Санкт-Петербургский государственный университет

Кафедра технологии программирования

Порохнявая Ольга Юрьевна

Магистерская диссертация

Разработка программного обеспечения для моделирования эпидемических процессов в безмасштабных сетях

Направление 02.04.02

Фундаментальные информатика и информационные технологии

Магистерская программа «Технологии баз данных»

Научный руководитель,

кандидат физ.-мат. наук,

доцент

Губар Е.А.

Санкт-Петербург

2017 г.

Оглавление

Введение

Большинство различных структур в современном обществе можно представить в виде сетей, где узлы обозначают людей, организации или иную сущность, а связи описывают взаимодействие между ними. Таким образом принято моделировать компьютерные сети, сети контактов (социальные сети) и даже сеть Интернет.

После публикации на рубеже 50-х и 60-х годов прошлого века статей Эрдеша-Реньи об эволюции случайных графов [1], теория построения таких сетей стала активно развиваться. В первой модели случайного графа, предложенной авторами статьи, вершины графа были фиксированы, а случайным представлялось множество ребер. Любые две вершины соединялись ребром с некоторой вероятностью, одинаковой для всех ребер и независимой от всех остальных пар вершин. Под эволюцией понималось изменение свойств графов с ростом этой вероятности. Оказалось, что при некоторых значениях этой вероятности происходит кардинальное изменение свойств графа. На эту тему было проведено множество интересных исследований.

Однако, в начале нашего века выяснилось, что реальные графы, возникающие в различных областях (напрмер, графы таких социальных сетей как Facebook, Twitter и т.д.), плохо укладываются в модель Эрдеша-Реньи. Это спровоцировало новый всплеск исследований математических моделей случайных графов [2].

Одной из разновидностей случайных графов стала безмасштабная сеть. Было показано, что именно безмасштабные сети наиболее удачно моделируют структуру сети Интернет, а так же большинства социальных и транспортных систем. На данный момент разработано несколько различных подходов к построению безмасштабных сетей [3].

Моделирование сетей интересно не только само по себе, но и для рассмотрения происходящих в этих сетях процессах. Исследования показали, что многие из этих процессов имеют схожую природу с биологическими и эпидемическими процессами. Так например, процесс распространения информации в социальных сетях имеет тот же механизм, что и распространение биологического вируса в популяции.

В такой области как эпидемиология математическое моделирование является незаменимым инструментом для исследования и прогнозирования, так как реальные эксперименты в этой сфере затруднены или даже невозможны. Первая модель распространения эпидемии была предложена в 1927 году Кермаком и Кендриком [4]. Позже было создано множество других эпидемических моделей, более точно описывающих конкретные процессы. Разработанные математические модели эпидемий в основном представляют собой системы дифференциальных уравнений.

В последующем этот подход получил широкое развитие и сейчас используется для описания распространения компьютерных вирусов, слухов, идей в обществе, вирусной рекламы, инфекционных заболеваний [5–9].

Таким образом, моделирование распространения информации в сетях является одним из наиболее значимых приложений. В работе [10] приведена модель распространения компьютерного вируса в безмасштабной сети. Авторы одними из первых стали рассматривать эпидемические процессы на сетях и представили систему нелинейных дифференциальных уравне-

ний, которая описывает распространение вируса с учетом параметров сети.

В первой части настоящей работы подробно рассматриваются безмасштабные сети и эпидемические модели. Затем приведено описание эпидемического процесса на сети, введены новые понятия. Некоторые особенности из теории эпидемических моделей видоизменяются в приложении к сетям, поэтому приведено описание таких изменений.

Результатом работы является программный комплекс, написанный на языке Java, позволяющий проводить моделирование эпидемический процессов, происходящих по различным законам в различных сетях. Во второй части работы приводится описание разработанного программного комплекса. Отдельно рассмотрена архитектура и ограничения полученного продукта. Далее приводится пример использования программы для исследований в рассматриваемой области, а также результаты тестирования производительности.

В открытом доступе не обнаружено аналогов разработанного программного комплекса. Встречаются упоминания использования программы для имитационного моделирования AnyLogic [11] при моделировании эпидемических процессов [12]. Но такое моделирование не берет в расчет сети и кроме того, требует дополнительных навыков для работы с этим инструментом. Сам инструмент AnyLogic не является свободно распространяемым.

Так же упоминаются специализированные инструменты для прогнозирования конкретного эпидемического процесса, например, в конце прошлого века в ГУ НИИЭМ им. Н.Ф. Гамалеи РАМН *«была реализована уникальная «коллекция» математических моделей в виде Windows-приложений для изучения эпидемий и вспышек значимых инфекций с феноменологией типа SEnImRF, где: En — n стадий инкубационного периода; Im —*

m стадий (различных клинических форм инфекционного заболевания); R — переболевшие заболеванием, F — погибшие от осложнений» [14]. Сама коллекция так же отсутствует в открытом доступе, но по описанию можно понять, что каждый вид эпидемической модели «жестко» вшит в отдельное приложение.

Таким образом, разработанная система уникальна на данный момент.

В приложении к данной работе можно найти основные фрагменты кода полученного программного продукта, содержащие алгоритмы построения сети, структуру описания эпидемических моделей и универсального для различных моделей алгоритма расчета пошагового распространения эпидемии.

Основные понятия

Прежде чем перейти к постановке задачи, опишем основные понятия, которыми в последствии будем оперировать.

Базмасштабная сеть

Определение 1. *Пусть есть* $V_n = \{1, \ldots, n\}$ *— множество вершин графа. Будем соединять любые две вершины* i *и* j *ребром с некоторой вероятностью* $p \in [0, 1]$ *независимо от всех остальных* $C_n^k - 1$ *пар вершин. Обозначим через* E *случайное множество ребер, которое возникает в результате реализации такой схемы. Положим* $G = (V_n, E)$*. Это и есть случайный граф (модель Эрдеша — Реньи) [1].*

Определение 2. *Безмасштабная сеть (Scale-Free Network, SF) — это случайный граф, где распределение связей узлов степенное и основные свойства сети не зависят от ее размера. Доля узлов в сети, имеющих* k *связей, составляет* $P(k) \sim k^{-\gamma}$ *для больших значений* k *[15].*

Замечание 1. *Величина* $P(k)$ *так же определяет вероятность того, что выбранный узел имеет ровно* k *связей.*

Основной особенностью безмасштабных сетей является наличие так называемых хабов (англ. hubs) или, по-другому, узлов-концентраторов. Такие узлы имеют очень большое количество связей по сравнению с аналогичным показателем остальных узлов. В ходе исследований было выяснено, что в безмасштабных сетях крупные хабы чаще всего окружены хабами

меньшего размера, а те, в свою очередь, еще меньшими и т.д. Это свойство делает безмасштабные сети практически нечувствительными к повреждениям, так как при потере одного хаба почти все связи сохранятся за счет его меньших соседей.

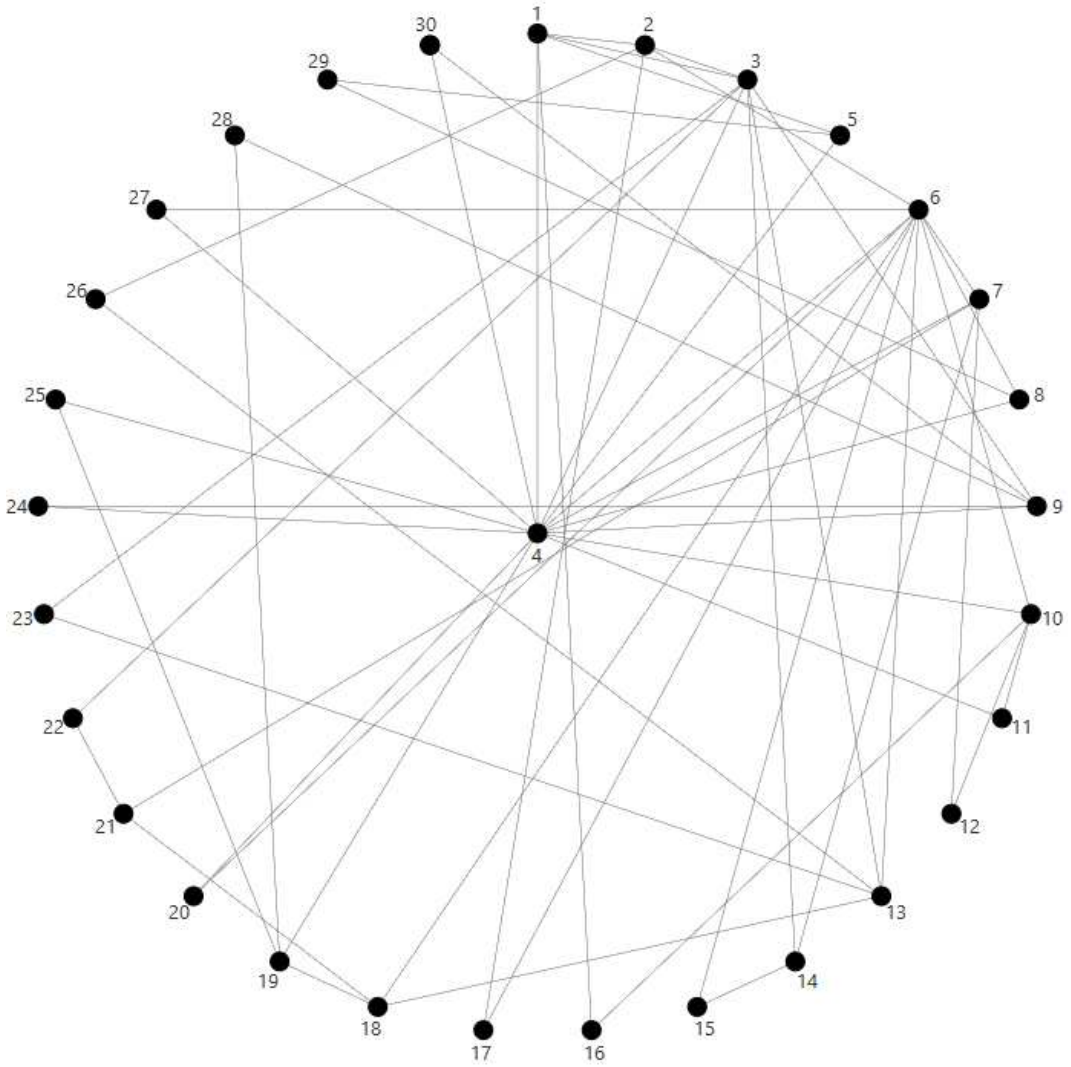


Рис. 1: Пример безмасштабной слабо связанной сети из 30 узлов. Узел 4 — наибольший хаб, имеет 14 связей. Узлы 6 и 3 — хабы меньшего размера, имеют 11 и 8 связей соответственно. Основная масса узлов имеет только $2-3$ связи.

Безмасштабные сети хорошо подходят для моделирования сетей контактов и сети Интернет.

В статье [10] описан базовый алгоритм построения безмасштабной сети с значением $\gamma = 3$. Его основная идея состоит повышении вероятности

получить новую связь с увеличением количества уже имеющихся, реализуя таким образом «относительный Эффект Матфея» (принцип «богатые богатеют»).

Этот и другие алгоритмы генерации безмасштабных сетей с определенной вероятностью, зависящей от параметров алгоритма, генерируют связные графы. Для моделирования эпидемического процесса графы с изолированными компонентами не представляют интереса — в них будут проходить независимые эпидемические процессы, которые можно воспроизвести на отдельных связных графах. Поэтому в данной работе мы будем использовать расширенный алгоритм, разработанный ранее и описанный в статье [13]. В результате его работы будет получена сеть, где каждая пара узлов связана не более чем одной связью, отсутствуют петли и изолированные участки.

Эпидемические модели

Ввиду широкого распространения процессов, имеющих схожий механизм с эпидемическими процессами, представляет интерес возможность пошагового отслеживания распространения вируса.

Модели популяционного уровня строятся на основе знания о взаимодействии отдельного индивида с инфекцией и с другими индивидами. Классическая эпидемиологическая модель выглядит следующим образом. Популяция делится на три группы:

- S (от английского susceptible) — индивиды, которые еще не заражены, но восприимчивы к вирусу
- I (от английского infected) — зараженные, инфицированные индивиды.
- R (от английского removed) — индивиды, более не восприимчивые к болезни, получившие иммунитет или умершие.

Внутри группы все индивиды считаются одинаковыми по своим свойствам. Из первых букв английских названий этих состояний образовано общепринятое обозначение данной модели: SIR.

Переходы индивидов из группы в группу, то есть смена их состояний описывается системой дифференциальных уравнений:

$$\begin{aligned} \frac{dS}{dt} &= -\alpha SI; \\ \frac{dI}{dt} &= \alpha SI - \beta I; \\ \frac{dR}{dt} &= \beta I. \end{aligned} \tag{1}$$

Здесь α — это вероятность передачи инфекции при встрече восприимчивого и инфицированного индивидов, произведение SI означает вероятность такой встречи. А β — вероятность выздоровления и приобретения иммунитета зараженным индивидом.

Замечание 2. *Процесс часто рассматривается в дискретном виде и может быть описан разностными уравнениями. Тем не менее, в данной сфере исторически сложилось, что смена состояний описывается дифференциальными уравнениями.*

Конечно, описанная здесь модель SIR является лишь одной из многих. Еще одним распространенным примером можно назвать модель SIS. Эта модель описывает явления или заболевания, к которым не вырабатывается иммунитет и выздоровевший индивид вновь становится восприимчивым (переходит из состояния I в состояние S). Некоторые процессы требуют введения дополнительного состояния, отражающего латентных, нераспространяющих инфекцию носителей вируса. SEIR является одной из таких моделей. Для смертельных заболеваний вводится подгруппа D, соответствующая умершим индивидам.

Для социальных процессов характерно распространение двух конку-

рирующих идей, таких что «заражение» одной из них исключает возможность «подхватить» другую.

Даже имея одинаковый набор состояний, модели могут отличаться правилами переходов. Например, имея состояния S, I и R, в одном случае интерпретация предполагает, что любой заболевший обязательно рано или поздно приобретет иммунитет, тогда как в другом для него остается вероятность заново перейти в S и даже подвергнуться заболеванию еще раз.

Все эти различные виды эпидемических процессов делают количество моделей для их описания бесконечным. Множество различных моделей описано в работе [16].

Эпидемический процесс на сети

В классическом представлении эпидемические модели отображают процесс распространения инфекции в популяции без учета ее внутренней структуры. Такое представление аналогично предположению, что внутри популяции каждый индивид контактирует со всеми остальными.

На практике оказывается, что значительное влияние на эпидемический процесс имеет структура общества, в котором распространяется та или иная идея. Распространение будет значительно замедленно по сравнению с классическим вариантом, если структура популяции соответствует слабо связному графу. Более того, с одинаковыми параметрами модели, отвечающими за вероятность передачи вируса, процесс может значительно отличаться в зависимости от состояния конкретных индивидов. Был ли в числе зараженных на начальном этапе «популярный» (имеющий большое число связей) узел? Или, может быть, носителями идеи оказались «необщительные» члены общества, так и не сумевшие передать информацию достаточному числу других участников, прежде чем перестали ей интере-

соваться (переход в состояние R в описанной модели SIR)?

Таким образом, имеет смысл рассматривать эпидемические модели в сочетании с конкретной сетью, моделирующей рассматриваемое общество.

Новая область исследования требует введения некоторых определений.

Определение 3. *Состояние сети — характеристика, описывающая в каком состоянии из данной модели пребывает каждый узел сети в конкретный момент времени, представляет собой набор пар (узел – состояние)*

Определение 4. *Эпидемический процесс на сети — упорядоченная по времени последовательность состояний сети*

Определение 5. *Шаг эпидемического процесса на сети — процесс перехода сети к новому состоянию, которое вычисляется из текущего состояния и уравнений переходов выбранной эпидемической модели.*

Здесь стоит отметить, что имея фиксированную сеть, мы точно можем сказать, контактируют два конкретных индивида или нет. Поэтому, понятие «вероятность встречи индивидов» из привычной интерпретации модели отсутствует в случае рассмотрения эпидемического процесса на сети. Вместо этого появляется следующая характеристика состояния сети: «вероятность того, что случайно выбранная связь ведет к узлу в состоянии X» (обычно обозначается θ_X). Она учитывает в себе количество узлов в состоянии X на данном шаге и количество связей у этих узлов. Таким образом она отображает, сколько потенциальных контактов, передающих это состояние, может случиться на ближайшем шаге.

Например, эта характеристика имеет смысл для состояния I в модели SIR. Увеличенное значение θ_I обозначает истинность одного из двух либо сразу обоих утверждений:

- увеличено количество восприимчивых узлов, которые, возможно, перейдут в состояние I;
- увеличена вероятность такого перехода, поскольку чем больше у восприимчивого узла инфицированных «соседей», тем больше вероятность получения им вируса на текущем шаге, так как передача вируса возможна от каждого инфицированного «соседа»;

В итоге, совмещая эпидемическую модель с заданными параметрами и сеть, установив начальное состояние сети, мы можем наблюдать эпидемический процесс на этой сети путем выполнения шагов эпидемического процесса.

Постановка задачи

Итак, учитывая распространенность эпидемических процессов в природе и обществе, а так же необходимость учитывать структуру сети, представляется востребованным программный сервис, который мог бы просчитывать эпидемический процесс на сети и отображать его по шагам.

Поскольку структура сети имеет высокое влияние на процессы распространения, исследователю необходимо моделировать рассматриваемую популяцию. Наиболее важными параметрами в такой модели представляются размер популяции и среднее количество связей ее индивидов.

Кроме того, существует множество различных эпидемических моделей, созданных для описания различных процессов. И тем не менее, зачастую существующих моделей бывает недостаточно. В таком случае, изучая конкретный процесс распространения, исследователь создает новую модель, стремясь наиболее точно отразить реальную ситуацию. Возникает потребность проверять различные модели, сравнивать их между собой с целью найти наиболее точное соответствие имеющимся данным. Этот факт выявляет потребность так же в инструменте, который позволит не только применять уже разработанные модели для моделирования эпидемического процесса, но и создавать свои собственные.

Из этих предпосылок можем сформулировать цель работы.

Цель. Создать программный комплекс, который позволяет:

- сгенерировать безмасштабную сеть по заданным значениям размера и средней связности в популяции;
- сконструировать произвольную эпидемическую модель путем задания перечня возможных состояний и правил перехода индивидов между этими состояниями;
- воспроизвести распространение эпидемии по выбранной модели на выбранной сети.

Предполагается, что данный комплекс позволит получать новые интересные результаты в области эпидемических процессов.

Программный комплекс

Требуемый программный продукт был разработан на языке Java. В качестве хранилища данных использована объектно-ориентированная БД db4o.

Архитектура

Программа состоит из трех основных блоков в соответствии со сформулированными пунктами задачи:

- Конструктор сети. Пользователь задает несколько параметров, по ним генерируется сеть, используя ранее разработанный алгоритм.
- Конструктор эпидемической модели. Пользователь задает перечень возможных состояний узла и описывает правила переходов.
- «Проигрыватель». Пользователь выбирает одну из ранее сохраненных сетей и модель, блок выполняет шаги эпидемического процесса и последовательно отображает состояния сети.

Как можно заметить, первые два блока представляют собой абсолютно самостоятельные модули. Результатом их работы является некоторый объект, который сохраняется в базу данных. Третий блок наоборот, получает из базы данных модель и сеть, на которой будем запускать эпидемический процесс. Результатом его работы является новый объект, представляющий собой последовательность состояний сети.

Детали реализации

Остановимся подробнее на каждом блоке.

Конструктор сети

Этот блок оперирует объектами типа Network и Node. Получив от пользователя информацию о размере и средней связности, запускает алгоритм создания безмасштабной сети, после чего отображает ее. Если результат удовлетворительный, полученный объект можно сохранить.

Конструктор модели

Объект Model содержит в себе перечень возможных состояний индивида (для каждого состояния дополнительно указывается цвет, которым будут помечаться узлы в таком состоянии) и правила переходов.

Правило перехода представляет собой объект, в котором указаны:

- начальное состояние;
- конечное состояние;
- требуется ли связь с узлом в конечном состоянии для перехода (в модели SIR переход из S в I требует наличие зараженного «соседа» у восприимчивого узла) или переход осуществляется без связи (в модели SIR переход из I в R не требует наличия соседа в состоянии R);
- обозначение коэффициента вероятности перехода.

Для удобства проверки правильности модели формируется схема в принятом в этой сфере виде и уравнения переходов.

Проигрыватель

Объект Epidemic получает в конструкторе объекты Model и Network. Дополнительно указывается начальное распределение узлов по состояни-

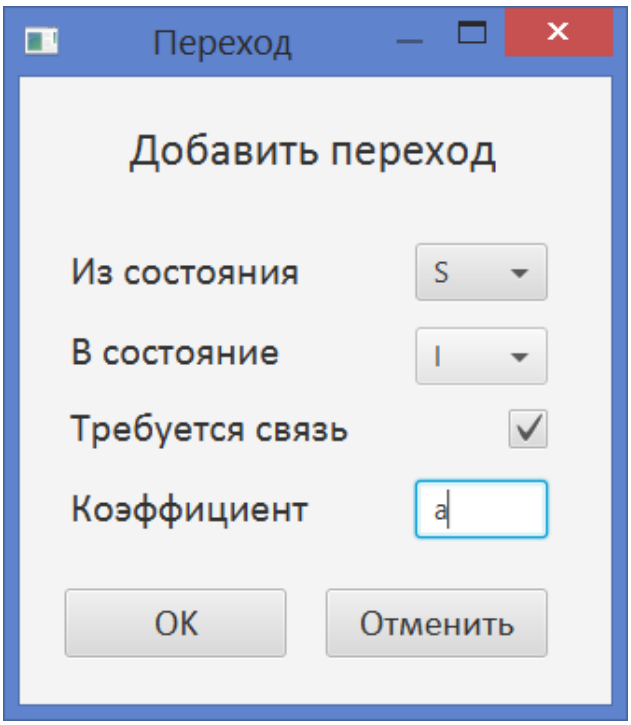


Рис. 2: Интерфейс для задания связи.

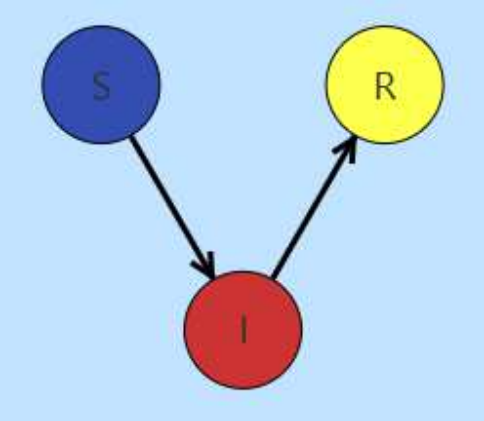
ям модели — по этой информации генерируется начальное состояние сети, и устанавливаются значения коэффициентов вероятностей переходов для данной модели.

Этот модуль отвечает за просчитывание шагов эпидемического процесса.

Рассчет шага

Все узлы сети делятся на группы согласно их состоянию на текущем шаге. Для каждого состояния проверяется наличие исходящего перехода. Если таких нет, на следующем шаге узел остается в том же состоянии. Если есть, возможны два варианта:

- **Для перехода требуется связь с узлом в соответствующем состоянии.** Тогда получаем всех соседей данного узла, для каждого из них проверяем, в каком состоянии он находится на текущем шаге. Если это нужное нам состояние, получаем случайное значение $[0; 1)$



$$dS/dt = -a*SI$$

$$dI/dt = +a*IS - b*I$$

$$dR/dt = +b*R$$

Рис. 3: Схема и уравнения переходов сформированы из введенных данных.

и проверяем состоялся ли переход в соответствии с коэффициентом вероятности. Если переход состоялся, переходим к следующему узлу. Если переход не состоялся, переходим к следующему соседу данного узла. Таким образом, чем больше «зараженных» соседей, тем выше итоговая вероятность перехода!

- **Для перехода не требуется связи.** Сразу получаем случайное значение и проверяем состоялся ли переход.

После этого процесса получаем новое распределение узлов по группам согласно их состоянию. Оно формирует следующее состояние сети.

Помимо прочего, здесь интересным является случай, когда из одного состояния есть переходы в два или более других. Необходимо проводить проверку перехода узла в какое-либо состояние за одно действие, иначе выбранная очередность проверок внесет изменения в итоговые вероятности перехода, уменьшая реальную вероятность перехода во второе состояние в списке.

Если обозначить через α коэффициент вероятности перехода, а через r — результат получения случайного значения из $[0; 1)$, то в обычном случае переход состоялся при условии $r < \alpha$. Если появляется еще один вариант перехода из данного состояния с коэффициентом β, то при усло-

вии $r < \alpha$ состоялся переход в состояние с коэффициентом α, при условии $\alpha <= r < \alpha + \beta$ состоялся переход в состояние с коэффициентом β, а при условии $r >= \alpha + \beta$ переход не состоялся.

Таким образом, если из одного состояния можно перейти в два или более других, сумма вероятностей этих переходов не должна превышать 1, причем равенство единице будет означать, что узел обязательно покинет текущее состояние на первом же шаге.

Каждое состояние сети можно отобразить, получив, таким образом, демонстрацию динамического процесса.

Ограничения

У разработанного программного комплекса есть несколько ограничений.

- Хотя алгоритм расчета шагов эпидемического процесса способен быстро обработать достаточно крупную сеть, удовлетворительно отобразить можно только сеть небольшого размера.

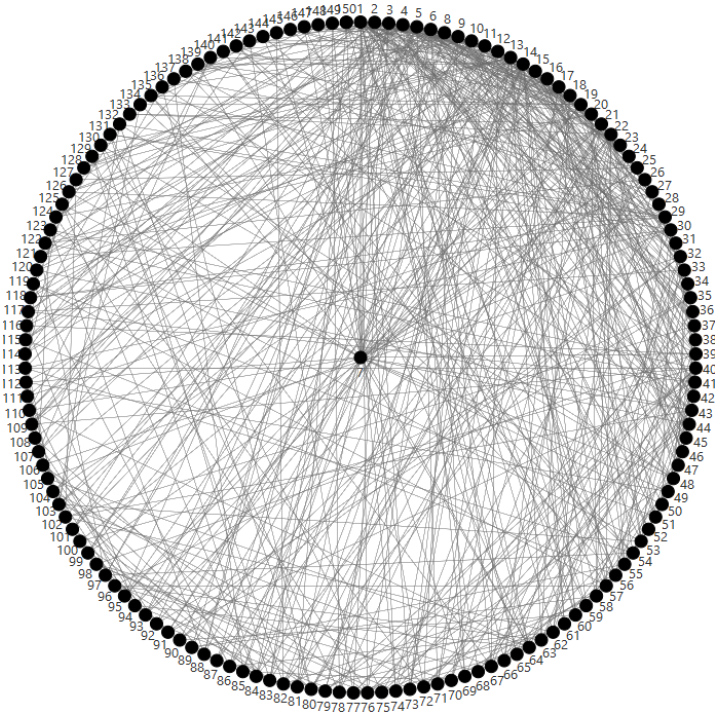


Рис. 4: Слишком большая сеть для отображения. 150 узлов.

- В модели не может быть более одного перехода из одного состояния в другое, даже разных типов (с требованием наличия связи и без). Это ограничение введено для избежания неопределенности. Были проанализированы различные существующие модели, моделей, где это требование нарушается, не обнаружено.

Показатели производительности

Было проведено тестирование производительности отдельных модулей.

- Построение безмасштабной сети на 100 узлов со средним количеством связей узла равным 10, в среднем занимает 2 миллисекунды (проведено 100000 измерений).
- При увеличении показателя связности до 30 при том же размере сети, среднее время достигает 17 миллисекунд (проведено 100000 измерений).
- Теперь рассмотрим сети большего размера. Сеть на 500 узлов с показателем связности 10 в среднем строится за 120 миллисекунд (10000 измерений).
- Построение сети на 500 узлов с показателем связности 30 занимает 870 миллисекунд в среднем (10000 измерений).
- Сеть размером 1000 узлов с маленьким показателем связности 10 в среднем строится за 400 миллисекунд (10000 измерений).

В этих экспериментах замерялось время без отображения сети в пользовательском интерфейсе. Кроме того, хотя сети размера 500 и 1000 успешно генерируется, отображение такой сети будет нечитаемым.

Теперь рассмотрим обсчет шага эпидемического процесса по модели SIR. К концу процесса время шага уменьшается и сокращается почти до нуля, поскольку процесс приходит в стационарное состояние и больше не происходит переходов. По этой причине мы будем замерять максимальное время шага в каждом процессе и усреднять его по всем процессам.

- Шаг эпидемического процесса на небольшой сети из 30 узлов со средней связностью 8 занимает около 2 миллисекунд.
- На сети размером 100 со средней связностью 10 максимальный шаг выполнялся 6 миллисекунд.
- Увеличим связность до 30 и получим время 10 миллисекунд.
- На крупной сети из 500 узлов при средней связности 30 получаем значения порядка 150 миллисекунд на один шаг.

Эта часть эксперимента использовала данные 10000 замеров по 100 шагов эпидемического процесса на каждую сеть.

Пример использования

Разработанный программный комплекс не требует специальных навыков для использования.

Опишем процесс построения модели эпидемического процесса.

Шаг 1: выбор сети. На рисунке приведен интерфейс основного окна программы. Для моделирования эпидемического процесса на сети необходимо выбрать сеть. Имена всех сохраненных сетей показаны в выпадающем списке.

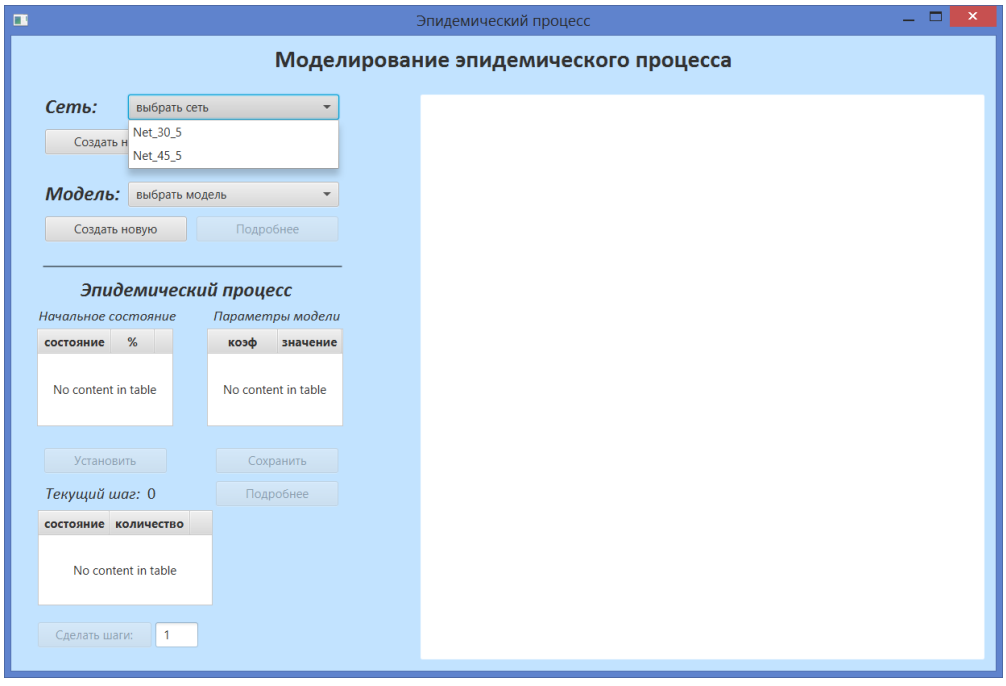


Рис. 5: Начальный экран, выбор сети.

Шаг 1.1: создание новой сети. Предположим, сохраненные варианты нас не устроили. Тогда нажав на кнопку "Создать новую"под выбо-

ром сети, мы попадаем в экран создания сети.

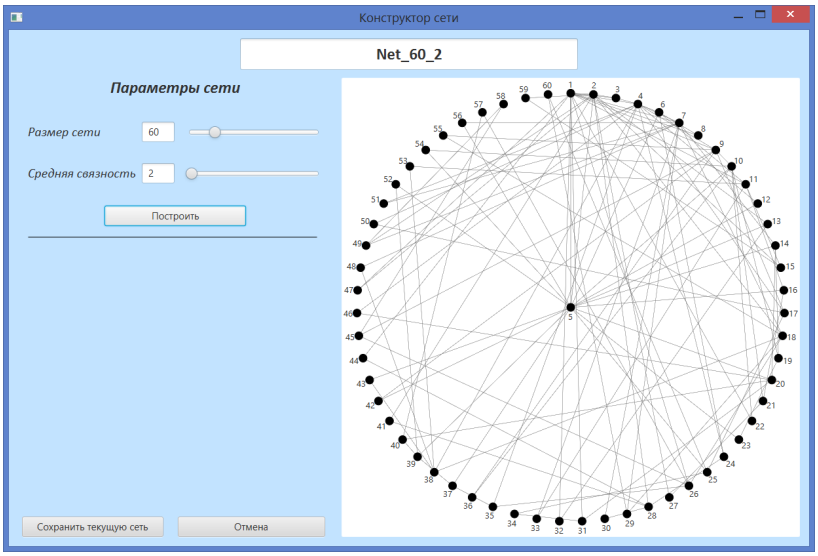


Рис. 6: Интерфейс создания сети.

Здесь мы можем устанавливать параметры размера и связности, после чего по нажатию кнопки «Построить» генерируется сеть. Добившись нужного варианта, сохраняем сеть и возвращаемся в предыдущий экран.

Шаг 2: выбор эпидемической модели. Теперь настала очередь выбирать модель. Сейчас в базе данных сохранена только классическая модель SIR. Исправим это, нажав на кнопку «Создать новую» в блоке модели.

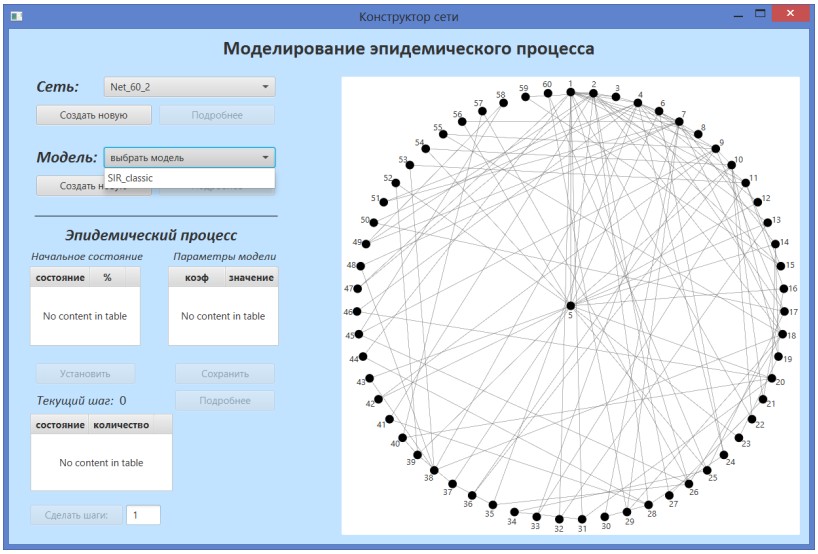


Рис. 7: Начальный экран, выбор модели.

Шаг 2.1: создание новой модели. Первым делом сформируем список состояний модели. Для каждого состояния необходимо задать цвет, которым будут помечены соответствующие ему узлы на этапе моделирования процесса. Можно так же указать описание состояния.

Рис. 8: Интерфейс создания модели.

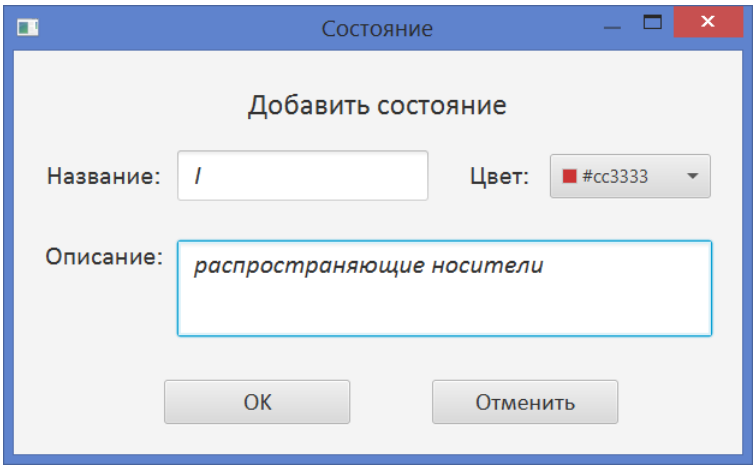


Рис. 9: Диалог добавления состояния.

По мере добавления состояний обновляется схема и уравнения модели. Пока в нашей модели нет никаких переходов из состояния в состояние, поэтому все изменения равны нулю. Исправим это.

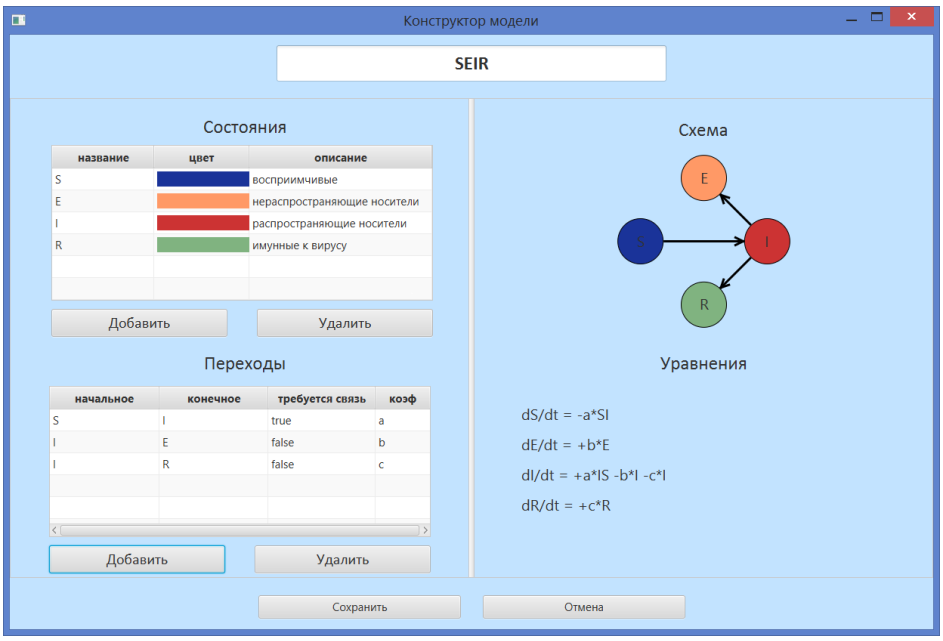


Рис. 10: Интерфейс создания модели.

Отлично! Модель дополнилась переходами, схема и уравнения дополнились. Если мы получили то, что хотели, сохраняем модель.

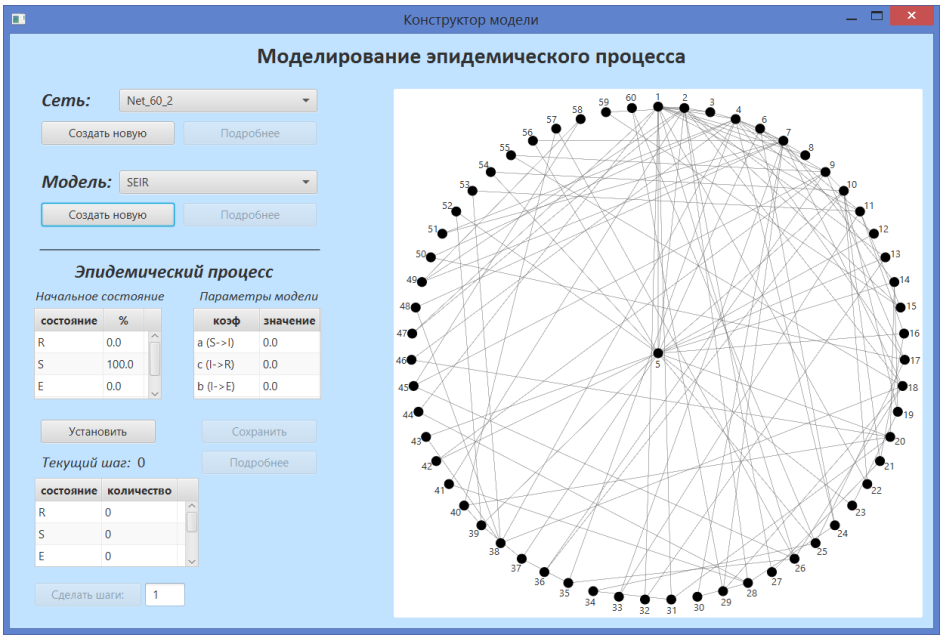


Рис. 11: Начальный экран.

Шаг 3: начальные параметры. Теперь, когда сеть и модель выбраны, нужно задать начальное распределение узлов по состояниям (по нему будет сгенерировано начальное состояние сети) и установить значения коэффициентов переходов. Сделать это можно, отредактировав соот-

ветствующие таблички.

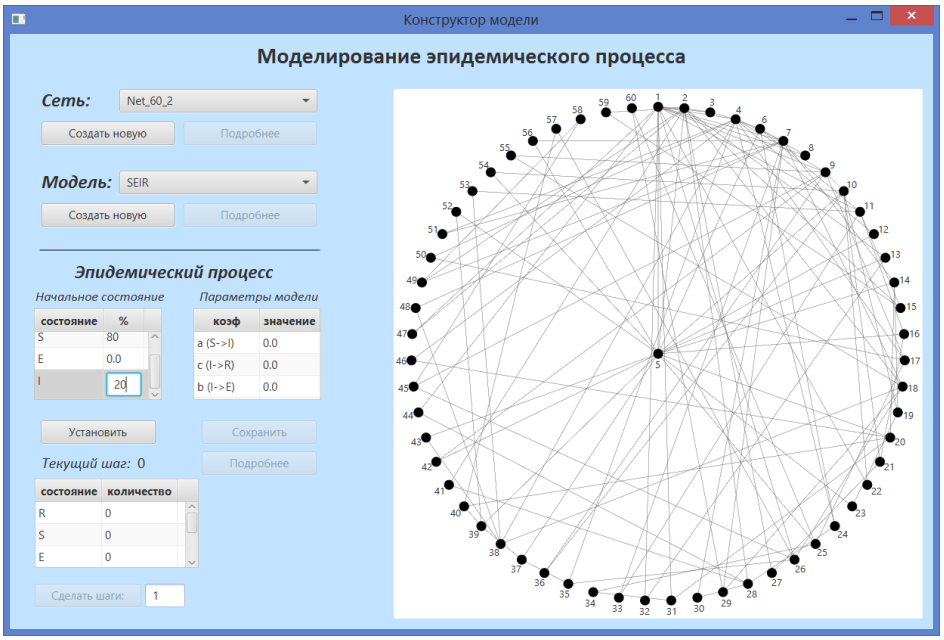


Рис. 12: Начальные параметры процесса: распределение узлов.

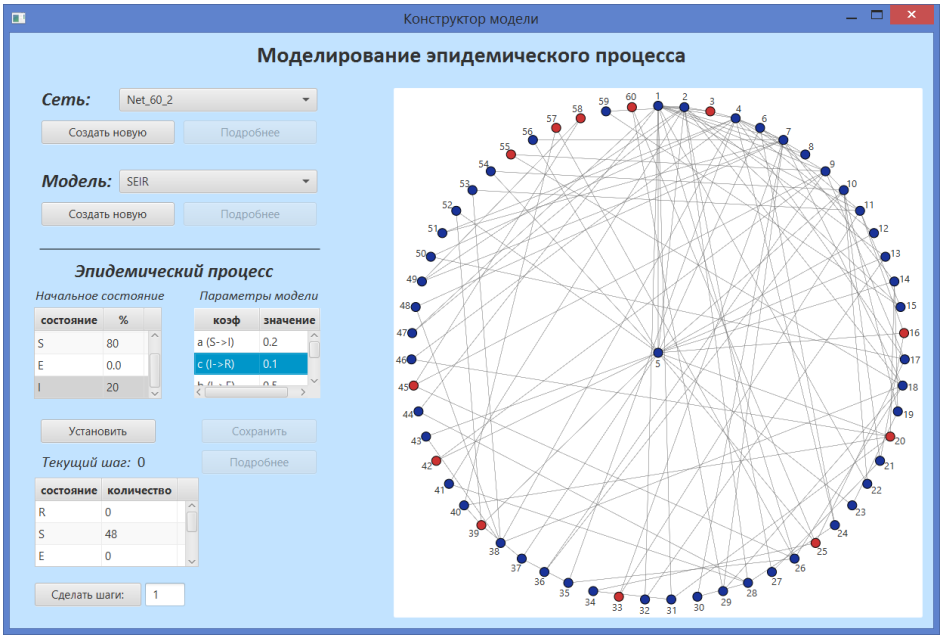


Рис. 13: Начальные параметры процесса: значения коэффициентов.

После того, как все параметры заданы, можно выполнять шаги эпидемического процесса.

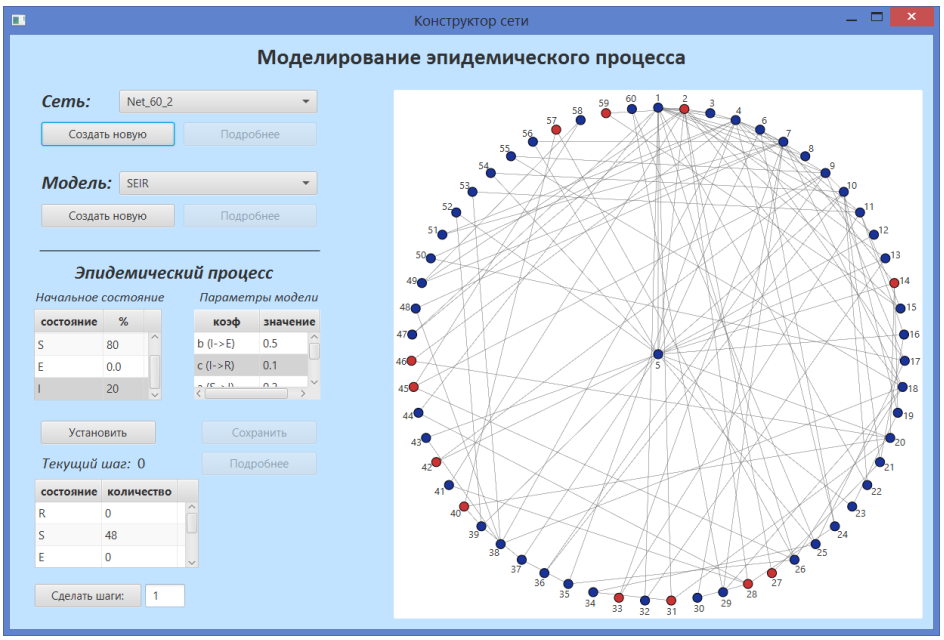


Рис. 14: Начальные параметры процесса: распределение узлов.

Эпидемический процесс. На первом шаге 8 узлов перешло из состояния S в состояние I, еще 6 совершили переход из I в E.

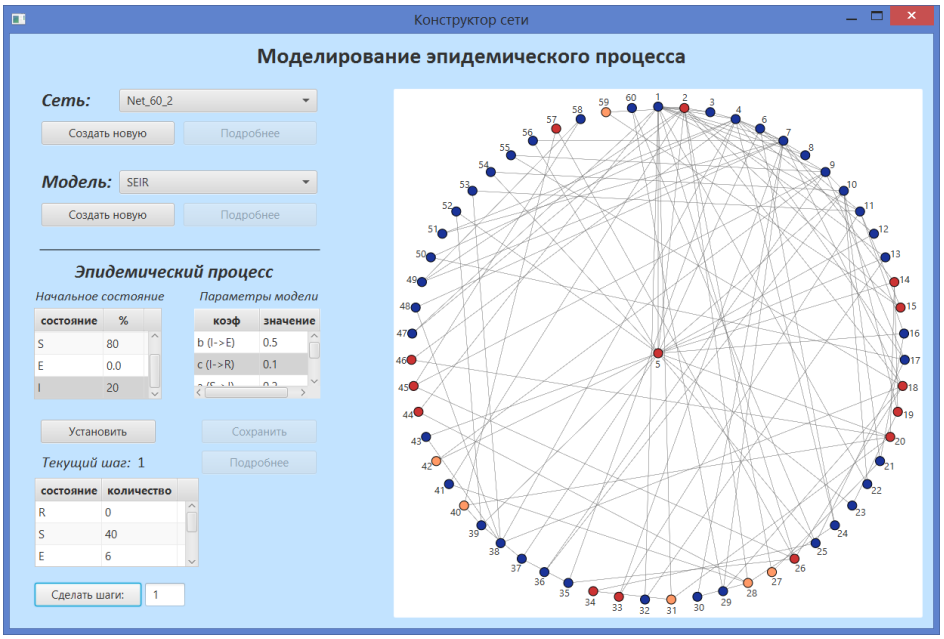


Рис. 15: Шаг 1.

На третьем шаге в состоянии S осталось менее половины узлов, а количество узлов в состоянии E увеличилось на 11.

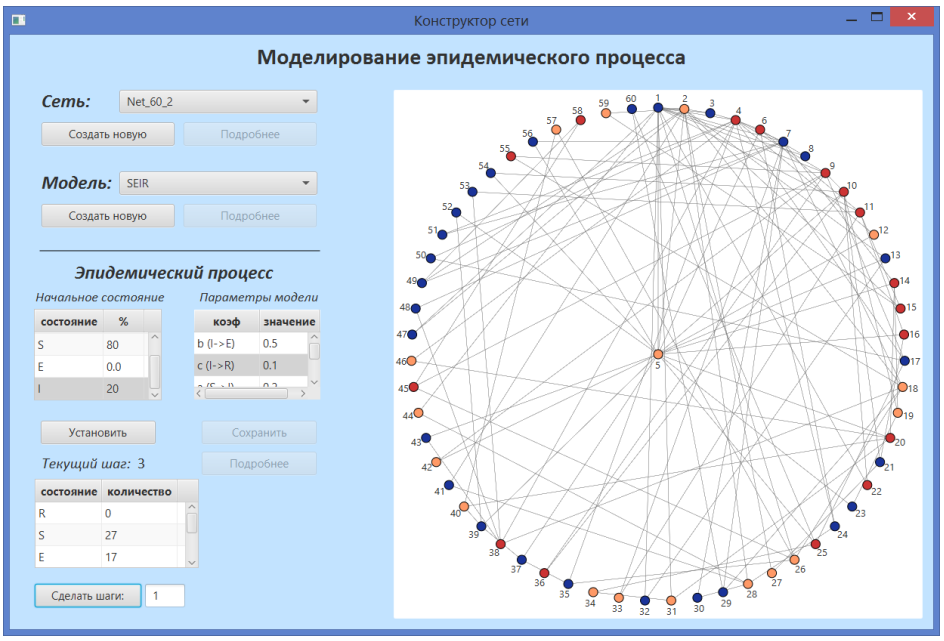


Рис. 16: Шаг 3.

Приведем еще несколько шагов процесса.

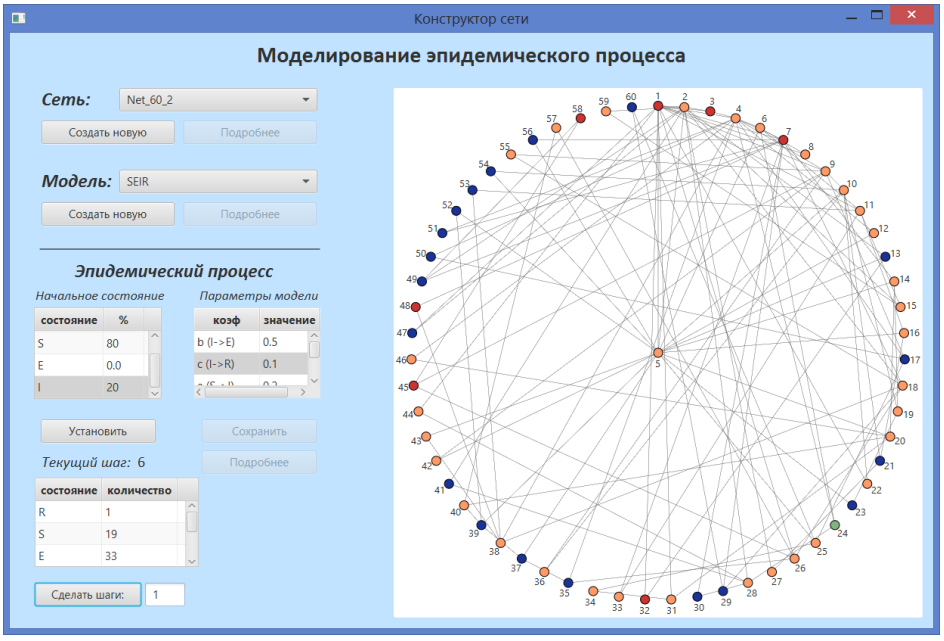


Рис. 17: Шаг 6.

На шаге 10 не осталось инфицированных узлов (в состоянии I). В этом процессе распространение остановилось, вирус не успел инфицировать всех восприимчивых.

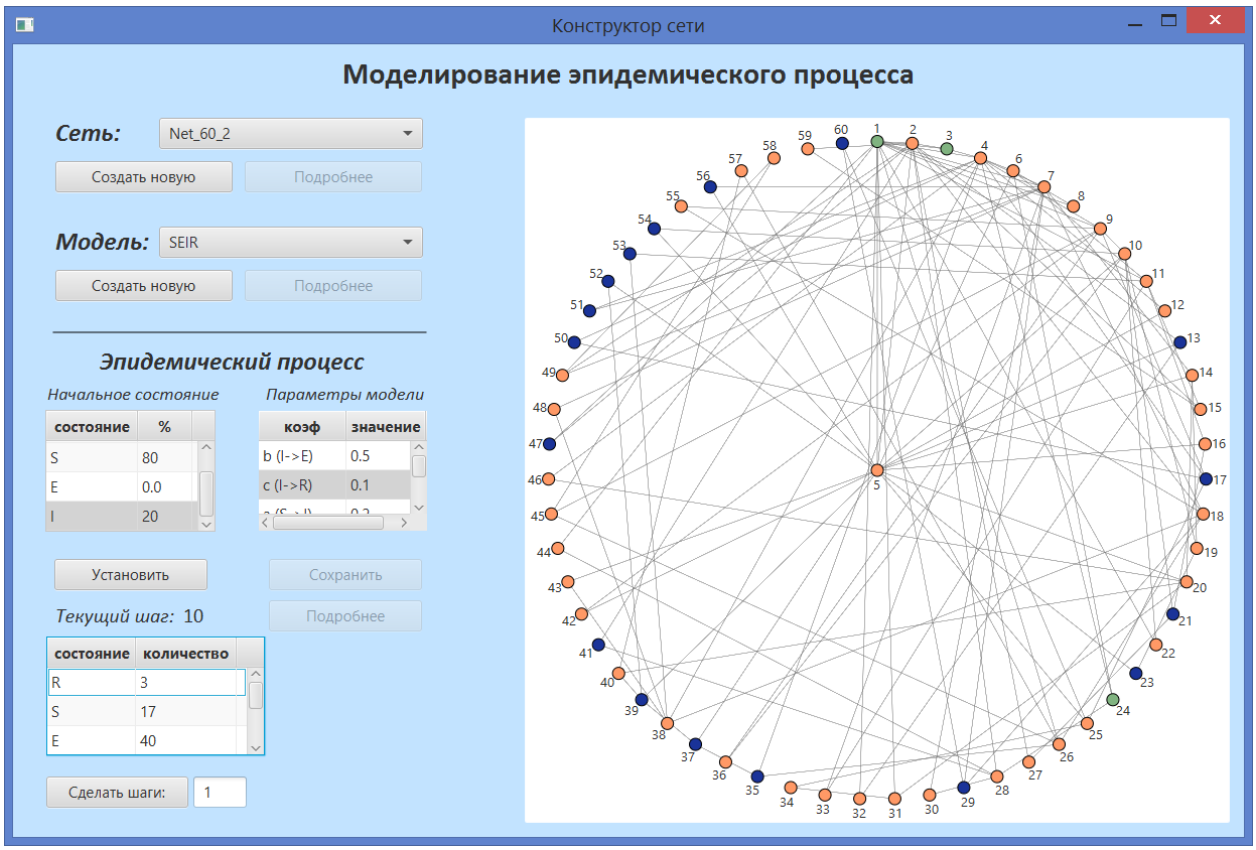


Рис. 18: Шаг 10.

Установив новое начальное состояние, можно сбросить процесс к нулевому шагу и провести новое моделирование на той же сети с использованием той же модели. При этом процесс пройдет по-другому, так как на начальном этапе окажутся зараженными другие узлы, а элемент случайности повлияет на вероятностные переходы.

Заключение

Целью данной работы было создание программного продукта, позволяющего генерировать безмасштабную сеть по заданным параметрам, конструировать эпидемические модели, и совмещать их для воспроизведения эпидемического процесса.

Программа разработана на языке Java с использованием объектно-ориентированной базы данных db4o в качестве хранилища.

Для модуля, отвечающего за создание безмасштабных сетей был использован алгоритм построения безмасштабной сети, разработанный в статье [13]. Модуль конструирования модели и модуль эпидемического процесса создавались с нуля в рамках этой работы.

В работе описывается архитектура программного комплекса и важнейшие детали реализации. В частности описаны важнейшие особенности шага эпидемического процесса на сети и приведены отличия от классической интерпретации.

Так же подробно разобран пример использования программного продукта. Для проведения исследований пользователю не требуется обладание особыми навыками, а в результате возможно наблюдение пошагового распространения эпидемии на выбранной сети по выбранной модели.

Проект разработан с некоторыми ограничениями. Первое касается возможностей отображения сети в читаемом виде и накладывает ограничение на количество узлов. Второе предполагает для определенности, что

между двумя состояниями модели в одном направлении существует только одно уравнение перехода.

Полученный результат протестирован на производительность. Основная доля моделируемых сетей (в рамках отображаемых в читаемом виде) генерируется менее чем за 20 миллисекунд. Самый сложный шаг эпидемического процесса на таких сетях будет расчитан менее чем за 10 миллисекунд.

Таким образом, поставленная цель достигнута в полном объеме.

Литература

1. Erdos P., Renyi A. On random graphs // Publicationes Mathematicae. 1959. P. 290–297.

2. Берновский М. М., Кузюрин Н. Н. Случайные графы, модели и генераторы безмасштабных графов // Труды Института системного программирования РАН. 2012. Т. 22, С. 419–434

3. Райгородский А. М. Модели случайных графов и их применение // Труды МФТИ. 2010. Т. 2, N.4, С. 130–140.

4. Kermack W.O., Mc Kendrick A.G. A contribution to the mathematical theory of epidemics // Proceedings of the Royal Society. 1927. Ser. A. Vol. 115, No A771. P. 700–721.

5. Тайницкий В.А., Губар Е.А., Житкова Е.М. Оптимальное управление в задаче моделирования взаимодействия двух типов вредоносного ПО // XII всероссийское совещание по проблемам управления ВСПУ-2014. Институт проблем управления им. В.А. Трапезникова РАН. 2014. С. 8224–8229.

6. Колесин И.Д. Принципы моделирования социальной самоорганизации: Учебное пособие. СПб.: Изд-во «Лань», 2013. 288 с.

7. Nekovee M., Moreno Y., Bianconi G., Marsili M. Theory of rumour spreading in complex social networks // Physica A 374, 2007. P. 457–470.

8. Wang Ya-Qi, Yang Xiao-Yuan, Han Yi-Liang, Wang Xu-An. Rumor spreading model with trust mechanism in complex social networks // Commun. Theor. Phys. 59. 2013. P. 510–516.

9. Gubar E., Kumacheva S., Zhitkova E., Porokhnyavaya O. Impact of Propagation Information in the Model of Tax Audit // Recent Advances in Game Theory and Applications, Springer International Publishing. 2016. P. 91–110.

10. Pastor-Satorras R., Vespignani A. Epidemic spreading in scale-free networks // Physic review letters. 2001. Vol. 86, No 14. P. 3200 – 3203.

11. Многоподходное имитационное моделирование AnyLogic. https://www.anylogic.ru/

12. Улыбин А. В. Математическая модель распространения инфекции // Вестник Тамбовского университета. Серия: Естественные и технические науки. 2011. Т. 16, С. 184–187

13. Порохнявая О. Ю. Один алгоритм построения безмасштабных сетей // Процессы управления и устойчивость. 2015. Т. 2. С. 702–707.

14. Боев Б. В. Прогнозно-аналитические модели эпидемий (оценка последствий техногенных аварий и природных катастроф) //лекция для слушателей МФТИ, 2005. 10 с.

15. Li L., Alderson D., Tanaka R., Doyle J.C., Willinger W. Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications. Internet Mathematics. 2006. Vol. 2, No 4. P. 431-523.

16. Pastor-Satorras R., Castellano C, Mieghem P.V., Vespignani A. Epidemic processes in complex networks // Reviews of Modern Physic 2015. Vol. 87, P. 925–987.

Приложения

Эта глава содержит основные фрагменты кода разработанного программного продукта.

Класс Network

```
1  private String uid;
2  private ArrayList<Node> nodes;
3  private NetworkData parameters;
4  public String name = "";
5
6  public Network(NetworkData data) {...}
7  public String getUid(){...}
8  public ArrayList<Node> getNodes(){...}
9  public Node getMaxHab() {...}
10 public Node getNodeById(int id) {...}
11 private void addNode (Node node) {...}
12 private void addLink (Node node1, Node node2) {...}
13 private void init() {...}
14 private void calculateLinkedNode(Node node) {...}
```

Создание сети

```
1  private void addNode (Node node) {
2    if (!this.nodes.contains(node)) {
3      this.nodes.add(node);
4      System.out.println("Node " + node.name + " added");
5    }
6  }
7
```

```
8   private void addLink (Node node1, Node node2) {
9     node1.addNeighbor(node2);
10    node2.addNeighbor(node1);
11
12    System.out.println("Nodes " + node1.name + " and " + node2.name + "
          linked");
13  }
14
15  private void init() {
16    NetworkData data = this.parameters;
17
18    //создаем набор несвязанных узлов
19    for (int i = 0; i < data.startSize; i++) {
20      Node node = new Node();
21      this.addNode(node);
22    }
23
24    //если количество связей нового узла больше чем существующих узлов,
          связываем со всеми
25    int allConnect = data.newNodeLinks - data.startSize;
26    for (int i = 0; i < allConnect; i++) {
27      Node newNode = new Node();
28      this.addNode(newNode);
29
30      for (Node oldNode : this.nodes) {
31        if (!oldNode.equals(newNode)) {
32          this.addLink(newNode, oldNode);
33        }
34      }
35    }
36
37    //оставшиеся узлы появляются с data.newNodeLinks связями, которые
          раздаются существующим узлам
38    for (int i = data.startSize + allConnect; i < data.size; i++) {
39      Node newNode = new Node();
40      this.addNode(newNode);
```

```
41
42          for (int linkCounter = 0; linkCounter < data.newNodeLinks;
                linkCounter++) {
43            this.calculateLinkedNode(newNode);
44          }
45        }
46
47      }
48
49      private void calculateLinkedNode(Node node) {
50        HashMap<Node, Integer> linksMap = new HashMap<>();
51        int allLinks = 0;
52
53        //ищем узлы сети, несвязанные с текущим, сохраняем количество их
              связей
54        for (Node networkNode : this.nodes) {
55          if (!networkNode.equals(node) && !networkNode.hasNeighbor(node))
                {
56            int links = networkNode.getNeighbors().size();
57            allLinks += links;
58            linksMap.put(networkNode, links);
59          }
60        }
61
62
63        int randValue = this.parameters.random.nextInt(allLinks) + 1;
64        int curValue = 0;
65
66        //интерпретируем значение рандома, чем больше связей у узла, тем
              выше вероятность выбрать его
67        for (Map.Entry<Node, Integer> entry : linksMap.entrySet()) {
68          curValue += entry.getValue();
69
70          if (randValue <= curValue) {
71            Node networkNode = entry.getKey();
72            this.addLink(networkNode, node);
```

```
73            break;
74          }
75        }
76      }
```

Класс Model

```
1   private String uid;
2   private LinkedList<State> states;
3   private LinkedList<StateTransition> transitions;
4   public String name = "";
5
6   public Model() {...}
7   public boolean addState(State newState) {...}
8   public boolean removeState(String stateName) {...}
9
10  public boolean addTransition(String startStateName, String
        endStateName, boolean linkRequired,
11  String coefName, Double coefValue) {...}
12
13  public boolean removeTransition(String startStateName, String
        endStateName) {...}
14
15  public State findState(String stateName) {...}
16
17  public StateTransition findTransition(String startName, String
        endName) {...}
18
19  public LinkedList<State> getStates() {...}
20  public LinkedList<String> getStateNames() {...}
21  public LinkedList<StateTransition> getTransitions() {...}
```

Переходы

```
1   private State startState;
2   private State endState;
```

```
3   private boolean linkRequired;
4   private String coefName;
5
6   public StateTransition(State startState, State endState, boolean
        linkRequired, String coefName) {...}
7
8   public void setCoefName(String value) {...}
9   public String getCoefName() {...}
10  public State getStartState() {...}
11  public State getEndState() {...}
12  public boolean isLinkRequired() {...}
13  public boolean pathEquals(State start, State end) {...}
14  public boolean pathEquals(String start, String end) {...}
15  public boolean pathEquals(StateTransition transition) {...}
16  public boolean equals(StateTransition transition){...}
```

Класс Epidemic

```
1   private Model model;
2   private Network network;
3   private ArrayList<Step> steps;
4   private Random rand;
5   private HashMap<StateTransition, Double> transitionCoefs;
6
7   public Epidemic(Model model, Network network) {...}
8
9   public void setModelCoefs(HashMap<StateTransition, Double> coefsMap)
        {...}
10
11  public boolean generateInitialStep(HashMap<String, Long> initialState
        ){...}
12
13  public Step getLastStep() {...}
14  public HashMap<String, Integer> getLastStepMap() {...}
15  public Step getStepByIndex(int ind) {...}
16  public int getProcessLength() {...}
```

```
17    public void clearProcess() {...}
18    public void makeSteps(int amount) {...}
19    private void makeStep() {...}
```

Инициализация и рассчет шага

```
1  public Epidemic(Model model, Network network) {
2    this.model = model;
3    this.network = network;
4    steps = new ArrayList<>();
5    transitionCoefs = new HashMap<>();
6    rand = new Random();
7
8    Step initial = new Step(network.getNodes());
9    steps.add(initial);
10   generateInitialStep(null);
11
12   for (State state : model.getStates()) {
13     for (StateTransition transition : state.getOutcomeTransitions()) {
14       if (!transitionCoefs.containsKey(transition)) {
15         transitionCoefs.put(transition, 0.0);
16       }
17     }
18   }
19 }
20
21 private void makeStep() {
22   Step lastStep = steps.get(steps.size()-1);
23   Step newStep = new Step(lastStep);
24
25   System.out.println("STEP " + steps.size());
26
27   LinkedList<State> states = model.getStates();
28   for (State state : states) {
29     LinkedList<StateTransition> outcome = state.getOutcomeTransitions()
         ;
```

```
30      if (outcome.size() > 0) {
31        ArrayList<Node> nodes = lastStep.getNodesAtState(state);
32        ArrayList<Double> probabilities;
33        for (Node node : nodes) {
34          probabilities = new ArrayList<>();
35          for (StateTransition transition : outcome) {
36            System.out.println("Node " + node.getId() + " from " +
                  transition.getStartState().getName() +
37            " to " + transition.getEndState().getName() + "?");
38            if (transition.isLinkRequired()) {
39              ArrayList<Node> nodesAtState = lastStep.getNodesAtState(
                    transition.getEndState());
40              System.out.println("  has " + node.getNeighbors().size() +
                    " neighbors");
41              int count = 0;
42              for (int neighborId : node.getNeighbors()) {
43                Node neighbor = network.getNodeById(neighborId);
44                if (nodesAtState.contains(neighbor)) {
45                  count++;
46                }
47              }
48              System.out.println("  " + count + " neighbors at state " +
                    transition.getEndState().getName());
49              double resultProbability = 1 - (Math.pow(1 -
                    transitionCoefs.get(transition), count));
50              System.out.println("  result probability = " +
                    resultProbability);
51              probabilities.add(resultProbability);
52
53
54
55            } else {
56              probabilities.add(transitionCoefs.get(transition));
57              System.out.println("  result probability = " +
                    transitionCoefs.get(transition));
58            }
```

```
59          }
60          double r = rand.nextDouble();
61          double sum = 0;
62          int i = 0;
63          while (r >= sum && i < probabilities.size()) {
64            sum += probabilities.get(i);
65            i++;
66          }
67
68          System.out.println("  rand = " + r);
69          if (r >= sum) System.out.println("no transition");
70          else {
71            StateTransition transition = outcome.get(i-1);
72            newStep.setNodeState(node, transition.getEndState());
73            System.out.println("transition " + transition.getStartState()
                .getName() + " => "
74            + transition.getEndState().getName());
75          }
76        }
77      }
78    }
79
80    steps.add(newStep);
81  }
```