 11 уклоняющихся, 14 неуклоняющихся

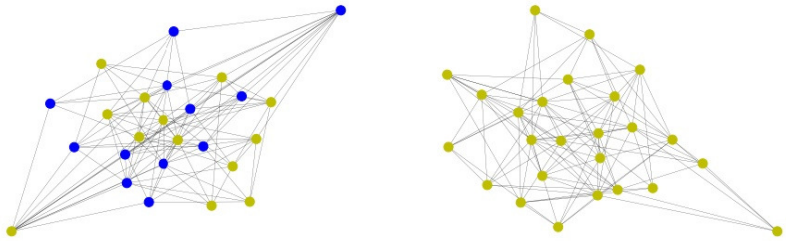

Рис. 32: Дана сильно связанная сеть из 25 налогоплательщиков. Используется матрица игры «Охота на оленя».
Начальное распределение: 12 уклоняющихся, 13 неуклоняющихся.
Число итераций: 7.
Конечное распределение: 0 уклоняющихся, 25 неуклоняющихся

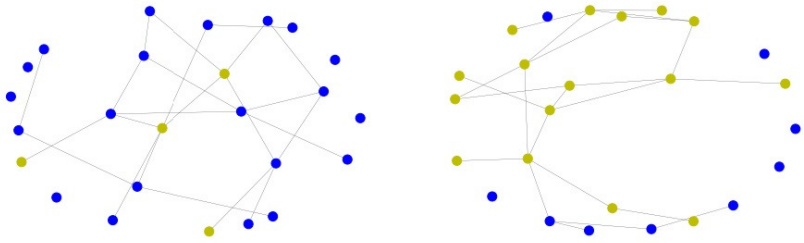

Рис. 33: Дана слабо связанная сеть из 25 налогоплательщиков. Используется матрица игры «Ястребы и голуби».
Начальное распределение: 21 уклоняющихся, 4 неуклоняющихся.
Число итераций: 6.
Конечное распределение: 9 уклоняющихся, 16 неуклоняющихся

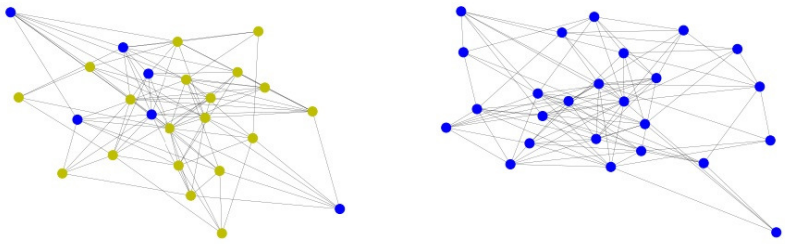

Рис. 34: Дана сильно связанная сеть из 25 налогоплательщиков. Используется матрица игры «Дилемма заключенного».
Начальное распределение: 6 уклоняющихся, 19 неуклоняющихся.
Число итераций: 6.
Конечное распределение: 25 уклоняющихся, 0 неуклоняющихся

Результаты

Рассмотренная модель позволила проанализировать ряд ситуаций взаимодействия налогоплательщиков, в которых помимо социальных связей и начальных данных учитывается возможность рациональной оценки окружающей ситуации экономическими агентами.

Было показано, что в условиях ограниченной информации о состоянии сети налогоплательщики не всегда приходят к желаемым результатам, и нередко ухудшают свое положение.

Влияние на модель такого типа со стороны фискальных органов может быть выражено в виде стимуляции к изменению стратегий стратегий некоторых из налогоплательщиков. С помощью созданного в рамках данной работы программного инструмента существует возможность оценить результаты такого взаимодействия.

Основной вопрос развития представленной модели — построение матрицы выигрышей A, основанной на параметрах реальной налоговой системы. Данный вопрос, однако, остается за рамками этой работы.

Еще одним направлением развития моделирования подобной системы можно назвать создание алгоритма, который позволит автоматически выбирать модель влияния, приводящую к результатам, необходимым для фискального органа.

Заключение

В данной работе были рассмотрены несколько моделей распространения информации в сети налогоплательщиков, основанные на различных трактовках процесса принятия налогоплательщиком решения об уплате или не уплате налогов.

Каждая из этих моделей учитывала сетевую структуру социальных связей между налогоплательщиками, анализ которой стал возможен совсем недавно благодаря развитию интернета и методов его изучения. Модели подобного типа имеют потенциал для практического применения в современном мире, что делает их изучение особенно актуальным.

В результате исследования процесса распространения информации в сети налогоплательщиков был получен ряд обобщенных результатов, а также создан программный инструмент, позволяющий прогнозировать естественное развитие ситуации в заданной сети.

Были рассмотрены две модели случайного распространения информации разной сложности и модель, учитывающая рациональность агентов.

Были описаны свойства распространения информации в сети налогоплательщиков, не подверженной влиянию извне. Для модели случайного распространения информации была проанализирована зависимость результатов от соотношения параметров модели. Для модели с рациональным анализом со стороны экономических агентов было оценено, как влияют на процесс различные матрицы выигрышей.

Также были предложены методы влияния на описанный процесс со стороны фискальных органов, приводящие к увеличению налоговых сборов. Было показано, что эти методы влияния заслуживают дальнейшего изучения, так как для всех рассмотренных моделей результаты их применения неоднозначны. Алгоритмизация выбора точных параметров подобного влияния на заданной сети — особенно интересная задача в рамках развития данного исследования.

Список литературы

1. Chandler P., Wilde L. Corruption in tax administration // Journal of Public Economics. 1992. №49. P. 333–349.

2. Vasin A., Vasina P. Tax Optimization under Tax Evasion: the role of penalty constraints // Moscow: EERC, 2002. P. 1–44.

3. Буре В. М., Кумачева С. Ш. Теоретико-игровая модель налоговых проверок с использованием статистической информации о налогоплательщиках // Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика. Информатика. Процессы управления. 2010. № 4. С. 16–24.

4. Сайт федеральной службы государственной статистики // URL: http://www.gks.ru/ (дата обращения: 20.12.2017).

5. Ниазашвили А. Г. Индивидуальные различия склонности к риску в разных социальных ситуациях развития личности: дис. канд. псих. наук: 19.00.01. М., 2007.

6. Kamada T., Kawai S. An algorithm for drawing general undirected graphs // Information Processing Letters. 1989. No 31. P. 7–15.

7. Koren Y. Drawing graphs by eigenvectors: theory and practice // Computers & Mathematics with Applications. 2005. No 49. P. 1867–1888.

8. Gubar E., Kumacheva S., Zhitkova E., Kurnosykh Z. Evolutionary behavior of taxpayers in the model of information dissemination // Constructive Nonsmooth Analysis and Related Topics (Dedicated to the Memory of V.F. Demyanov), CNSA 2017 — Proceedings. IEEE Conference Publications. 2017., 2017. P. 1–4.

9. Боровков А. А. Теория вероятностей: Учебное пособие для вузов. 2 изд. // М.: Наука, 1986.

10. Де Гроот М. Оптимальные статистические решения // М.: Мир, 1974.

11. DeGroot M. H. Reaching a consensus // Journal of the American Statistical Association. 1974. No 69. P. 118–121.

12. Губанов Д. А., Новиков Д. А., Чхартишвили А. Г. Социальные сети: модели информационного влияния, управления и противоборства // М.: Издательство физико-математической литературы, 2010.

13. Bure V. M., Parilina E. M., Sedakov A. A. Consensus in a social network with two principals // Automation and Remote Control. 2017. Vol. 78, No 8. P. 1489–1499.

14. Sandholm W. H. Population Games and Evolutionary Dynamics // Cambridge, Massachusetts: The MIT Press, 2010.

15. Колесин И. Д., Губар Е. А., Житкова Е. М. Стратегии управления в медико-социальных системах. Учебное пособие // СПб.: Издательство СПбГУ, 2014.

16. Skyrms B. The Stag Hunt and the Evolution of Social Structure // Cambridge: Cambridge University Press, 2003.

17. Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр. 2 изд. // СПб.: БХВ-Петербург, 2012.

18. Оуэн Г. Теория игр. 5 изд. // М.: ЛКИ, 2010.

19. Мулен Э. Теория игр с примерами из математической экономики // М.: Мир, 1985.

20. Васин А. А., Морозов В. В. Теория игр и модели математической экономики. Учебное пособие // М.: Макс-пресс, 2003.

21. Gubar E., Kumacheva S., Zhitkova E., Kurnosykh Z., Skovorodina T. Modelling of information spreading in the population of taxpayers: evolutionary approach // Contributions to Game Theory and Management. 2017. Vol. 10. P. 100–128.

22. Томилина Г. А. Моделирование сценариев распространения информации о проверках в сети налогоплательщиков // Процессы управления и устойчивость. 2018. (В печати)

23. Kumacheva S., Gubar E., Zhitkova E., Tomilina G. A model of the impact of information about tax audits on the risk statuses of taxpayers. // Устойчивость и колебания нелинейных систем управления (конференция Пятницкого). Материалы XIV Международной конференции. (В печати)

24. Kumacheva S., Gubar E., Zhitkova E., Tomilina G. Evolution of risk-statuses in one model of tax control // Springer, 2018. (In press)

Приложения

Приложение 1. Модель случайного распространения информации с двумя уровнями дохода

В этом приложении приведены выборочные участки кода, использующегося для проведения экспериментов с моделью случайного распространения информации с двумя уровнями дохода.

Классы

```
In [ ]: class Node:
            def __init__(self, value, isInformer = False):
                self.value = value
                self.isInformer = isInformer

In [ ]: class Informer:
            def __init__(self, index, value, trust = 1):
                self.index = index
                self.value = value
                self.trust = trust

In [ ]: class Graph:
            def __init__(self, n, nodes, relations, value, \
                         layout):
                self.n = n
                self.nodes = nodes
                self.relations = relations
                self.value = value
                self.layout = layout

            def add_informers(self, informers):
                for informer in informers:
                    self.nodes[informer.index] = Node(\
                                        informer.value, True)

                    neighbours = []
                    for i in range(len(self.nodes)):
                        if self.relations[informer.index][i]>0\
                            and not i == informer.index:
                            neighbours.append(i)
                    if len(neighbours) > 0:
                        value = (1 - informer.trust)\
                                /(len(neighbours))

                        self.relations[informer.index] = [\
                                value if i in neighbours else\
                                informer.trust \
                                if \
                                    i == informer.index \
                                else 0
                                    for i in range(\
                                            len(self.nodes))]
                    else:
                        self.relations[informer.index] = [
                                \1 if i == informer.index else\
```

```
                                0 for i in range(\
                                        len(self.nodes))]

        def update_values(self):
            updated_nodes = copy.deepcopy(self.nodes)

            for i in range(self.n):
                temp = 0
                for j in range(self.n):
                    temp += self.relations[i][j]\
                        *self.nodes[j].value
                updated_nodes[i].value = temp

            self.nodes = updated_nodes

        def print(self, relations_on = True):
            print('Number of nodes: {}'.format(self.n))
            print('Given value: {}'.format(self.value))
            print('Layout: {}'.format(self.layout))

            if relations_on == True:
                print('Relations: ')
                for row in self.relations:
                    print(row)

            print('nodes: ')
            for i, node in enumerate(self.nodes):
                print('"{}": value {}, isInformer {}'\
                      .format(i, node.value, \
                              node.isInformer))
```

```
In [ ]: class Result:
        def __init__(self, graph, start, end, iterations):
            self.graph = graph
            self.start = start
            self.end = end
            self.iterations = iterations

        def print(self):
            print('Initial state: {}'.format(self.start))
            print('Iterations: {}'.format(self.iterations))
            print('Final state: {}'.format(self.end))
```

Создание матрицы P

```
In [ ]: def generate_relations_random(n):
        result = np.zeros((n, n))
        rare = int(input(
            'Введите k, если 1/k - вероятность \
            образования связи между двумя участниками: '))

        for i in range(n):
            for j in range(n):
                temp = random.randrange(0, rare, 1)
                if temp == 0:
                    result[i][j] = random.random()

        for i in range(n):
```

```
            if not sum(result[i]) == 0:
                result[i] /= sum(result[i])
            else:
                result[i][i] = 1

        return result
```

In [ ]:
```
def generate_relations_ring(n):
    result = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if i == j + 1 or i == j - 1:
                result[i][j] = 0.5
            elif (i == 0 and (j == 1 or j == n - 1)) or\
                (i == n - 1 and (j == 0 or j == n - 2)):
                result[i][j] = 0.5

    return result
```

In [ ]:
```
def generate_simple_grid(n):
    rows = []
    m = mat.floor(mat.sqrt(n))
    s = m ** 2
    for i in range(s):
        k = i + 1
        row = [0 for _ in range(s)]
        if not k % m == 0:
            row[i + 1] = 1
        if not i - 1 < 0 and not (k - 1) % m == 0:
            row[i - 1] = 1
        if not k + m > s:
            row[i + m] = 1
        if not k - m <= 0:
            row[i - m] = 1
        rows.append(row)
    return np.array(rows)

def generate_relations_grid(n):
    grid_matrix = generate_simple_grid(n)
    n = len(grid_matrix)
    result = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if grid_matrix[i][j]:
                result[i][j] = random.random()

    for i in range(n):
        if not sum(result[i]) == 0:
            result[i] /= sum(result[i])
        else:
            result[i][i] = 1

    return result
```

In [ ]:
```
def generate_relations_sparse(n):
    matrix = generate_relations_random(n)
    k = int(input(\
        'Введите число изолированных игроков \
```

```
                                            (< n/2): '))
        random_indices = random.sample(range(0, n), k)
        print('Изолированные игроки: {}'\
              .format(random_indices))
        for ind in random_indices:
            weights = np.random\
                      .dirichlet(np.ones(2), size = 1)
            neighbour_index = -1
            while neighbour_index < 0 or \
                neighbour_index in random_indices:
                neighbour_index = random.randrange(0, n, 1)
            matrix[ind] = [weights[0][0] \
                           if \
                               i == ind \
                           else weights[0][1] \
                           if \
                               i == neighbour_index \
                           else 0 for i in range(n)]
            for j in range(n):
                if not j == ind and \
                        not j == neighbour_index:
                    matrix[j][ind] = 0
        for i in range(n):
            if not sum(matrix[i]) == 0:
                matrix[i] /= sum(matrix[i])
            else:
                matrix[i][i] = 1
        return matrix
```

Внедрение информационных центров

```
In [ ]: def get_information_centers_in_random_places(\
                                        numberOfNodes):
        result = []
        k = int(input('Введите число \
                    информационных центров: '))
        random_indices = random.sample(range(\
                             0, numberOfNodes), k)
        for i in range(k):
            value = float(input(\
                'Введите значение вероятности \
                (f) для центра: '))
            alpha = float(input(\
                'Введите уверенность в информации \
                (alpha) для центра: '))
            result.append(Informer(random_indices[i],\
                                   value, alpha))
        return result
```

Отображение результатов

```
In [ ]: def create_graph(relations):
        G = nx.DiGraph()
```

```
            nodes = range(len(relations))
            G.add_nodes_from(nodes)

            for i in nodes:
                for j in nodes:
                    if relations[i][j] > 0:
                        G.add_edge(i, j, \
                            weight = relations[i][j])

            return G

        def set_edges(graph, G, \
                      only_to_informers = False):
            if not only_to_informers:
                return G.edges()

            result = []

            for edge in G.edges():
                if graph.nodes[edge[1]].isInformer:
                    result.append(edge)

            return result
```

In [ ]:
```
        def set_sizes(nodes):
            return [400 if node.isInformer else \
                    100 for node in nodes]

        def set_colors(nodes, value):
            colors = []
            counter = {"evaders": 0, "not evaders": 0}

            for node in nodes:
                if value - node.value > 10e-3:
                    counter["evaders"] += 1
                    colors.append('b')
                else:
                    counter["not evaders"] += 1
                    colors.append('y')

            return [colors, counter]

        def choose_layout():
            print('Layout: ')
            layouts = ['spectral', 'spring', \
                       'shell', 'circular']
            for i, layout in enumerate(layouts):
                print('{}) {}'.format(i + 1, layout))
            choice = int(input('Введите цифру, \
                    соответсвующую вашему выбору: '))
            return layouts[choice - 1]
```

In [ ]:
```
        def count_and_draw(graph, only_to_informers, drawings_on):
            colors, counter = set_colors(graph.nodes, graph.value)

            layout = graph.layout
            if drawings_on:
                G = create_graph(graph.relations)

                if layout == 'spectral':
                    l = nx.spectral_layout(G)
```

```
            elif layout == 'spring':
                l = nx.spring_layout(G)
            elif layout == 'shell':
                l = nx.shell_layout(G)
            else:
                l = nx.circular_layout(G)

            nx.draw_networkx(G, pos = l, arrows = False, \
                             edgelist = set_edges(\
                                 graph, G, only_to_informers), \
                             node_size = set_sizes(graph.nodes), \
                             node_color = colors, \
                             width = 0.1, edge_color = 'k', \
                             vmin = 0.0, vmax = 2.0, \
                             with_labels = False)

            plt.show()

        return counter
```

Эксперимент

```
In [ ]: def get_value():
            return float(input('Введите пороговое \
                    значение вероятности: '))

        def get_nodes(n, value):
            print('Информация в сети в начальный момент \
                    будет распределена нормально \
                    со следующими параметрами')
            print('Математическое ожидание: {}'.format(value))
            sd = float(input('Введите стандартное отклонение: '))
            return generate_nodes(n, value, sd)

        def get_graph_for_experiment():
            value = get_value()
            n = int(input('Введите число агентов: '))
            relations = get_relations(n)
            n = len(relations)
            nodes = get_nodes(n, value)
            layout = choose_layout()
            g = Graph(len(nodes), nodes, relations, \
                      value, layout)
            g.add_informers(\
                get_information_centers_in_random_places(\
                                 len(nodes)))
            return g

        def stabilize(graph, eps, drawings_on, info_on):
            start = count_and_draw(graph, \
                        only_to_informers = False,\
                        drawings_on = drawings_on)

            previous_nodes = copy.deepcopy(graph.nodes)
            k = 0
```

```
        while True:
            graph.update_values()
            k += 1
            dist = distance(previous_nodes, graph.nodes)

            if info_on:
                print("\nGraph on {} iteration".format(k))
                graph.print(relations_on = False)
                print("\nDifference on {} iteration = {}"\
                      .format(k, dist))

            if dist < eps:
                end = count_and_draw(graph, \
                                     only_to_informers = False, \
                                     drawings_on = drawings_on)
                if info_on:
                    print("\n\nIterations: {}".format(k))
                return Result(graph, start, end, k)

            previous_nodes = copy.deepcopy(graph.nodes)
```

Приложение 2. Модель случайного распространения информации с тремя уровнями дохода

В этом приложении приведены выборочные участки кода, использующегося для проведения экспериментов с моделью случайного распространения информации с тремя уровнями дохода.

Классы

```
In [ ]: class Risk(Enum):
            negative = 1
            neutural = 2
            positive = 3

        class Income(Enum):
            low = 1
            medium = 2
            high = 3

        class Node:
            def __init__(self, income, p_income, risk, \
                         l_value, m_value, isInformer = False):
                self.income = income
                self.p_income = p_income
                self.risk = risk
                self.l_value = l_value
                self.m_value = m_value
                self.isInformer = isInformer

            def check_p_income(self, threshold):
                if self.risk == Risk.negative or \
                self.income == Income.low:
                    return self
                if self.income == Income.high:
                    if self.p_income == Income.low:
```

```
                    if self.l_value >= threshold:
                        if self.m_value >= threshold:
                            self.p_income = Income.high
                        else:
                            self.p_income = Income.medium
                elif self.p_income == Income.medium:
                    if self.l_value < threshold:
                        self.p_income = Income.low
                    elif self.m_value >= threshold:
                        self.p_income = Income.high
                elif self.p_income == Income.high:
                    if self.l_value < threshold:
                        self.p_income = Income.low
                    elif self.m_value < threshold:
                        self.p_income = Income.medium
                return self
            else:
                if self.p_income == Income.low:
                    if self.l_value >= threshold:
                        self.p_income = Income.medium
                elif self.p_income == Income.medium:
                    if self.l_value < threshold:
                        self.p_income = Income.low
                return self

        def print(self):
            return 'income {}, p_income {}, \
                    risk {}, l_value {}, m_value {}, \
                    isInformer {}'.format(\
                    self.income, self.p_income,\
                    self.risk, self.l_value, \
                    self.m_value, self.isInformer)

    class Informer:
        def __init__(self, index, l_value, m_value, \
                     canBeNegative = True, trust = 1):
            self.index = index
            self.l_value = l_value
            self.m_value = m_value
            self.trust = trust
            self.canBeNegative = canBeNegative

    class Graph:
        def __init__(self, n, nodes, relations, \
                     threshold, layout):
            self.n = n
            self.nodes = nodes
            self.relations = relations
            self.threshold = threshold
            self.layout = layout

        def make_informer(self, node, new_l_value, \
                          new_m_value):
            node.isInformer = True
            node.l_value = new_l_value
            node.m_value = new_m_value
            return node.check_p_income(self.threshold)

        def add_informers(self, informers):
            for informer in informers:
                self.nodes[informer.index] = \
                    self.make_informer(\
                        self.nodes[informer.index], \
```

```
                        informer.l_value, informer.m_value)
                neighbours = []
                for i in range(len(self.nodes)):
                    if self.relations[informer.index][i] > 0 \
                        and not i == informer.index:
                        neighbours.append(i)
                if len(neighbours) > 0:
                    value = (1 - informer.trust)\
                            /(len(neighbours))

                    self.relations[informer.index] = \
                        [value if i in neighbours else \
                        informer.trust if i == informer.index \
                        else 0 \
                            for i in range(len(self.nodes))]
                else:
                    self.relations[informer.index] = \
                        [1 if i == informer.index else \
                        0 for i in range(len(self.nodes))]

        def update_values(self):
            updated_nodes = copy.deepcopy(self.nodes)
            for i in range(self.n):
                temp_l = 0
                temp_m = 0
                for j in range(self.n):
                    temp_l += self.relations[i][j]*\
                                self.nodes[j].l_value
                    temp_m += self.relations[i][j]*\
                                self.nodes[j].m_value
                updated_nodes[i].l_value = temp_l
                updated_nodes[i].m_value = temp_m
            for node in updated_nodes:
                node.check_p_income(self.threshold)
            self.nodes = updated_nodes

        def print(self, relations_on = True):
            data = 'Number of nodes: {}\n'.format(self.n)
            data += 'Threshold: {}\n'.format(self.threshold)
            data += 'Layout: {}\n'.format(self.layout)
            if relations_on == True:
                data += 'Relations: \n'
                for row in self.relations:
                    data += str(row) + '\n'
            data += 'nodes: \n'
            for i, node in enumerate(self.nodes):
                data += '{}) {}\n'.format(i, node.print())
            print(data)
            return data

    class Subgraph:
        def __init__(self, shape, nodes, g, colors, sizes):
            self.shape = shape
            self.nodes = nodes
            self.g = g
            self.colors = colors
            self.sizes = sizes
```

Создание матрицы P

Аналогично созданию матрицы P для модели с двумя уровнями дохода налогоплательщиков.

Создание начального распределения

```
In [ ]: def calculate_numbers_by_risk(n):
            positive = 0.18
            neutural = 0.65
            negative = 0.17
            n_positive = round(n * positive)
            n_neutural = round(n * neutural)
            n_negative = round(n * negative)
            sums = n_positive + n_neutural + n_negative
            if sums - n > 0:
                n_neutural -= sums - n
            elif sums - n < 0:
                n_neutural += n - sums
            return {"positive": n_positive, \
                    "neutural": n_neutural, \
                    "negative": n_negative}

In [ ]: def generate_nodes_for_income(n, income):
            result = []
            risk_numbers = calculate_numbers_by_risk(n)
            for _ in range(risk_numbers["negative"]):
                result.append(Node(income, \
                    income, Risk.negative, 1, 1))
            for _ in range(risk_numbers["neutural"]):
                if income == income.low:
                    result.append(Node(income, \
                income, Risk.neutural, 0.5, 0.5))
                else:
                    result.append(Node(income, \
                        Income.low, Risk.neutural, 0.5, 0.5))
            for _ in range(risk_numbers["positive"]):
                if income == income.low:
                    result.append(Node(income, \
                        income, Risk.positive, 0, 0))
                else:
                    result.append(Node(income, \
                        Income.low, Risk.positive, 0, 0))
            random.shuffle(result)
            return result

In [ ]: def generate_nodes(income_numbers):
            result = []
            result.extend(generate_nodes_for_income(\
                    income_numbers["low"], Income.low))
            result.extend(generate_nodes_for_income(\
                    income_numbers["medium"], Income.medium))
            result.extend(generate_nodes_for_income(
                    income_numbers["high"], Income.high))
            random.shuffle(result)
            return result
```

Внедрение информационных центров

```
In [ ]: def get_information_centers_in_random_places(\
                                        numberOfNodes):
            result = []
            k = int(input('Введите число \
                информационных центров: '))
            random_indices = random.sample(\
                        range(0, numberOfNodes), k)
            for i in range(k):
                l_value = float(input(\
                    'Введите значение \
                    вероятности (L) для центра: '))
                m_value = float(input(\
                    'Введите значение \
                    вероятности (M) для центра: '))
                alpha = float(input(\
                    'Введите уверенность \
                    в информации (alpha) для центра: '))
                result.append(Informer(\
                    random_indices[i], l_value, \
                                m_value, alpha))
            return result
```

Отображение результатов

```
In [ ]: def set_shapes(nodes):
            shapes = []
            counter = {"L": 0, "M": 0, "H": 0, \
                       "not evaders": 0, "L(H)": 0,\
                       "L(M)": 0, "M(H)": 0}
            for node in nodes:
                if node.income == Income.low:
                    counter["L"] += 1
                    counter["not evaders"] += 1
                    shapes.append('o')
                elif node.income == Income.medium:
                    counter["M"] += 1
                    if node.p_income == Income.low:
                        counter["L(M)"] += 1
                        shapes.append("D")
                    else:
                        counter["not evaders"] += 1
                        shapes.append("s")
                else:
                    counter["H"] +=1
                    if node.p_income == Income.low:
                        counter["L(H)"] += 1
                        shapes.append("v")
                    elif node.p_income == Income.medium:
                        counter["M(H)"] += 1
                        shapes.append("<")
                    else:
                        counter["not evaders"] += 1
                        shapes.append("^")
            print(shapes)
```

```
            print(counter)
            return (shapes, counter)
```

```
In [ ]: def set_colors_based_on_values(nodes):
            sets = {"Neg": [0, 0.25], \
                    "Neut": [0.25, 0.75], \
                    "Pos": [0.75, 1]}
            colors = []
            for node in nodes:
                if node.risk == Risk.negative:
                    colors.append('g')
                else:
                    value = (node.l_value + node.m_value) / 2
                    if value >= sets["Neg"][0] and \
                    value < sets["Neg"][1]:
                        colors.append('r')
                    elif value >= sets["Neut"][0] and \
                    value < sets["Neut"][1]:
                        colors.append('b')
                    else:
                        colors.append('g')
            return colors
```

```
In [ ]: def generate_subgraph_based_on_shape(graph, G, \
                                                shape, shapes):
            indices = [i for i in range(len(graph.nodes)) \
                       if shapes[i] == shape]
            nodes = [graph.nodes[i] for i in range(\
                                    len(graph.nodes))\
                     if i in indices]
            g = G.subgraph(indices)
            colors = set_colors_based_on_values(nodes)
            sizes = set_sizes(nodes)
            return Subgraph(shape, nodes, g, colors, sizes)
```

```
In [ ]: def draw_different_shapes(graph, G, shapes, \
                                 layout, only_to_informers, \
                                 images_to_file, filename):
            available_shapes = ['o', 'D', 's', 'v', '<', '^']
            subgraphs = []
            for shape in available_shapes:
                subgraphs.append(\
                    generate_subgraph_based_on_shape(\
                            graph, G, shape, shapes))
            fig = plt.figure()
            ax = plt.Axes(fig, [0., 0., 1., 1.])
            ax.set_axis_off()
            fig.add_axes(ax)
            for subgraph in subgraphs:
                nx.draw_networkx(\
                    subgraph.g, pos = layout, \
                    arrows = False, \
                    edgelist = set_edges(\
                        graph, G, only_to_informers), \
                    node_size = subgraph.sizes, \
```

```
                    node_color = subgraph.colors, \
                    node_shape = subgraph.shape, \
                    width = 0.05, edge_color = 'k', \
                    vmin = 0.0, vmax = 2.0, \
                                with_labels = False)
            plt.show()
            if images_to_file:
                fig.savefig(filename, bbox_inches='tight')
```

```
In [ ]: def count_and_draw(graph, only_to_informers, \
                           drawings_on, images_to_file, \
                           filename):
            shapes, counter = set_shapes(graph.nodes)
            layout = graph.layout
            if drawings_on:
                G = create_graph(graph.relations)
                if layout == 'spectral':
                    l = nx.spectral_layout(G)
                elif layout == 'spring':
                    l = nx.spring_layout(G)
                elif layout == 'shell':
                    l = nx.shell_layout(G)
                else:
                    l = nx.circular_layout(G)
                draw_different_shapes(graph, G, shapes, \
                    l, only_to_informers, images_to_file, \
                                        filename)
            return counter
```

Эксперимент

```
In [ ]: def distance(previous_nodes, current_nodes):
            distance_l = 0
            distance_m = 0
            for i in range(len(previous_nodes)):
                distance_l += (current_nodes[i].l_value \
                               - previous_nodes[i].l_value) ** 2
                distance_m += (current_nodes[i].m_value \
                               - previous_nodes[i].m_value) ** 2
            return mat.sqrt(distance_l + distance_m)
```

```
In [ ]: def calculate_numbers_by_income(n, low, medium, high):
            n_low = round(n * low)
            n_medium = round(n * medium)
            n_high = round(n * high)
            sums = n_low + n_medium + n_high
            if sums - n > 0:
                n_low -= sums - n
            elif sums - n < 0:
                n_low += n - sums
            return {"low": n_low, "medium": n_medium, \
                    "high": n_high}
```

```
In [ ]: def get_nodes(n):
            low_income = 0.231
```

```
            medium_income = 0.56
            high_income = 0.209
            return generate_nodes(\
                    calculate_numbers_by_income(n, \
                    low_income, medium_income, high_income))
```

```
In [ ]: def get_graph_for_experiment():
            threshold = float(input(\
                'Введите пороговое значение вероятности: '))
            n = int(input('Введите число агентов: '))
            relations = get_relations(n)
            n = len(relations)
            nodes = get_nodes(n)
            layout = choose_layout()
            g = Graph(len(nodes), nodes, relations, \
                      threshold, layout)
            g.add_informers(\
                get_information_centers_in_random_places(\
                                              len(nodes)))
            return g
```

```
In [ ]: def stabilize(graph, eps, drawings_on, info_on, \
                      images_to_file, filename = '.'):
            start = count_and_draw(graph, False, \
                        drawings_on, images_to_file, \
                                   filename + '0.png')
            previous_nodes = copy.deepcopy(graph.nodes)
            k = 0
            while True:
                graph.update_values()
                k += 1
                dist = distance(previous_nodes, graph.nodes)
                if info_on:
                    print("\nGraph on {} iteration".format(k))
                    graph.print(relations_on = False)
                    print("\nDifference on {} iteration = {}".\
                          format(k, dist))
                if dist < eps:
                    end = count_and_draw(graph, False, \
                                         drawings_on, \
                                         images_to_file, \
                                         filename + '1.png')
                    print("End {}".format(end))
                    if info_on:
                        print("\n\nIterations: {}".format(k))
                    return Result(graph, start, end, k)
                previous_nodes = copy.deepcopy(graph.nodes)
```

```
In [ ]: def run_experiment(eps = 10e-3):
            g = get_graph_for_experiment()
            g.print()
            result = stabilize(g, eps, \
                               drawings_on = True, \
                               info_on = False, \
```

```
                                images_to_file = False)
        result.print()
        return result
```

Приложение 3. Модель пропорциональной имитации

В этом приложении приведены выборочные участки кода, использующегося для проведения экспериментов с моделью пропорциональной имитации

Матрицы выигрышей

```
In [ ]: prisoners_dilemm = np.array([[4, -1],
                                     [5, 0]])
        howks_and_doves = np.array([[-1, 4],
                                    [0, 2]])
        stag_hunt = np.array([[3, 0],
                              [1, 1]])

        names = ("Дилемма заключенного", \
                 "Ястребы - голуби", \
                 "Охота на оленя")
        games = (prisoners_dilemm, howks_and_doves, stag_hunt)
```

```
In [ ]: def get_matrix():
            rows = []
            for i in range(2):
                row = input('Введите строку или \
                    нажмите Enter для прекращения ввода: ')
                if row:
                    rows.append(row)
                else:
                    break
            matrix = '\n'.join(rows)
            return np.array([[int(j) \
                                  for j in i.split(' ')] \
                             for i in matrix.splitlines()])

        def get_choice(names, games):
            choice = 1;
            for name in names:
                print("{}) {}".format(choice, name))
                choice += 1
            print('{}) Ввести матрицу выигрышей \
                    самостоятельно'.format(choice))
            choice += 1
            choice = int(input('Введите цифру, \
                    соответствующую вашему выбору: '))
            return choice

        def get_payoff_matrix():
            choice = get_choice(names, games)
            if choice == len(names) + 1:
                game = get_matrix()
            else:
```

```
            game = games[choice - 1]
        return game
```

Создание матрицы P

```
In [ ]: def create_grid_graph_auto(n):
            rows = []
            m = mat.floor(mat.sqrt(n))
            s = m ** 2
            for i in range(s):
                k = i + 1
                row = [0 for _ in range(s)]
                if not k % m == 0:
                    row[i + 1] = 1
                if not i - 1 < 0 and not (k - 1) % m == 0:
                    row[i - 1] = 1
                if not k + m > s:
                    row[i + m] = 1
                if not k - m <= 0:
                    row[i - m] = 1
                rows.append(row)
            return np.array(rows)

        def create_grid_graph():
            n = int(input('Введите число агентов: '))
            return create_grid_graph_auto(n)

        func_to_generate_graphs.append(create_grid_graph)
        func_to_generate_graph_descs.append(\
                            'Сгенерировать граф-решетку')
```

```
In [ ]: def create_rand_graph_auto(n, k):
            adjacency_matrix = np.zeros((n, n))
            for i in range(n):
                for j in range(i + 1):
                    if not i == j:
                        rand = random.randrange(0, k, 1)
                        if rand == 0:
                            adjacency_matrix[i][j] = 1
                            adjacency_matrix[j][i] = 1
            return adjacency_matrix

        def create_rand_graph():
            n = int(input('Введите число агентов: '))
            k = int(input('Введите k, если 1/k - вероятность \
                    образования связи между двумя агентами: '))
            return create_rand_graph_auto(n, k)

        func_to_generate_graphs.append(create_rand_graph)
        func_to_generate_graph_descs.append(\
                'Сгенерировать граф со случайными рёбрами \
                (алгоритм основан на генерации случайных чисел)')
```

```
In [ ]: def create_ring_graph():
            n = int(input('Введите число агентов: '))
            adjacency_matrix = np.zeros((n, n))
```

```
            for i in range(n):
                for j in range(n):
                    if i == j + 1 or i == j - 1:
                        adjacency_matrix[i][j] = 1
                    elif (i == 0 and (j == 1 or \
                                      j == n - 1)) or \
                            (i == n - 1 and \
                             (j == 0 or j == n - 2)):
                        adjacency_matrix[i][j] = 1
            return adjacency_matrix

        func_to_generate_graphs.append(create_ring_graph)
        func_to_generate_graph_descs.append(\
                    'Сгенерировать граф - кольцо из агентов')
```

Создание начального распределения

```
In [ ]: def form_random_strategies(n):
            strategies = []
            for i in range(n):
                strategies.append(random.choice([0, 1]))
            return strategies

        def form_random_strategies_with_proportions(n, \
                                                    paying_part):
            strategies = [1 for _ in range(n)]
            m = round(n * paying_part)
            indices = random.sample(range(n), m)
            for i in indices:
                strategies[i] = 0
            return strategies


In [ ]: def get_strategies(n):
            choice = int(input('Для формирования \
                    случайных стратегий введите 1, \
                    \nдля формирования стратегий \
                    с соблюдением пропорций введите 2: '))
            if choice == 1:
                return form_random_strategies(n)
            proportion = float(input('Введите долю \
                    платящих налоги в начальный момент \
                    (от 0 до 1, через .): '))
            return form_random_strategies_with_proportions(\
                                                n, proportion)
```

Отображение результатов

```
In [ ]: def get_colors_and_count(strategies):
            colors = []
            counter = {'evading': 0, 'not evading': 0}
            for s in strategies:
                if s == 0:
                    colors.append('y')
```

```
                counter['not evading'] += 1
            else:
                colors.append('b')
                counter['evading'] += 1
        return counter, colors
```

```
In [ ]: def draw(G, strategies, save_to_file, filename):
        counter, colors = get_colors_and_count(strategies)
        fig = plt.figure()
        ax = plt.Axes(fig, [0., 0., 1., 1.])
        ax.set_axis_off()
        fig.add_axes(ax)
        nx.draw_networkx(G, pos = nx.spectral_layout(G), \
                         node_size = 100,\
                         node_color = colors, \
                         width = 0.2, edge_color = 'k', \
                         vmin = 0.0, vmax = 2.0, \
                         with_labels = True)
        plt.show()
        if save_to_file:
            fig.savefig(filename, bbox_inches='tight')
        return counter
```

Эксперимент

```
In [ ]: def get_neighbours_indices(adjacency_matrix, i, n):
        indices = []
        for j in range(n):
            if adjacency_matrix[i][j] == 1 and \
                    not j == i:
                indices.append(j)
        return indices

    def get_n_neighbours(adjacency_matrix, i, n):
        return len(get_neighbours_indices(\
                            adjacency_matrix, \
                            i, n))
```

```
In [ ]: def get_lambda_ind(adjacency_matrix, i, n):
        return get_n_neighbours(adjacency_matrix, \
                                i, n)

    def get_lambda_mean(adjacency_matrix, n):
        lambdas = []
        for i in range(n):
            lambdas.append(get_lambda_ind(\
                            adjacency_matrix, \
                            i, n))
        return np.mean(lambdas)

    def init_lambdas(adjacency_matrix, n):
        choice = int(input('Введите 1 если вы хотите \
                использовать среднее значение лямбда \
```

```
                и 2 - индивидуальные лямбда'))
            init_lambdas_auto(adjacency_matrix, n, choice)
```

```
In [ ]: def get_omega(i, n_i, omega_choice):
            if omega_choice == 1 or n_i == 0:
                return 1
            return 1 / n_i
```

```
In [ ]: def get_current_payoff(i, adjacency_matrix, \
                               strategies, \
                               payoff_matrix, omega_choice, \
                               n):
            neighbours = get_neighbours_indices(\
                                    adjacency_matrix, \
                                    i, n)
            omega = get_omega(i, len(neighbours), \
                              omega_choice)
            payoff = 0
            for j in neighbours:
                payoff += \
                    payoff_matrix[strategies[i]][strategies[j]]
            return omega * payoff

        def get_current_payoffs(adjacency_matrix, strategies, \
                                payoff_matrix, omega_choice, n):
            payoffs = []
            for i in range(n):
                payoffs.append(get_current_payoff(i, \
                        adjacency_matrix, strategies, \
                        payoff_matrix, omega_choice, n))
            return payoffs
```

```
In [ ]: def get_p(z):
            return max(0, min(1, z))

        def get_proportion(lyambda, n_i, payoff_i, payoff_j):
            return get_p(lyambda * (payoff_j - payoff_i) / n_i)
```

```
In [ ]: def get_next_strategy(i, adjacency_matrix, strategies, \
                              current_payoffs, lyambda, n):
            neighbours = get_neighbours_indices(\
                                    adjacency_matrix, i, n)
            if len(neighbours) == 0:
                return 0, strategies[i]
            random_neighbour = random.choice(neighbours)
            proportion = get_proportion(\
                           lyambda, len(neighbours), \
                           current_payoffs[i], \
                           current_payoffs[random_neighbour])
            if proportion == 0:
                return proportion, strategies[i]

            rand = random.random()
            if rand <= proportion:
```

```
                return proportion, strategies[random_neighbour]
            return proportion, strategies[i]

        def get_next_strategies(\
                        adjacency_matrix, strategies, \
                        current_payoffs, lambdas, n):
            proportions = []
            next_strategies = []
            for i in range(n):
                proportion, next_strategy =
                    get_next_strategy(i, \
                        adjacency_matrix, strategies, \
                        current_payoffs, lambdas[i], n)
                proportions.append(proportion)
                next_strategies.append(next_strategy)
            print('p(x_i(t+1) = x_j(t)): {}'\
                  .format(proportions))
            return next_strategies
```

In [ ]:
```
def compare(strategies, next_strategies):
    count = 0
    for i in range(len(strategies)):
        count += (strategies[i] - next_strategies[i]) ** 2
    return mat.sqrt(count)
```

In [ ]:
```
def iterate(G, adjacency_matrix, \
            strategies, \
            payoff_matrix, lambdas, \
            omega_choice, \
            n, save_to_file, \
            directory = '.', filename = '.'):
    num_iter = 0
    initial_counter = draw(G, strategies, \
                           save_to_file, \
                           filename + '_0.png')
    data =  'Начальное распределение: \
            {} уклоняющихся, \
            {} не уклоняющихся\n'.format(\
                    initial_counter['evading'], \
                    initial_counter['not evading'])
    print(data)
    while True:
        previous_strategies = copy.deepcopy(strategies)
        num_iter += 1
        current_payoffs = get_current_payoffs(\
                            adjacency_matrix, \
                            strategies, \
                            payoff_matrix, \
                            omega_choice, n)
        data += 'Текущие выигрыши: {}\n'\
                    .format(current_payoffs)
        strategies = get_next_strategies(\
                        adjacency_matrix, \
```

```
                                strategies, current_payoffs, \
                                lambdas, n)
            if compare(previous_strategies, strategies) < eps:
                break
        final_counter = draw(G, strategies, save_to_file, \
                             filename + '_1.png')
        data += 'Конечное распределение: {} уклоняющихся, \
            {} не уклоняющихся\n'.format(\
                                final_counter['evading'], \
                            final_counter['not evading'])
        data += 'Число итераций: {}\n'.format(num_iter)
        print(data)
        if save_to_file:
            print_to_data_file(directory, data)
```

In [ ]:
```
def run(eps):
    payoff_matrix = get_payoff_matrix()
    adjacency_matrix = get_adjacency_matrix()
    G = create_graph(adjacency_matrix)
    n = len(adjacency_matrix)
    strategies = get_strategies(n)
    lambdas = init_lambdas(\
                adjacency_matrix, n)
    omega_choice = int(\
            input(\
            'Введите 1 для использования \
            cumulative payoff, 2 - averaged payoff:'))
    iterate(G, adjacency_matrix, \
            strategies, payoff_matrix, \
            lambdas, omega_choice, n, \
            save_to_file = False)
```