САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПЕТРОВ Антон Борисович

Выпускная квалификационная работа

Моделирование стабильных социальных сетей

Уровень образования: бакалавриат

Направление: 02.03.02 «Фундаментальная информатика и информационные технологии

ООП: Программирование и информационные технологии

Научный руководитель:

Парилина Елена Михайловна,

доктор физико-математических наук,

профессор кафедры математической

теории игр и статистических решений

Санкт-Петербург

2021

Содержание

ВВЕДЕНИЕ

Сети играют фундаментальную роль в самых разных социальных и экономических взаимодействиях. С помощью моделей, построенных на использовании сетей, можно описать, например, взаимодействия на рынке различных участников, модели поведения людей в обществе. Так, например, можно проанализировать построение сетей в системе с агентами разных типов. Каждый тип агента может обозначать расу, например, или национальность. Агентам одного типа легче взаимодействовать друг с другом, но они получают выгоду от взаимодействия с агентами другого типа. Так моделируется формирование дружеских отношений в американских школах, в которых учатся ученики разных рас [1].

Игроки в социальных сетях формируют взаимоотношения с другими игроками и получают из-за этих взаимоотношений выгоду.

1. Теория игр и социальные сети

Социальные сети рассматривают в рамках теоретико-игрового подхода. Некоторые определения, например, равновесие по Нэшу, имеют смысл и для случаев социальных сетей.

Конечное множество $N = \{1, ..., n\}$ называют *множеством игроков*, а каждый элемент этого множества - *игроком*. Игроками могут быть как отдельные люди, так и организации, например, фирмы или даже страны.

Игроки могут выбирать, с кем им следует строить отношения. Значение отношений меняется в зависимости от модели.

Сетевые отношения между игроками представляются графом, в котором вершины представляют игроков, а ребра - их попарные отношения. *Граф взаимоотношений (сеть)* обозначим как g. Полная сеть g^N - множество всех подмножеств Nстепени 2.

В зависимости от контекста, взаимоотношения могут принимать разные формы. Чаще всего рассматривают самую простую форму взаимоотношений - когда граф g не ориентированный. Однако g может быть и ориентированным, когда отношение первого игрока ко второму не означает, что второй имеет отношение к первому. Подобные взаимоотношения встречаются, например, в ссылках на веб-сайтах или в списках литературы. Далее будем рассматривать только те случаи, когда g не ориентирован, так как такие взаимоотношения наиболее распространены. [2]

Граф g также может быть взвешенным. Как пример взаимоотношений, когда каждому отношению поставлено какое-то значение, можно привести отношения дружбы. Каким-то друзьям игрок может доверять больше, чем другим. В литературе обычно рассматриваются не взвешенные графы отношений. В данной работе будут рассмотрены только такие случаи, когда g не взвешен.

Ребро между игроками i и j обозначим как ij. Для любых двух игроков i и j если $\{i, j\} \in g$, то значит, что в сети g игроки состоят в отношении.

Отношения игроки выбирают сами - это значит, что для того, чтобы ребро существовало в графе, нужно, чтобы между игроками существовало *взаимное согласие* на существование такого отношения.

Для сетей в литературе приняты следующие обозначения. Как $g + ij$ обозначим сеть, которая получена после добавления ребра ij в сеть g. Аналогично обозначим $g - ij$ как сеть, которая получена удалением ребра ij из сети g.

Обозначим $g|_S$ как сеть, состоящую только из отношений игроков из $S \subset N$:

$$g|_S = \{ij \mid ij \in g \text{ и } i, j \in S\}.$$

Введем $N(g)$ как множество игроков, у которых есть хотя бы одно отношение в g:

$$N(g) : \{i \mid \exists j \text{ такой, что } ij \in g\}.$$

Ценой сети назовем

$$\upsilon : \{g \mid g \subset g^N\} \to \Re,$$

где $\upsilon(g)$ обозначает полную стоимость сети. Множество всех функций υ обозначим V. Цену сети удобно представить как

$$\upsilon = \sum u_i,$$

$$u_i : \{g \mid g \subset g^N\} \to \Re,$$

где u_i - *функция цены игрока* i при состоянии системы g.

Правило распределения

$$Y: \{g \mid g \subset g^N\} \times V \to \Re^N$$

описывает, какой выигрыш получает каждый игрок при определенном состоянии сети.

Путь в сети g между игроками i и j - это последовательность игроков $i_1,\ldots, i_K$ такая, что $i_k i_{k+1} \in g$ для каждого $k \in \{1,\ldots, K-1\}$, где $i_1 = i$ и $i_k = j$.

Компонент сети g - непустая подсеть $g' \in g$ такая, что

- Если $i \in N(g')$ и $j \in N(g')$, $i \neq j$, тогда существует путь в g' между i и j, и
- Если $i \in N(g')$ и $ij \in g$, то $ij \in g'$.

Множество компонентов gобозначается как $C(g)$.

Функция υ называется *компонентно аддитивной*, если $\Sigma_{h \in C(g)} \upsilon(h) = \upsilon(g)$.

2. Примеры моделей сетей

Рассмотрим несколько примеров сетевых игр. Покажем, как игры могут задаваться.

2.1 Пример 1. Модель соавторов

Каждый игрок - это исследователь, который пишет научные работы. Если два игрока соединены, то это значит, что они пишут одну научную работу. Игрок i тратит на работу время, равное n_i, оно обратно зависит от количества проектов, в которых он участвует. Выигрыш игрока i равен

$$u_i(g) = \sum_{i \in N} 1/n_i + 1/n_j + 1/n_i * n_j.$$

2.2 Пример 2. Криминальные сети

Каждый игрок - преступник. Если два игрока связаны, то они являются частью одной преступной сети. Каждая группа связанных преступников имеет положительную вероятность выиграть добычу. Добыча распределяется между подключенными преступниками в зависимости от сетевой архитектуры. Выплата преступника i задается как

$$Y_i(g) = y_i(g) - p_i(g) * ф,$$

где $y_i(g)$ - ожидаемый выигрыш игрока i, $p_i(g)$ - вероятность того, что игрока i поймают, ф - численный эквивалент штрафа при поимке. Несмотря на то, что игроки соревнуются друг с другом за получение добычи, они также выигрывают за счет своих друзей-преступников. Предполагается, что чем больше группа взаимодействующих преступников, тем выше вероятность получить добычу, и чем больше связей у одного игрока, тем меньше у него вероятность быть пойманным.

Пусть $n_i(g)$ - количество связей игрока i в сети g. Так, $p_i(g)$ понижается при повышении $n_i(g)$. Преступник i, являющийся частью группы S, ожидает долю от выигрыша B, заданную формулами:

$$y_i(g) = \frac{1}{n} \cdot B, \text{ при } |S| = 1$$

$$y_i(g) = \frac{|S|}{n} \cdot \frac{n_i}{\sum_{j \in S} n_j} \cdot B, \text{ при } |S| \geq 2,$$

где $|S|/n$ - вероятность того, что группа S получит добычу, а $n_i / \sum_{j \in S} n_j$ - доля добычи, которую получит игрок i.

3. Формирование социальных сетей, стабильность

Игроки стремятся максимизировать свой выигрыш. При заданных правилах и состоянии графа g, игроки могут захотеть формировать новые связи, либо же отказаться от уже имеющихся. Процесс перехода от одной сети к другой назовем *формированием* социальных сетей.

Формирование сетей обычно начинается с пустой сети. В дальнейшем, модели, о которых пойдет речь, будут учитывать этот факт. Однако, стоит отметить, что это условие не всегда выполняется - в какой-то момент времени может поменяться какой-либо из параметров сети, например, количество игроков. Тогда формирование новых сетей пойдет уже от имеющегося графа отношений (или от графа, полученного после удаления или добавления новых игроков, связей, и т. п.).

Сеть назовем *стабильной*, на ней заканчивается процесс формирования. Это достаточно грубое определение, в дальнейшем будет показано, что конкретные условия стабильности могут различаться в зависимости от модели, используемой при анализе сетей.

В литературе ([3]) встречается подход к определению стабильности, который называется “негативной” методологией. Это значит, что сеть стабильна тогда, когда ни у одного из участников нет причин менять свои связи. Эта методология учитывает только то, какие сети не могут появиться из-за существующих стимулов к изменению сети, которыми наделены игроки.

3.1 Модель Майерсона. M-сети.

Ауман и Майерсон (1988) первыми смогли смоделировать формирование социальных сетей как игру в развернутой форме. В их модели игроки последовательно предлагают связи, которые впоследствии принимаются или отвергаются.

Подход к моделированию Майерсона основывается на стандартной некооперативной теории игр. Пусть N - множество игроков, игра на N - это пара (α, π), где $\alpha = (\alpha_1, \alpha_2,..., \alpha_n)$ - упорядоченное множество стратегий таких, что каждому игроку i поставлено в соответствие множество α_i, и функция выплат $\pi = (\pi_1, \pi_2,..., \pi_n): A \to \Re^N$, где $A = \prod_{i \in N} A_i$ - множество всех стратегий, сгенерированных в α.

В некооперативной игре каждый игрок i наделен своим индивидуальным набором стратегий A_i и функцией выигрыша $\pi_i: A \to \Re$. Основная идея состоит в том, что каждый игрок выбирает стратегию, которая оптимизирует его выплаты, при условии, что другие игроки также выбирают стратегии, которые влияют на этот выигрыш. По сути, игра представляет собой математическое представление ситуации социального взаимодействия. Теория игр теперь представляет собой набор правил и инструментов, которые моделируют то, как игроки принимают решения в контексте таких ситуаций социального взаимодействия.

Кортеж стратегий обозначим как $a = (a_1, a_2,..., a_n)$. Список стратегий, используемых игроками, кроме i, обозначим как $a_{-i} = (a_1, a_2,..., a_{i-1}, a_{i+1},..., a_n)$.

Майерсон не вводит для своей модели понятие “стабильность”. Вместо этого он исследует только равновесие по Нэшу.

Кортеж a^* назовем *равновесием по Нэшу* в игре (α, π), если для каждого игрока $i \in N$ и для каждой стратегии $b_i \in A_i$ функция выплат $\pi_i(a^*) \geq \pi_i(b_i, a^*_{-i})$. В равновесии по Нэшу каждый игрок оптимизирует свою стратегию, зная, какие стратегии выбрали для себя другие игроки.

Сама модель строится следующим образом. Игроки бесплатно показывают друг другу, хотят ли они строить связь или нет. Связь устанавливается тогда и только тогда, когда оба игрока показали друг другу, что

хотят участвовать в связи. Формально *модель Майерсона* Γ_{υ}^{m} на множестве игроков Nэто некооперативная игра $\Gamma_{\upsilon}^{m} = (\alpha^{m}, \pi^{m})$, для которой:

- Для каждого игрока $i \in N$ множество стратегий задано всеми векторами сигналов другим игрокам из N:

$$A_i^m = \{l_i = (l_{i1}, l_{i2},..., l_{in}) \mid l_{ij} \in \{0, 1\} \text{ и } l_{ii} = 1\}$$

- Связь ij сформирована, если оба игрока i и j показали друг другу, что хотят сформировать связь, то есть, если $l_{ij} = l_{ji} = 1$. Если обозначить профиль стратегий как $l = (l_1, l_2,..., l_n) \in A^m = A_1^m \times ... \times A_n^m$, то результирующую сеть можно зафиксировать как

$$g(l) = \{ij \in g^N \mid l_{ij} = l_{ji} = 1\}.$$

Говорят, что $g(l)$ - сеть, *поддерживаемая* профилем стратегий l в модели Майерсона.

- Модель Майерсона дополняется теоретической функцией выигрыша игры $\pi^m: A^m \to \Re^N$, определяемая как

$$\pi_i^m(l) = \upsilon_i(g(l)).$$

Назовем сеть g *М-сетью*, если существует такой профиль стратегий l^g, что $g(l^g) = g$.

В модели Майерсона ситуация, когда никто из игроков не дает сигнал на создание связи, является равновесием по Нэшу. Таким образом, пустая сеть является М-сетью (Лемма Майерсона). Лемма Майерсона указывает на фундаментальную проблему человеческого сотрудничества: Почему рациональные люди должны сотрудничать? В связи с этим лемма Майерсона является очень важным вопросом в социальных науках и экономике. [3, 4, 5, 6]

3.2 Попарная стабильность

Задача моделирования формирования сети, сформулированная при изучении модели Майерсона, была взята Джексоном и Волински (1996). Они сформулировали концепции кооперативного равновесия, адаптированные к конкретным требованиям моделирования формирования двусторонних связей. Это привело к понятию *попарно стабильной* сети.

Сеть g назовем попарно стабильной, если выполняются следующие условия: зафиксируем цену сети υ и правило распределения Y

(i) $\forall ij \in g,\ Y_i(g,\upsilon) \geq Y_i(g - ij,\upsilon)$ и $Y_j(g,\upsilon) \geq Y_j(g - ij,\upsilon)$,

(ii) $\forall ij \notin g,\ Y_i(g,\upsilon) < Y_i(g + ij,\upsilon) \Rightarrow Y_j(g,\upsilon) > (g + ij,\upsilon)$.

Первая часть определения парной стабильности требует, чтобы ни один игрок не желал удалять ссылку, в которой он участвует. Любой игрок имеет право в одностороннем порядке прекратить отношения, в которые он вовлечен. Вторая часть определения требует, что если какая-то связь отсутствует в сети и один из вовлеченных игроков выиграет от ее добавления, то должно быть так, что другой игрок пострадает от добавления ссылки. Здесь подразумевается, что для добавления связи необходимо согласие обоих игроков.

С попарной стабильностью легко работать, однако концепт попарной стабильности накладывает некоторые ограничения. Во-первых, это слабое понятие, поскольку оно учитывает отклонения только на одной вершине за раз.

Благодаря этому, с ним легко работать. Во-вторых, при исследовании попарной стабильности рассматриваются отклонения только для пары игроков за раз.

Попарная стабильность может рассматриваться как необходимое, но недостаточное требование для стабильности сети во времени. Тем не менее, попарная стабильность по-прежнему оказывается весьма полезной и, в частности, часто дает точные прогнозы относительно множества стабильных сетей. []

Формирование попарно стабильных систем рассматривается, как динамический процесс. Это значит, что прежде, чем прийти к попарно стабильной сети, последовательно перебираются другие сети, не удовлетворяющие условию попарной стабильности. Опишем процесс формально.

Назовем *улучшающем путем* от g до g' конечную последовательность сетей $g_1, g_2, \dots, g_K$, где $g = g_1$ и $g' = g_K$. Для последовательности должны выполняться следующие условия. Пусть $k \in [1, 2, \dots, K-1]$, тогда должно выполняться одно условие из двух:

(1) $g_{k+1} = g_k - ij$ для такого ребра ij, что $Y_i(g_k - ij) > Y_i(g_k)$

(2) $g_{k+1} = g_k + ij$ для такого ребра ij такого, что $Y_i(g_k + ij) > Y_i(g_k)$ и $Y_j(g_k + ij) \geq Y_j(g_k)$.

Иными словами, улучшающий путь - это последовательность сетей, каждая из которых отличается от предыдущей добавлением или удалением ребра и каждый игрок получает не меньший выигрыш, чем в предыдущей сети.

Для данной модели обычно предполагают близорукое поведение игроков (myopic behavior) - каждый игрок знает только о тех отношениях, в которых он участвует. Поэтому игроки не избавляются от ребер и не добавляют их, если от этого изменится выигрыш игрока, не участвующего в рассматриваемом взаимоотношении, и этот выигрыш как-то влияет на выигрыш данного игрока.

Циклы свойственны динамическим процессам, особенно в улучшающем пути. Множество сетей C назовем *циклом*, если для каждых сетей $g \in C$и $g' \in C$существует улучшающий путь от g до g'.

Цикл C назовем закрытым, если для ни одна сеть $g \in C$ не лежит на улучшающем пути, ведущем к сети $g^* \notin C$.

Доказано, что для игр с любыми υ, Y существует хотя бы одна попарно стабильная сеть или закрытый цикл попарно стабильных сетей [сеульский журнал].

Дополнительно Жан-Жак Херингс и Ана Молеон (2006) предложили концепт дальновидно стабильных сетей. Вкратце, это определение для сетевой игры, которое покрывает все попарно стабильные сети [2].

3.3 Модель связи эффективности и стабильности

Эффективной сетью назовем сеть g, для которой цена сети наибольшая:

$$\forall g' \in g^N: \upsilon(g) \geq \upsilon(g').$$

Сеть g назовем *эффективной по Парето*, если

$$\nexists g' \in g^N \text{ таких, что } Y_i(g') \geq Y_i(g),$$

для $\forall i$, со строгим неравенством для некоторых i.

Чтобы понять взаимосвязь между двумя определениями, обратим внимание, что g эффективен относительно υ, если он эффективен по Парето относительно υ и Y для всех Y.

Эффективные сети - простой и интуитивно понятный концепт. Однако, в определенных ситуациях, особенно, когда стоимость формирования связей не слишком высока или низка, эффективная сеть не является стабильной. В общем

случае, обратное тоже не верно - попарно стабильная сеть не является ни эффективной, ни эффективной по Парето.

Однако, если υ - компонентно аддитивная, то существует алгоритм поиска попарно стабильной и эффективной по Парето сети.

Пусть $g(\upsilon, S) = argmax_{g \in g^s} \upsilon(g)/\#N(g)$ - сеть, у которой наибольшее значение среди тех, которые могут быть сформированы на $S \subset N$.

Если υ - компонентно аддитивная, найти попарно стабильную и эффективную по Парето сеть g^{υ} можно следуя следующему алгоритму. Возьмем $h_1 \in g(\upsilon, N)$ с наибольшим количеством связей. Потом возьмем $h_2 \in g(\upsilon, N \backslash N(h_1))$ с максимальным количеством связей. Продолжим выбирать h. Тогда на шаге k выберем компонент $h_k \in g(\upsilon, N \backslash \bigcup_{i \leq k-1} h_i)$ с максимальным количеством связей. Когда останутся только пустые сети, останавливаемся. Объединение всех компонент h_j и будет искомая сеть g^{υ}.

3.4 Прочие модели

Вышеприведенные модели не исчерпывают все разнообразие подходов, представленных в литературе. Ниже приведем описание некоторых подходов.

- Сильная стабильность: Джексон и Нувеланд (2005) ввели более сильное требование для существования стабильности. Для создания требования, был использован подход, применяемый при попарной стабильности. В сильной стабильности, в отличие от попарной, учитываются отклонения на компонентах сети. Так преодолевается главный недостаток попарной стабильности - для последнего, как было указано выше, рассматриваются отклонения только на паре игроков.

- Стохастические модели: есть два динамических подхода, которые могут преодолеть трудность, когда поиск стабильной сети зашел в тупик. Один из них - избавиться от близорукости выбора игроков. Это имеет смысл в ситуациях, когда сети относительно небольшие, игроки знают друг друга и могут делать хорошие прогнозы относительно будущих игр. Другой подход состоит в том, чтобы внести некоторую случайность в процесс формирования сети, когда ссылки могут быть добавлены или удалены с помощью какого-либо экзогенного стимула или просто по ошибке. В рамках стохастического подхода выбираются случайные пары игроков и анализируется их связь, или отсутствие связи. Как и в случае с улучшающим путем, мы проверяем, хотят ли игроки, о которых идет речь, добавить ссылку, если ее нет в сети, или разорвать ссылку, если она находится в сети. Новое в этом подходе то, что намерения игроков выполняются с вероятностью $1 - \varepsilon$, где $0 < \varepsilon < 1$, а с вероятностью ε происходит обратное. Учитывая случайные изменения во время процесса, он может проходить достаточно долгое время и есть возможность посетить все возможные сети.

Некоторые идеи, предложенные в рамках этих подходов, будут использованы при реализации подходов, изложенных в пунктах 3.1-3.3.

4. Программная реализация. Общие принципы

Для реализации построения моделей на языке Python 3.8.7 были проанализированы имеющиеся подходы к работе с сетями. Оказалось, что задачи поиска стабильности не реализованы в известных пакетах. Более того, каждый из пакетов использует свою внутреннюю архитектуру данных, что

усложняет работу с графами. Поэтому было принято решение писать библиотеку по работе с графами собственноручно.

Весь программный код хранится в репозитории: https://github.com/KoljanTestwamore/stable_networks_modelling.

Сети хранятся в формате {[key: str]: set()}, то есть как словарь строковых ключей со значениями - множествами других ключей. Использование встроенной структуры множества - set() позволяет не волноваться о случайных дублированиях ребер. Также в Python [7], в отличие от многих других языков программирования, два разных множества с одинаковыми элементами при сравнении будут всегда давать True. В дальнейшем, элементы значения - множества будем называть *коннекторами* ключа.

Для большинства классов был реализован механизм цепной записи методов (chaining). После каждого вызова функции возвращается экземпляр класса, который вызвал функцию. Так мы можем вызывать последовательно несколько методов на одной строчке.

Большинство функций проектировались иммутабельными. Это значит, что вызов функции не изменяет объект, вызвавший функцию. Вместо этого он создает новый экземпляр класса. Из-за создания новых экземпляров программа может занимать больше памяти, чем ей требовалось для эффективной работы. Поэтому для функций также были написаны функции, повторяющие их работу, но не создающие новые объекты.

Был реализован ряд вспомогательных функций, упрощающих работу с графами. Коротко опишем некоторые из них:

- get_unique_keys(graph) - позволяет найти все вершины, упоминаемые в графе. Иногда некоторые вершины могут по ошибке не быть указаны в списке ключей, а только в качестве коннекторов другой вершины. Тогда, используя эту функцию, можно получить список всех ключей.
- get_all_graphs(graph) - возвращает массив из всех возможных массивов ребер на вершинах указанного графа. Реализовано с помощью создания

списка всех ребер M, а затем перебора комбинаций из этого списка, путем движения от 0 до $2^{|M|}$. На каждом шаге производится перевод итератора в двоичный формат, а затем - в финальный для шага список отбираются ребра, положение которых в M соответствует всем ненулевым битам итератора.

- memoize(f) - обертка над функцией f. Она запоминает все результаты выполнения f над определенным набором аргументов, и затем, если f вызывается опять над тем же набором аргументов, просто возвращает сохраненное значение. Это позволяет сохранить большое количество времени, если f очень трудоемкая.

Для описания каждой модели используется отдельный класс. Однако, описание сетей (графов) универсально для всех моделей. Для взаимодействия с ними был создан класс Configuration. Объекты класса представляют состояние графа, каждое состояние содержит ссылку на модель, для которой оно было создано. Класс содержит следующие функции:

- add_edges(edges: list) - создает новый экземпляр состояния, в котором содержатся ребра из параметра edges. Как можно заметить, ребра передаются списком (каждое ребро - кортеж), если нужно передать одно ребро - его нужно обернуть в массив.
- remove_edges(edges: list) - аналогично первому, но для удаления ребер.
- print() - консольный вывод сети
- xml() - сохранение текущей сети в формате “.gexf”. Подробнее, зачем это, будет рассказано в пункте 8. Визуализация.

4.1 Реализация примеров

Для всех реализованных классов для поиска стабильности или равновесия нужно задать правило распределения. Правило распределения - лямбда функция

формата $(x, g) \rightarrow float$, которая принимает первым аргументом внутренний номер игрока i, а вторым - состояние сети g. Механизм удобен, потому что позволяет задать сетевой игре любую функцию распределения.

Приведем реализацию функций распределения для примеров, указанных в пункте 2.:

- Модель соавторов:

```
def coauthors_rule(x, g):
    ni = len(g[x])
    if ni == 0:
        return 0
    result = 1 / ni
    for connector in g[x]:
        nj = len(g[connector])
        if nj == 0:
            continue
        result = result + 1 / nj + 1 / (ni * nj)
    return result
```

- Криминальные сети:

```
def criminal_networks_rule(x, g):
    fine = 10
    pot = 5
    S = detect_player_component(x, g)
    if S.get_size() == 1:
        y = pot / len(get_unique_keys(g))
    else:
        y = (S.get_size() / len(get_unique_keys(g))) * pot * S.get_share(i)
    p = 1 / (1 + len(connections(x, g)))
```

```
return y - p * fine
```

Как видно из примера с криминальными сетями, для реализации правил можно использовать собственные структуры и методы.

5. Реализация модели Майерсона

Создадим первоначальную версию модели. Введем класс для модели Майерсона. Класс будет содержать первоначальное положение сети, правило распределения. Нужно построить матрицу сигналов и составить сеть, удовлетворяющую пожеланиям игроков.

Составим матрицу пожеланий игроков. Для каждого игрока возьмем множество всех игроков без текущего игрока и посчитаем его вектор сигналов. Так как одновременно посчитать не получится ввиду программных ограничений, можем сделать процесс стохастическим: выбирать порядок исследуемых игроков случайно. Для этих целей в классе реализован механизм перемешивания списка игроков.

Посчитаем промежуточный выигрыш игроков. В дальнейшем, изменяя стратегию, каждый игрок будет выбирать только тот профиль стратегий, который увеличивает его выигрыш. Когда таких не останется, получим, что профиль игроков соответствует требованиям стабильности Майерсона. Составляем полученную сеть. Результат можно посмотреть на рисунке 1.

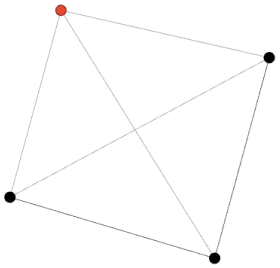

Рисунок 1, результат работы модели Майерсона, представленный в Gephi

6. Поиск попарно стабильной сети

Для поиска попарно стабильной сети также был реализован отдельный класс. В нем содержатся все нужные методы по работе с сетями. Для вывода сетей из улучшающего пути, класс также имеет статическое свойство, которое запоминает графы, лежащие на улучшающем пути.

Алгоритм поиска попарно стабильной сети можно описать так:

Шаг 1: Пронумеруем игроков, создадим пустой граф, вершины которого - номера игроков.

Шаг 2: Запустим поиск в ширину из начального графа, где каждый последующий элемент - граф, которому добавили или удалили ребро, мешающее выполнению условия стабильности, если такого ребра нет - отмечаем конфигурацию как стабильную. Сохраняем пройденные конфигурации, чтобы не считать их заново. Сохраняем стабильные конфигурации.

Шаг 3: Выводим стабильные конфигурации.

Пример визуализации работы алгоритма представлен на рисунке 2.

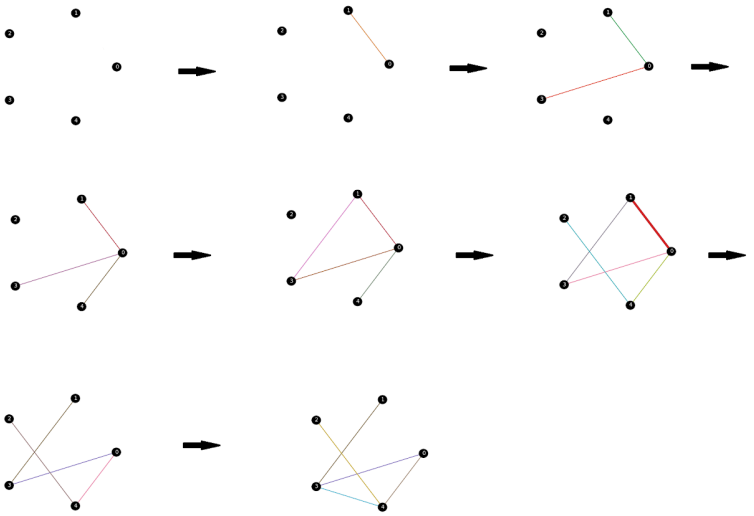

Рисунок 2, результат работы поиска попарной стабильности, собственный вывод

Алгоритм поиска чувствителен к точке начала вычислений. Поэтому для поиска большего количества попарно стабильных сетей, найдем все возможные графы на вершинах, представленных в первоначальной сети, и запустим алгоритм из них. Чтобы избавиться от лишних вычислений, выигрыш каждого игрока для каждого состояния графа будем запоминать с помощью memoize.

7. Реализация связи эффективности и стабильности

Алгоритм по реализации подробно описан в пункте 3.3 этой работы. Нужно только программно его реализовать. Повторим описание алгоритма.

Шаг 1. Строим первоначальную сеть

Шаг 2. Находим h_1.

Шаг 3. Повторяем шаг 2, находя h_i. Останавливаемся, когда остаются только пустые сети.

Шаг 4. Объединяем все компоненты h_j, получая искомую сеть.

Так как данный алгоритм тесно работает с понятием эффективности, существует потребность в создании способа подсчета эффективности сети. Один из возможных вариантов - найти все возможные графы на данном количестве игроков, запомнить их значения, и обращаться к ним впоследствии. Отметим, что если правило и количество игроков не меняется, значения эффективности можно записать в файл, чтобы не терять время при последующих запусках программы [5].

8. Визуализация результатов

Для представления результатов был сделан обзор нескольких продуктов и методов. Есть несколько больших проектов, помогающих визуализировать графы. Сильнее всего выделяются GUESS, Gephi и решения в библиотеке networkx для Python.

Система исследования графов (The Graph Exploration System - GUESS) - это инструмент с открытым программным кодом для анализа сетей. За счет комбинации системы визуализации и собственного языка (расширение Python), GUESS поддерживает множество типов графиков и инструментов их анализа [8].

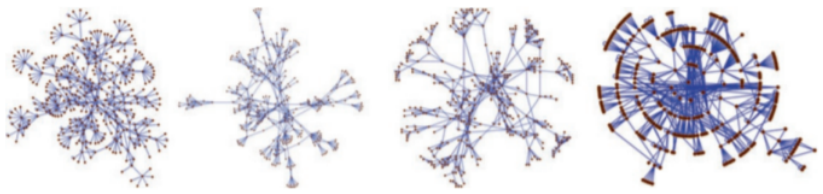

Рисунок 3, пример работы программы GUESS с сайта программы

Данное решение имеет ряд ограничений: использование нового языка и устаревший интерфейс. Авторы программного продукта не развивают его уже

несколько лет, а сайт, где можно получить программный продукт, не работает. Из-за этих причин от использования GUESS пришлось отказаться.

Gephi был создан в 2008 году сообществом авторов. Подходит для анализа всех видов сложных сетей, хотя в основном он используется для социальных сетевой анализ. Распространяется с помощью двойной схемы лицензирования в рамках GNU General Public License (GNU GPL) v3 и Общей лицензии на разработку и распространение (CDDL) v1. Gephi представлен как отдельное приложение, так и библиотека для языка программирования Java. [9]

Gephi позволяет в реальном времени работать с сетями, смотреть и моделировать их. Gephi принимает на вход файлы определенного формата, поэтому для работы с ним был реализован алгоритм сохранения графа g в формате GEXF.

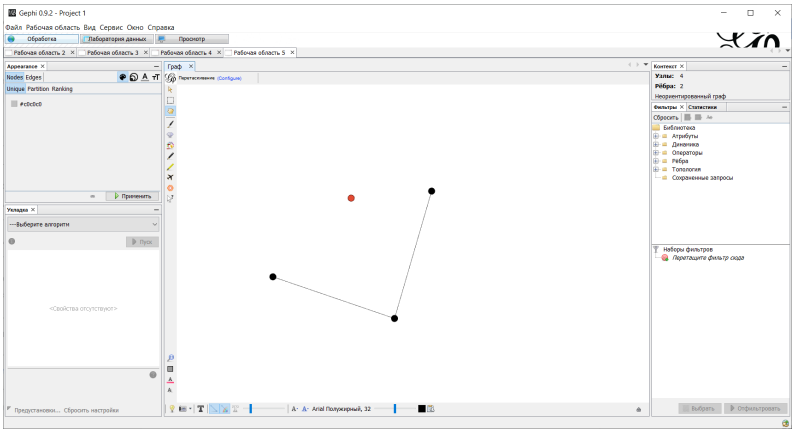

Рисунок 4, интерфейс программы Gephi

networkx - библиотека Python для изучения графов и сетей. Данная библиотека предлагает ряд подходов и решений для взаимодействия с графами [10]. В данной библиотеке также имеются способы визуализации.

От подобных способов пришлось отказаться, так как библиотека работает со своим форматом графов, и перевод графов из формата, который используется в программе в формат networkx может оказаться излишним и слишком сложным.

Основной недостаток стороннего ПО для визуализации решения - ограниченная область задач, к решению которых оно может быть применено. Так, для задачи поиска попарной стабильности, нужно визуализировать последовательность графов, что тяжело добиться в формате GEXF. Поэтому для работы программы был написан специальный метод визуализации, используя распространенную библиотеку для языка Python - matplotlib [11].

Написанный метод вывода позволяет вывести несколько графов на одной картинке, четко показать, какие ребра удаляются или добавляются на следующем шаге.

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы распространенные подходы к моделированию социальных сетей, создана их программная реализация. В рамках поставленной задачи был разработан ряд инструментов по работе с изученными моделями. Был проведен обзор имеющихся инструментов по работе с сетями и визуализацией сетей.

Теперь для языка Python имеется инструмент по поиску стабильных сетей, основанных на распространенных подходах к моделированию. В дальнейшем можно опубликовать этот инструмент как пакет, чтобы была база для будущих исследований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Joan de Marti, Yves Zenou “Segregation in Friendship Networks” — Scandinavian Journal of Economics, 119(3), 658-708, 2017

[2] P. Jean-Jacques Herings, Ana Mauleon, Vincent Vannetelbosch, “Farsightedly Stable Networks” — CORE Discussion Paper, 2006-92

[3] Matthew O. Jackson, “A Survey of Models of Network Formation: Stability and Efficiency” — HSS 228-77, California Institute of Technology (2003)

[4] Tim Hellmann, Jakob Landwehr, “Pairwise Stable Networks in Homogeneous Societies” — Center for Mathematical Economics Working Papers (IMW), 2018, 517

[5] Robert P. Gilles - “Building social networks under consent: A survey” Economics Group, Management School, Queen’s University of Belfast. 2019

[6] Matthew O. Jackson, Alison Watts “The Existence of Pairwise Stable Networks” — Seoul Journal of Economics, A14, D20, J00

[7] Python documentation — [электронный ресурс] // URL: https://docs.python.org/

[8] Eytan Adar, “Guess” — School of Information and Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA 2013

[9] Sébastien Heymann “Gephi” — LIP6, Université Pierre et Marie Curie (UPMC)

[10] networkx documentation — [электронный ресурс] // URL: https://networkx.org/

[11] Matplotlib documentation — [электронный ресурс] // URL: https://matplotlib.org/3.1.1/api/