

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**ИССЛЕДОВАНИЕ СЛУЧАЙНЫХ ОРГРАФОВ СТАТИЧЕСКИХ КОМПЬЮТЕРНЫХ AD HOC СЕТЕЙ**

Работу выполнил Н. А. Крючков

СОДЕРЖАНИЕ

Введение ….……..……………………………………………………………... 4

1. Постановка задачи ………………………………………………………...... 5
2. Общие сведения об ad hoc сетях …………………………………………... 6
   1. Определения ……………...………………………………………....... 6
   2. Разновидности ad hoc сетей …………………………………….…… 7
      1. Мобильные ad hoc сети ………………………………………… 8
      2. Беспроводные ячеистые ad hoc сети ………………………….. 9
      3. Беспроводные сенсорные ad hoc сети ….……………………... 9
      4. Ad hoc сети для транспортных средств ……………………….. 10
   3. Технологии связи в ad hoc ..………………………………………….. 11
      1. Wi-Fi ……………………………………………………………... 12
      2. ZigBee ……………………………………………………………. 13
      3. Bluetooth …………………………………………………………. 13
   4. Построение ad hoc сети в домашних условиях …………………….. 15
      1. Создание сети …………………………………………………... 15
      2. Добавление других компьютеров в сеть ……………………… 16
   5. Маршрутизация в мобильных ad hoc сетях ………………………… 17
   6. Протоколы маршрутизации в ad hoc сетях …………………………. 21
      1. Протокол AODV ………………………………………………… 21
      2. Протокол DSR ………………………………………………….... 24
      3. Протокол OSLR ………………………………………………… 26
      4. Протокол ZPR …………………………………………………… 28
   7. Эмуляция ad hoc сетей ……………………………………………….. 29
      1. NS2 ………………………………………………………………. 30
      2. NS3 ………………………………………………………………. 31
      3. OmNET++ ……………………………………………………….. 32
3. Общие сведения о графах …………………………………………............ 33
4. Практическая реализация ………………….…………….……………….. 37
   1. Представление ad hoc сети и оценка сложности алгоритмов ………. 37
   2. Язык Python ………….…………………………..……………………. 37

4.3 Библиотека matplotlib ………………………………………… 38

* 1. Интегрированная среда разработки PyCharm …………………........... 38
  2. Структура программы ………………………………………………… 39

4.5.1 Основные методы программы …………………………………... 39

4.5.2. Описание работы программы ……………....... 40

4.5.3. Результат работы программы ………………………………........ 45

Заключение …………………………………………………………………...... 58

Список использованной литературы ……………………................................. 59

Приложение А Программный код ……..…………………………………60

ВВЕДЕНИЕ

В последние несколько лет широкое распространение получили ad hoc сети, главной чертой которых является самоорганизация. Мобильные узлы имеют возможность присоединяться к сети и отсоединяться от ее, менять положение в ней, вследствие чего топология будет изменяться постоянно непредсказуемым образом. Узлы сети могут являться не только оконечным пользовательским терминалом, но и ретранслятором-маршрутизатором, передавая при этом принадлежащие другим абонентам пакеты и непосредственно участвуя в процессе маршрутизации.

Для развертывания самоорганизующихся сетей достаточно одной или нескольких точек доступа в отличие от «традиционных» беспроводных сетей, в случае которых обычно разворачивается дорогостоящая инфраструктура базовых станций. Самоорганизующиеся сети связи — сети, имеющие изменяемую децентрализованную инфраструктуру. В общем случае такие сети имеют следующие преимущества: широкое покрытие, широкая абонентская база без использования большого количества дорогостоящих базовых станций и увеличения мощности излучаемого сигнала.

На данный момент идут активные исследования и применения самоорганизующихся сетей в следующих сферах:

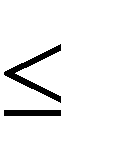
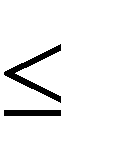
* интеллектуальные транспортные системы;
* военная связь;
* сенсорные сети; - локальные сети.

# 1 Постановка задачи

В выпускной квалификационной работе должны быть разработаны алгоритмы и программы для исследования, а также проведены исследования того, какие именно ориентированные графы (с точки зрения теории графов) соответствуют возможным конфигурациям ad hoc сетей. При этом анализируется не одна конфигурация сети, а множество различных конфигураций, рассматриваемых как реализации случайной сети – случайный геометрический ориентированный граф.

Результаты исследования должны быть представлены в виде формул или графиков, показывающих следующие зависимости для заданных исходных данных.

Областью расположения *S* сети является круг радиуса *M* = 5. Количество узлов в сети находится в пределах 2 *n* 10. Положение узлов в области *S* случайно, взаимно независимо, равномерно распределено. Для радиусов *R* узлов необходимо рассмотреть два случая расположения:

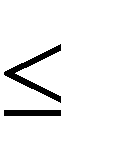
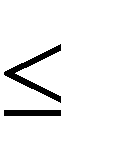


- случайное, взаимно независимое, равномерно распределенное на

[1 – *c*, 1 + *c*] со средним значением 1,0;

- постоянное *R* = 1 (для сравнительных оценок).

Анализ должен проводиться для различных значений 0,2 *c* 0,8 с шагом 0,1.



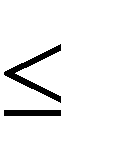
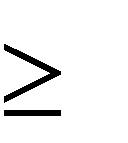
# 2 Общие сведения об ad hoc сетях

### 2.1 Определения

Компьютерные ad hoc сети – это беспроводные децентрализованные сети с *n* узлами, расположенными в некоторой ограниченной или неограниченной области *S*. Каждый узел характеризуется координатами и радиусом действия передатчика сигналов. Математической моделью такой сети является так называемый *геометрический граф*.

Математическая модель расположения на плоскости ad hoc сети представляет собой объединение кругов заданных радиусов. В области расположения сети имеется естественный электромагнитный шум некоторого уровня. В центре одного из кругов (*a*) находится передатчик сигналов. Уровень электромагнитного поля, создаваемого передатчиком, уменьшается при удалении от центра круга. Радиусом *Ra* считается такое значение удаления, при котором отношение сигнал/шум соответствует чувствительности стандартного приемника, находящегося на этом удалении. Иначе говоря, при меньшем, чем *Ra* , удалении приемник способен принять и распознать сигнал, а при большем или равном расстоянии – нет (либо вообще не принимает сигнал, либо не может его декодировать и считает шумом). Таким образом, те узлы сети (вершины графа), которые попали в круг, способны «слушать», что может быть обозначается дугой, идущей из центра круга *a* к «слушающей» вершине.

Слушающая вершина *b* также снабжена передатчиком, и также создает круг радиуса *Rb* с центром в этой вершине. Вершины, попадающие в ее круг, способны ее слышать.

Вершины *a* и *b* слышат друг друга, т.е. могут непосредственно обмениваться информацией, тогда и только тогда, когда расстояние между ними *d*(*a*, *b*) < min(*Ra* , *Rb*). Если *d*(*a*, *b*) max(*Ra* , *Rb*), то связи нет. Если min(*Ra* , *Rb*) *d*(*a*, *b*) < max(*Ra* , *Rb*), то связь односторонняя. Таким образом, граф компьютерной сети является ориентированным (орграф).

На рисунке 1 изображен пример ad hoc сети.

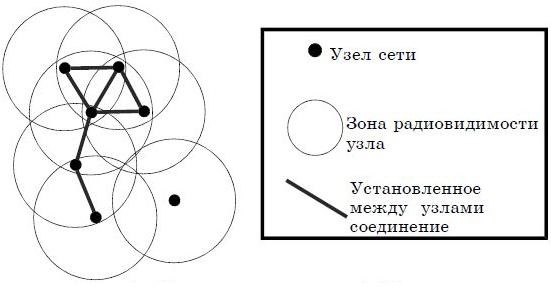


Рисунок 1 — Пример ad hoc сети

Необходимым условием функционирования сети является связность (понимаемая так, как в теории графов). Для ориентированных графов имеется три уровня связности. Орграф сильно связный, если все его вершины взаимно достижимы (т.е. существует маршрут с учетом направления дуг из одной вершины в другую); односторонне связный, если для любых двух вершин, по крайней мере, одна достижима из другой; слабо связный, если при игнорировании направления дуг получается связный (мульти)граф. В противном случае граф несвязный.

Максимальный сильный подграф называется сильной компонентой; односторонняя компонента и слабая компонента определяются аналогично.

### 2.2 Разновидности ad hoc сетей

Самостоятельный характер сетей ad hoc делает их полезными в таких ситуациях, как стихийные бедствия, экстренные военные операции или даже быстрая передача информации между двумя компьютерами дома. Однако, несмотря на простоту использования и масштабируемость, в практическом мире существуют физические и эксплуатационные ограничения в использовании самоорганизующихся сетей. По этой причине ad hoc сети имеют несколько реализаций, используемых в различных ситуациях:

###### 2.2.1 Мобильные ad hoc сети

Мобильные сети ad hoc (MANET) представляют собой сеть

самообучения мобильных устройств, подключенных по беспроводной сети. Такие сети могут менять местоположение и настраивать себя на лету. Некоторые MANET ограничены локальной областью беспроводных устройств (например, группой ноутбуков), а другие могут быть подключены к Интернету. На рисунке 2 представлен пример мобильной ad hoc сети.



Рисунок 2 — Пример мобильной ad hoc сети

###### 2.2.2 Беспроводные ячеистые сети ad hoc

Беспроводная ячеистая сеть (WMN) представляет собой сеть связи радиоузлов, структурированных в сетчатой топологии. Клиентами в сети обычно являются ноутбуки, мобильные телефоны и другое беспроводное оборудование. Сетевая сеть с помощью маршрутизаторов и шлюзов передает данные на беспроводные устройства и обратно. Связь находится в сетке, а не в Интернете. На рисунке 3 изображен пример ячеистой ad hoc сети.

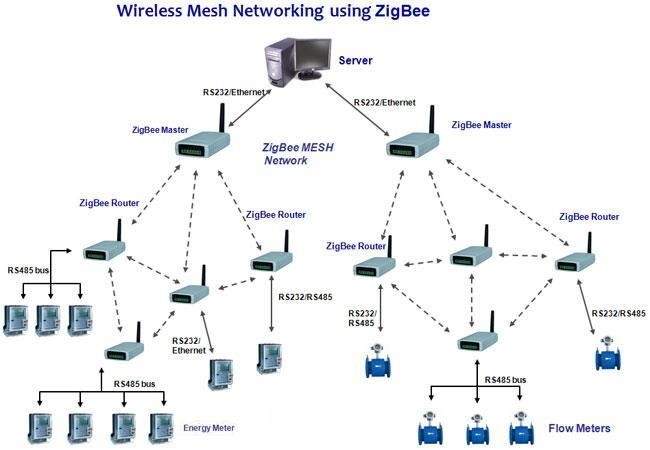


Рисунок 3 — Пример ячеистой беспроводной ad hoc сети

###### 2.2.3 Беспроводные сенсорные ad hoc сети

Беспроводная сенсорная сеть (WSN) использует сенсорные устройства для совместного наблюдения физических или экологических параметров, таких как звук, давление, климатические изменения и т. д. Беспроводные сенсорные сети используются в широком диапазоне областей: управление трафиком, обнаружение транспортных средств, мониторинг теплиц и т. п.

Рисунок 4 является примером беспроводной сенсорной сети.

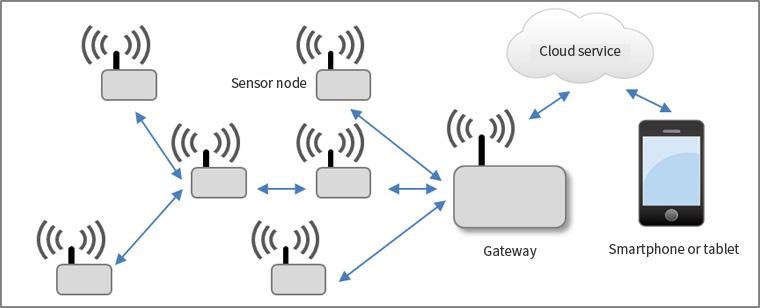


Рисунок 4 — Пример беспроводной сенсорной ad hoc сети

###### 2.2.4 Ad hoc сети для транспортных средств

Автотранспортная сеть ad hoc или VANET — это технология, которая использует двигатели в качестве узлов в сети для создания мобильной сети. VANET превращает каждый участвующий автомобиль в беспроводной маршрутизатор или узел, позволяя автомобилю приблизительно 100-300 метров друг от друга подключаться и, в свою очередь, создавать сеть с широким диапазоном. Поскольку автомобили выпадают из диапазона сигналов и выходят из сети, к ним могут присоединяться другие автомобили, соединяющие транспортные средства друг с другом, чтобы создать мобильный Интернет. По оценкам, первые системы, которые будут интегрировать эту технологию, — это полицейские и пожарные транспортные средства для связи друг с другом в целях безопасности.

Пример подобной сети изображен на рисунке 5.

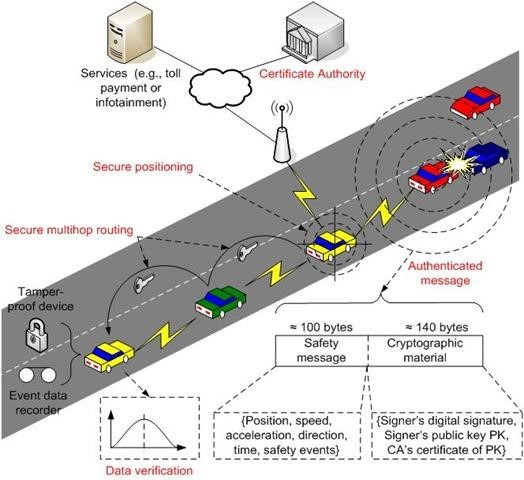


Рисунок 5 — Пример ad hoc сети для транспортных средств

##### 2.3 Технологии связи в ad hoc сетях

На данный момент существует несколько технологий связи самоорганизующихся сетей:

* WiFi 802.11;
* ZigBee;
* Bluetooth.

###### 2.3.1 Wi-Fi

Сети стандарта 802.11 изначально были задуманы как способ замены проводных сетей. Однако, относительно высокие скорости передачи (до 108 Мбит/с) делают перспективным возможное применение в тех самоорганизующихся сетях, в которых необходимо передавать большие объемы информации в реальном времени (например, видеосигнала).

В 2007 году впервые была выпущена черновая версия стандарта 802.11s, определяющего основные характеристики самоорганизующихся сетей на основе Wi-Fi.

В отличии от традиционных сетей Wi-Fi, в которых существует только два типа устройств — «точка доступа» и «терминал», стандарт 802.11s предполагает наличие так называемых «узлов сети» и «порталов сети». Узлы могут взаимодействовать друг с другом и поддерживать различные службы. Узлы могут быть совмещены с точками доступа, порталы же служат для соединения с внешними сетями.

На основе уже существующих стандартов 802.11 можно строить MANET-сети (мобильные самоорганизующиеся сети), отличительной чертой которых можно назвать большую зону покрытия (несколько квадратных километров).

Проблемы, требующие особого внимания при дальнейшем развитии самоорганизующихся сетей на базе Wi-Fi можно разделить на следующие классы:

* проблемы пропускной способности;
* проблемы масштабируемости сетей.

###### 2.3.2 ZigBee

Стандарт 802.15.4 (ZigBee) описывает низкоскоростные сети связи малого радиуса действия с маломощными передающими устройствами.

Предусмотрено использование трех диапазонов частот: 868-868.6 МГц, 902928 МГц, 2.4-2.4835 ГГц.

В качестве метода доступа к каналу используется DSSS с различными длинами последовательности для диапазонов 868/915 и 2450 МГц.

Скорости передачи данных варьируются от 20 до 250 кбит/с.

Согласно стандарту, сеть ZigBee поддерживает работу с топологиями типа «звезда» и «каждый с каждым».

Существуют два варианта приемопередающих устройств:

полнофункциональные (FFD) и неполнофункциональные (RFD). Коренное отличие этих устройств состоит в том, что FFD могут устанавливать прямую связь с любыми устройствами, а RFD — только с FFD.

Сеть ZigBee может состоять из нескольких кластеров, образованных устройствами FFD.

Сети стандарта ZigBee могут работать в режиме mesh. При этом предполагается, что каждый узел сети (узел сети образует устройство FFD, RFD работают в качестве т.н. сенсоров) постоянно следит за состоянием соседних узлов, обновляя при необходимости свои таблицы маршрутизации.

В отличие от всех предыдущих вариантов сетей ad hoc, технология ZigBee рассчитана на низкие скорости передачи данных, и проблемы

возможности ее увеличения не существует.

###### 2.3.3 Bluetooth

Самоорганизующиеся сети на основе Bluetooth состоят из ведущих и ведомых устройств (эти роли могут совмещаться), способных передавать данные как в синхронном, так и в асинхронном режимах. Синхронный режим передачи предполагает прямую связь между ведущим и ведомым устройствами с закрепленным каналом и временными слотами доступа. Данный режим используется в случае ограниченных по времени передач. Асинхронный режим предполагает обмен данными между ведущим и несколькими ведомыми устройствами с использованием пакетной передачи данных. Используется для организации пикосетей. Одно устройство (как ведущее, так и ведомое) может поддерживать до 3-х синхронных соединений.

В синхронном режиме максимальная скорость передачи данных равна 64 кбит/с. Максимальная скорость передачи в асинхронном режиме составляем 720 кбит/с.

Достоинства сетей на базе Bluetooth:

* возможность быстрого развертывания;
* сравнительно малое энергопотребление абонентских устройств; - широкий спектр поддерживающих эту технологию устройств.

Недостатки сети:

* небольшой радиус действия (радиус действия одного абонентского устройства составляет 0.1 — 100 м);
* малые скорости передачи данных (для сравнения: в сетях Wi-Fi этот показатель составляет 11 — 108 Мбит/с); - нехватка частотного ресурса.

Возможно, последняя проблема будет решена с выходом устройств Bluetooth 3.0, где предполагается возможность использовать альтернативные протоколы уровней MAC и физического с целью ускоренной передачи данных профилей Bluetooth (AMP). В частности, могут быть использованы протоколы стандарта 802.11.

Исходя из вышеприведенного, можно заключить, что сети на основе Bluetooth применимы лишь в местах большого скопления людей (например, в центрах городов, небольших офисах, магазинах). Например, подобная сеть может служить для организации видеонаблюдения на небольшом объекте.

Сравнение технологий Wi-Fi, Bluetooth и ZigBee представлено на рисунке 6.

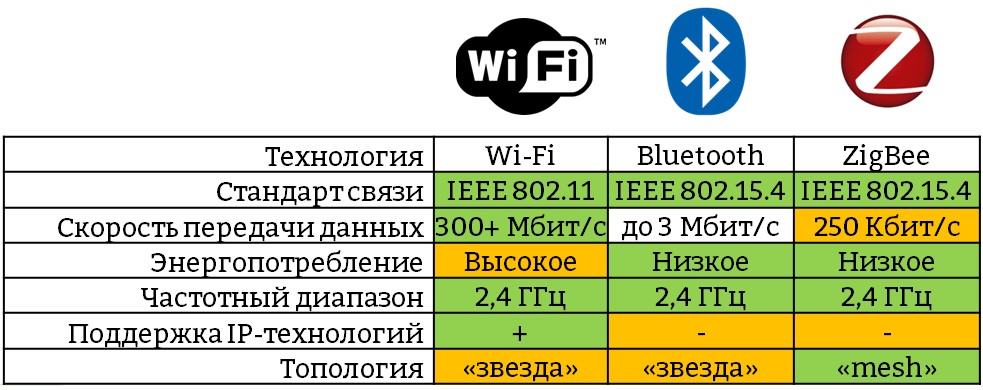


Рисунок 6 — Сравнение технологий WiFi, Bluetooth, ZigBee

##### 2.4 Построение ad hoc сети в домашних условиях

###### 2.4.1 Создание сети

Чтобы установить соединение ad hoc, необходимо иметь два или более персональных компьютера или ноутбука с сетевой картой, поддерживающей беспроводные сети. Чтобы установить подключение ad hoc на домашнем компьютере, требуется выполнить следующие действия:

1. Нажмите «Пуск», а затем «Панель управления».
2. Дважды щелкните значок «Сетевые подключения».
3. Щелкните правой кнопкой мыши беспроводное сетевое соединение и выберите «Свойства».
4. Появится окно «Свойства беспроводной сети». Перейдите на вкладку «Беспроводная сеть».
5. Нажмите кнопку «Добавить», которая находится в разделе «Предпочтительные сети».
6. Появится окно «Свойства беспроводной сети».
7. Введите имя своей сети в текстовое поле «Имя сети» (SSID), представленное под вкладкой «Ассоциация».
8. Убедитесь, что флаги «Сгенерировать ключ автоматически» и «Это сеть компьютер-компьютер (ad hoc)» установлены.
9. Нажмите «ОК», чтобы сохранить внесенные изменения. Теперь сеть Ad hoc настроена.

Необходимо обратить внимание на то, что для обеспечения безопасности сети ad hoc, можно убрать флаг «Сгенерировать ключ автоматически» и указать свой собственный 13-значный пароль.

###### 2.4.2 Добавление других компьютеров в сеть

Чтобы подключить другой компьютер к сети ad hoc, необходимо выполнить следующие действия:

1. В правом нижнем углу экрана вашего рабочего стола щелкните правой кнопкой мыши значок «Беспроводная сеть» и выберите «Просмотреть доступные беспроводные сети».
2. Появится окно подключения к беспроводной сети. Оно отобразит недавно созданную специальную сеть с указанным именем SSID. Выделите ваше соединение в списке, щелкнув его, а затем нажмите «Подключиться».
3. Введите ключ шифрования в текстовых полях «Сетевой ключ» и «Подтвердить сетевой ключ» и нажмите «Подключить», если вы выбрали пароль ранее.
4. Когда соединение будет установлено, отобразится индикатор выполнения. По завершении подключения вы можете выйти из окна

«Беспроводное сетевое подключение».

##### 2.5 Маршрутизация в мобильных ad hoc сетях

Сети MANET (Mobile Ad-hoc NETworks) – это радиосети со случайными мобильными абонентами, реализующие полностью децентрализованное управление при отсутствии базовых станций или опорных узлов. Топология MANET динамична и соединение узлов является случайным. Беспроводные сети, построенные на базе мобильных устройств, обладают рядом особенностей:

* мобильность узлов ведет к дополнительному повышению динамичности топологии сети, поскольку, вследствие помех или включения/выключения узла, к возможности обрыва связи добавляется вероятность перемещения узлов;
* запас источников питания мобильных узлов может быть ограничен, в связи, с чем при проектировании аппаратных средств и протоколов необходимо учитывать энергопотребление (особенно для сенсорных сетей).

Самоорганизующиеся сети MANET обладают следующими преимуществами над беспроводными сетями традиционной архитектуры:

* возможность передачи данных на большие расстояния без увеличения мощности передатчика;
* устойчивость к изменениям в инфраструктуре сети;
* возможность быстрой реконфигурации в условиях неблагоприятной помеховой обстановки;
* простота и высокая скорость развертывания.

В сетях MANET каждый узел может независимо перемещаться в произвольном направлении вследствие чего изменения в топологии сети должны быть переданы другим узлам для поддержания правильной маршрутизации. Например, когда узел MH2, указанный на рисунке 7, вследствие перемещения изменяет своё соединение с MH3 на соединение с узлом MH4, другие узлы сети должны получить информацию о новом маршруте от источника до пункта назначения через промежуточные узлы для пересылки пакетов между MH2 и МН3.

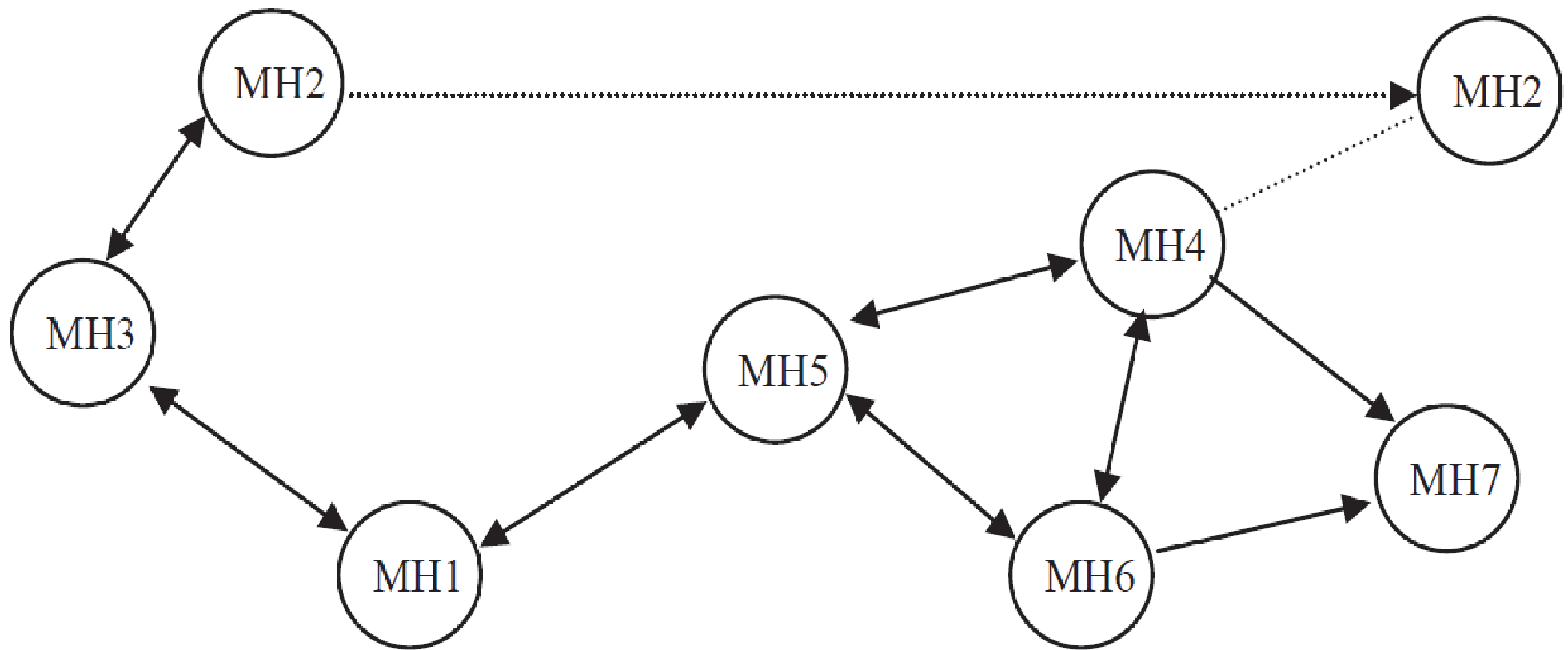


Рисунок 7 — MANET сеть

Таким образом, требования к маршрутизации в MANET сетях существенно отличаются от требований к маршрутизации в инфраструктурных сетях, из-за влияния дополнительных факторов: непрерывное изменение топологии сети из-за перемещения узлов или условий распространения сигнала, ограничения зоны радиовидимости и полосы пропускания радиоканала, лимитированный ресурс источников питания беспроводных узлов и т.д. С учётом этого протоколы маршрутизации для MANET сетей должны обладать следующими свойствами:

* малым временем построения маршрута в условиях постоянно изменяющейся топологии сети;
* высокой надёжностью доставки пакетов в условиях постоянно изменяющихся межузловых связей;
* минимальным объёмом передаваемой служебной информации;
* не допускать формирования петель в маршрутах и иметь механизмы оперативного обнаружения и восстановления разорванных маршрутов;
* обеспечивать высокую масштабируемость сети при сохранении заданной производительности;
* поддерживать требуемый уровень качества обслуживания (QoS).

В соответствие с механизмами формирования таблиц маршрутизации в известных протоколах маршрутизации для Ad-hoc-сетей могут быть выделены три типа протоколов: проактивные (табличные), реактивные (по требованию) и гибридные (комбинированные) (рисунок 8).

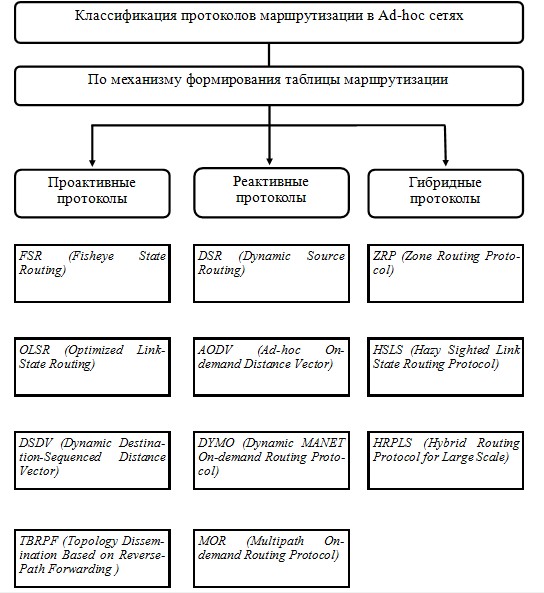


Рисунок 8 — Классификация протоколов маршрутизации

Проактивные или табличные (proactive, table-driven) протоколы строят маршруты на основе периодически рассылаемых во всех направлениях служебных сообщений с информацией о текущей топологии сети, в соответствии с которой каждый узел выбирает маршруты до всех остальных узлов и сохраняет их в таблице маршрутизации. Проактивные протоколы отличаются минимальной задержкой отправки пакетов узлами, так как используют имеющиеся таблицы маршрутизации, но загружают значительную часть пропускной способности сети для поддержания текущей маршрутной информации.

Реактивные или работающие по требованию (reactive, on-demand) протоколы формируют маршруты, используя механизм рассылки узламиотправителями широковещательных запросов, приняв которые узлыадресаты передают сообщения-подтверждения, на основе которых узелотправитель строит требуемый маршрут, одновременно сохраняя его в своей таблице маршрутизации для повторных передач пакетов данному адресату. В случае обнаружения разрыва маршрута запускается процедура поиска маршрута до требуемого адресата. Реактивные требуют существенно меньшей пропускной способности сети, чем проактивные, но вносят значительную задержку на стадии определения маршрутов между конкретными узлами.

В гибридных (hybrid) протоколах комбинируются механизмы формирования таблиц маршрутизации, свойственные проактивным и реактивным протоколам. В частности, в сети выделяется ряд подсетей, в пределах которых используется один из типов проактивных протоколов, а маршрутизация между подсетями производится на основе реактивных протоколов. Такой подход сокращает размеры таблиц маршрутизации узлов внутри соответствующих подсетей и уменьшает объёмы текущей служебной информации, так как основная её часть циркулирует в пределах подсетей.

Динамический характер MANET-сетей определяет частое изменение топологии сети в связи с чем для поиска и поддержания маршрутов требуется значительный объём управляющей информации в общем трафике обмена данными между узлами сети. В основе выбора оптимального маршрута следования пакетов к адресату в большинстве современных протоколов маршрутизации лежат следующие критерии и механизмы:

* наименьшее число ретрансляций пакета (хопов) между

отправителем и адресатом (протоколы выбора вектора расстояния);

* комплексная оценка маршрутов по нескольким параметрам, например, минимальное число хопов, задержка на доставку пакетов, пропускная способность канала и др. (протоколы со сложной метрикой);
* информация о географических координатах и расположении всех узлов сети, получаемая, например, с помощью глобальной системы позиционирования GPS (протоколы географической маршрутизации).

##### 2.6 Протоколы маршрутизации в ad hoc сетях

###### 2.6.1 Протокол AODV

AODV (Ad hoc On-Demand Distance Vector) – специализированный протокол вектора расстояния по запросу. Узел не выполняет поиск маршрута или его поддержку до тех пор, пока ему не понадобится маршрут к другому узлу или пока он не предложит свои услуги в качестве промежуточного узла. Узлы, находящиеся в стороне активного маршрута, не поддерживают информацию о маршрутизации и не участвуют в формировании таблицы маршрутизации. AODV использует маршрутизацию от узла к узлу. Маршрутизация основана на динамической таблице, поддерживаемой в промежуточных узлах. Протокол аналогичен DSR, но DSR использует

маршрутизацию источника.

Для определения местных соединений используются локальные сообщения «hello». Таким образом можно уменьшить время отклика на запросы маршрута и инициировать обновление по мере необходимости.

Маршрутам и записям в таблице маршрутизации присваиваются последовательные номера, которые используются для замещения устаревшей информации. Каждый узел поддерживает два счетчика: последовательный счетчик узла и широковещательный ID.

Запрос маршрута AODV инициируется в том случае, когда узел хочет соединиться с другим узлом, но не знает маршрута. Узел-источник посылает широковещательный пакет с запросом маршрута (RREQ) своим соседям.

Поля пакета приведены на рисунке 9.

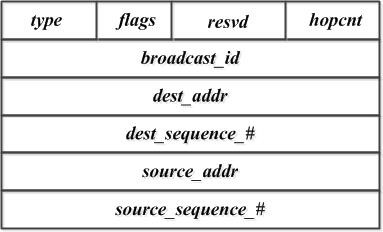


Рисунок 9 — Поля пакета RREQ

Порядковый номер источника в запросе маршрута показывает «степень свежести» обратного маршрута к источнику. Порядковый номер приемника показывает «степень свежести» обратного маршрута к приемнику.

Каждый узел, принимающий RREQ либо возвращает пакет с ответом маршрута (PREQ), либо пересылает RREQ своим соседям.

Поля source\_addr и broadcast\_id уникально идентифицируют RREQ. Broadcasr\_id инкрементируется для каждого посылаемого пакета RREQ, так что приемники могут распознать и удалить дублирующие пакеты RREQ.

Если узел не может ответить на RREQ, то он инкрементирует счетчик узлов и сохраняет информацию, необходимую для поддержки обратного маршрута (в протоколе предполагается наличие симметричных соединений).

К необходимой информации относится:

* идентификатор соседнего узла, который прислал данный пакет RREQ;
* IP адрес места назначения;
* IP адрес источника;
* широковещательный ID;
* последовательный номер узла источника;
* время истечения записи для обратного маршрута.

Если узел принимает пакет RREQ и у него есть текущий маршрут к месту назначения, то он отошлет однонаправленный пакет с ответом маршрута (RREP) тому соседу, от которого он принял пакет RREQ. На рисунке 10 приведена структура пакета RREP.

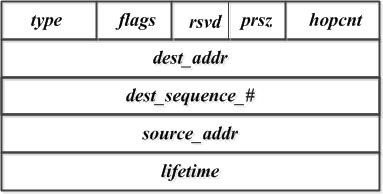


Рисунок 10 — Структура пакета RREP

Промежуточные узлы будут ретранслировать первый RREP по направлению к источнику, используя кэшированные записи обратного маршрута. Остальные пакеты RREP отбрасываются до тех пор, пока номер dest\_sequence\_# больше предыдущего, либо dest\_sequence\_# такой же, но hopcnt меньше (то есть это лучший путь). В конце концов RREP достигает узла, который может использовать соседа, приславшего RREP, в качестве следующего луча для пересылки информации к месту назначения. Кэшированные обратные маршруты будут удаляться в тех узлах, которые не видят пакета RREP.

Причиной изменения маршрута может быть потеря периодических сообщений «hello», авария на связном уровне, ошибка передачи пакета к следующему узлу (может быть обнаружена с помощью прослушивания ретрансляции, если это не конечная точка назначения). Вышестоящий (по направлению к источнику) узел, обнаружив ошибку, передает пакет ошибочного маршрута (RERR) с новым порядковым номером места назначения и количеством лучей, равным бесконечности (недостижимый маршрут). Источник (или другой узел маршрута) может снова построить путь, послав пакет RREQ.

###### 2.6.2 Протокол DSR

DSR (Dynamic source routing) – реактивный протокол динамической маршрутизации от источника, как следует из названия, является одним из протоколов маршрутизации по требованию, основанный на концепции построения пути от источника передачи, то есть явную маршрутизацию. Сети на основе DSR полностью самоорганизующиеся и самостоятельно конфигурируемые. В DSR не применяется метод периодической рассылки сообщений как в AODV, таким образом снижается нагрузка на полосу пропускания, сохраняется заряд аккумулятора мобильных устройств, а также удается избежать слишком частого обновления маршрутных данных и обмена слишком большим количеством информации.

Согласно протоколу DSR, поиск маршрута и поддержка информации о нем от узла к узлу осуществляется двумя главными механизмами: поиск маршрута и поддержка маршрута. DSR постоянно обновляет кэш маршрутов с целью доступности новых удобных маршрутов. «Поиск маршрута» – механизм, с помощью которого узел S, собирающийся послать некие пакеты узлу D, получает исходный маршрут к нему. Поиск маршрута используется только в том случае, если узел S в первый раз пытается получить доступ к D и не знает пути к нему. Поиск осуществляется отправкой broadcast-запроса RREQ. Во время поиска пути происходит накопление адресов устройств, находящихся между получателем и отправителем. В этом процессе участвуют все узлы, обрабатывающие broadcast-запрос. Информация об адресах узлов, через которые прошли пакеты маршрутизации, записывается в заголовках пакетов. Таким образом, узлы могут получать информацию о состоянии других устройств и существующих маршрутах, при условии, что через них проходят пакеты. Маршрут считается сформированным только в том случае, если пакет достиг адресата, тогда отправляется сообщение-ответ. Ответ может быть отправлен либо по известному маршруту, записанному в памяти узла-получателя, либо по цепочке, записанной в заголовке принятого пакета-запроса. В этом случае накладывается условие на линии связи сети – они должны быть симметричны. «Поддержка маршрута» – механизм, с помощью которого узел S способен обнаружить с помощью существующего маршрута к D, изменилась ли топология сети таким образом, что данный маршрут более нельзя использовать вследствие потери соединения на пути. Если разрыв соединения между отправителем и получателем найден, узелотправитель пытается подыскать другой путь или применяет механизм поиска маршрута DSR.

В протоколе присутствует средство защиты от образования петель. Вся информация о пакете находится в его заголовке, в том числе и информация по маршрутизации. Поэтому промежуточные узлы могут записывать эту информацию в кэш и свои маршрутные таблицы для использования в будущем.

Итак, когда устройство хочет передать какие-то данные другому устройству, к которому нет известного пути, оно инициализирует пакет RREQ, который распространяется по сети. Каждый узел после получения пакета RREQ пересылает его своим соседям, если не является получателем и время жизни пакета не вышло. В случае, если копия этого пакета снова пришла в узел, повторно пересылаться она не будет. Проверка на дубликаты производится по порядковому номеру, записанному в заголовке пакета. Номер этот был присвоен узлом-источником, он запоминается каждым узлом, через который прошел однажды. Таким образом, петли в маршрутах полностью исключаются. Узел-адресат при получении этого пакета RREQ отвечает на запрос сообщением RREP, направляя по обратному маршруту, либо по уже существующему пути, хранящемся в кэше узла.

К недостаткам протокола относится неспособность восстанавливать разорванные соединения в местном масштабе механизмом поддержки маршрута. Устаревшая информация из кэша маршрута может привести к рассогласованности при реконструкции маршрута. Производительность протокола уменьшается с увеличением подвижности узлов. При использовании метода маршрутизации от источника наблюдаются издержки маршрутизации при увеличении длины пути.

###### 2.6.3 Протокол OLSR

Протокол OLSR (Optimized Link-State Routing Protocol) решает задачи обнаружения соседних узлов и поддержания соединений с ними, распространения информации о существующих соединениях с соседними узлами по всей сети, поиска кратчайших маршрутов на основании имеющейся на узле маршрутной информации и пошаговой ретрансляции пакетов.

Для обнаружения соседних узлов и поддержания соединения с ними (пара узлов является соседями, если находится в области уверенного приема друг друга) все узлы сети периодически (с интервалом HELLO\_INTERVAL) широковещательно рассылают служебные сообщения HELLO, содержащие адреса соседних узлов и информацию об установленных с ними соединениях. Если в течении времени NEIGHB\_HOLD\_INTERVAL узел не получает ни одного сообщения HELLO от своего соседа, то соединение с этим узлом считается разорванным.

Сообщения HELLO не ретранслируются по всей сети, поэтому с их помощью каждый узел может узнать сетевую информацию лишь о своем двухшаговом окружении.

Назовем узел n одношаговым соседом узла x, если узел х находится в области уверенного приема узла n. Узел d, не являющийся одношаговым соседом узла х, назовем двухшаговым соседом узла х, если узел d является одношаговым соседом хотя бы одного одношагового соседа узла х.

Для распространения информации о соединениях с одношаговыми соседями по всей сети узлы периодически (с интервалом TC\_INTERVAL) отправляют широковещательные сообщения TOPOLOGY\_CONTROL (TC).

Информация о соединении между парой узлов, полученная из TC некоторого узла-источника, обновляется при получении каждого нового TC сообщения от этого узла, и удаляется, если либо узел-источник TC больше не рассылает информацию о данном соединении, либо завершился интервал TOP\_HOLD\_INTERVAL с момента получения последнего TC от рассматриваемого узла-источника.

Все широковещательные служебные сообщения ретранслируются с использованием случайной задержки – джиттера; по умолчанию он выбирается равновероятно из интервала [0, HELLO\_INTERVAL/4).

На основании информации, получаемой из HELLO и TC, каждый узел строит ориентированный граф, который является представлением беспроводной сети данного узла. До каждого узла сети в полученном графе определяется кратчайший маршрут, представляющий собой цепочку ретрансляторов. Адрес конечного получателя и первого ретранслятора образуют запись в таблице маршрутизации.

При необходимости доставить пакет до конечного получателя узел находит нужную запись в таблице маршрутизации и пересылает пакет указанному в ней ретранслятору. Ретранслятор, получив пакет, проделывает аналогичную процедуру, при этом маршрут, используемый узломретранслятором, может отличаться от маршрута источника, поскольку ретранслятор имеет собственное видение топологии сети. Таким образом, пакет передается до тех пор, пока не достигнет коечного получателя или не будет отброшен в случае зацикливания маршрута.

Ключевой особенностью протокола OLSR, снижающей загруженность сети при широковещательной рассылке, является использование так называемых MPR-ретрансляторов (MultiPoint Relays). Каждый узел выбирает из множества своих одношаговых соседей, с которыми установлено двунаправленное соединение, MPR-ретрансляторы таким образом, чтобы каждый двухшаговый сосед данного узла являлся одношаговым соседом по крайней мере одного из его MPR-ретрансляторов.

MPR-ретрансляторы играют важную роль при распространении маршрутной информации и пересылке широковещательных сообщений. Вопервых, каждый узел, по умолчанию, включает в сообщение TC информацию о двунаправленных соединениях только с теми соседями, которые выбрали данный узел в качестве MPR-ретранслятора. Благодаря этому уменьшается число соединений, информация о которых рассылается по сети. Во-вторых, узел Y пересылает широковещательное сообщение, полученное от его соседа – узла X, только в том случае, если Y является MPR-ретранслятором узла Х. Таким образом снижается число пересылок при распространении одного широковещательного сообщения.

###### 2.6.4 Протокол ZPR

ZRP (Zone Routing Protocol) гибридный протокол, в котором используется проактивный механизм поиска узлов, находящихся в зоне маршрутизации, определяемой совокупностью узлов с минимальным расстоянием в хопах, не превышающем выбранный радиус зоны. Поиск маршрута вне зоны основан на рассылке служебной информации по её периметру, вместо рассылки по всей сети, и использовании реактивного механизма поиска маршрутов по требованию до пунктов назначения, расположенных за пределами зоны маршрутизации. Проактивный компонент ZRP, называемый внутризоновым протоколом маршрутизации (Intrazone routing protocol, IARP), реализован на основе дистанционно-векторного алгоритма поиска маршрутов, а межзоновый протокол (Interzone routing protocol, IERP) использует механизм запрос-ответ для поиска маршрутов по требованию для взаимодействия с узлами в разных зонах.

На рисунке 11 представлено сравнение протоколов маршрутизации AODV, DSR, OLSR, ZRP в ad hoc сетях.

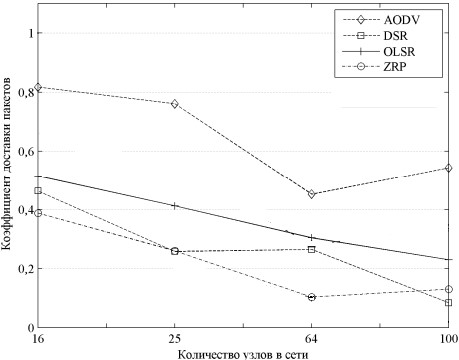


Рисунок 11 — Сравнение протоколов маршрутизации в ad hoc сетях

##### 2.7 Эмуляция ad hoc сетей

Появляющаяся область сетей ad hoc породила множество протоколов и архитектур маршрутизации. В настоящее время существует гораздо больше проектов и проектов протокола, чем есть реализации, поскольку тестирование беспроводных ad hoc сетей требует значительных организационных навыков и человеческих ресурсов.

Симуляторы, такие как NS, дают разработчику протокола очень подробный отчет о событиях, которые происходят в данном смоделированном сценарии. Однако часто этот уровень детализации не требуется, и разработчик протокола больше предпочитает работать с симулятором, который позволит ему интерактивно взаимодействовать с протоколом.

Кроме того, если протокол должен быть реализован дважды, один раз для симулятора и затем еще раз — для развертывания, это добавляет нагрузки к рабочей реализации.

Ниже описываются инструменты моделирования, доступные разработчикам систем, работающих с ad hoc сетями.

###### 2.7.1 NS2

Ns2 — объектно-ориентированное программное обеспечение, ядро которого реализовано на языке C++. Интерпретатором служит язык OTcL (Object oriented Tool Command Language) – язык сценариев. Присущая обоим языкам иерархия классов полностью поддерживается в ns2, кроме того существует однозначное соответствие между классами двух иерархий.

В ns2 реализовано множество протоколов. Можно найти практически любой, применяемый в современных сетях связи протокол. В качестве примера можно привести MPLS, IPv6, OSPF, RSVP. Кроме того, реализованы дисциплины обслуживания очередей, такие как RED, WFQ, CBQ, SFQ и другие. Среди прочего следует отметить поддержку протоколов маршрутизации в беспроводных сетях, например, AODV, DSDV, DSR и так далее.

Благодаря применению математического аппарата в ns2 реализуются различные виды трафика – от простейшего, подчиняющегося закону Пуассона, до самоподобного. Гибкость ns2 дает возможность пользователю реализовывать собственные математические функции на C++.

В ns2 возможно моделирование возникновения ошибок в процессе передачи данных на канальном уровне, то есть искажения и потери информации. Задать можно как битовые, так и пакетные ошибки, кроме того можно обозначить вероятность ошибки в секунду или использовать пользовательскую модель ошибок.

В ns2 имеется средство для визуализации результатов моделирования – NAM (Network Animator). NAM графически воспроизводит проведенный эксперимент: отображает топологию сети, потоки передаваемых пакетов, узлы, очереди и так далее. NAM работает с данными, полученными по ходу моделирования сети, записываемыми в специальный файл трассировки.

Также в ns2 присутствует средство построения графиков Xgraph, которое можно запустить прямо из скрипта \*.tcl сразу по завершению процесса моделирования, получив, таким образом, графическое отображение статистических данных по моделируемой сети.

Ns2 является бесплатной программой, так как изначально разрабатывалась как программное обеспечение с открытым кодом. На распространение, модификацию и использование такого ПО не накладываются никакие ограничения. Другой особенностью ns2 с точки зрения гибкости является поддержка работы во многих операционных системах, таких как Linux, SunOS, Solaris, FreeBSD, Windows.

###### 2.7.2 NS3

Ns-3 разрабатывается с 2006 и является современным наследником ns2, но при этом с ns-2 совершенно не совместим, так как писался с нуля. Так же, как и ns-2 является программным продуктом с открытым кодом и ориентирован на использование в исследовательской среде. Для работы с ним предполагается программирование на С++ и Python. Для визуализации процесса моделирования необходимы внешние программы.

Ns-3 отличает современная архитектура и качественное ядро, а также приближенные к реальности модели и интеграция с реальным ПО. Но в тоже время в программе довольно ограниченная библиотека моделей, отсутствует GUI/IDE. В данном продукте не представлены необходимые протоколы маршрутизации.

###### 2.7.3 OMNeT++

OMNeT++ является системой моделирования на основе дискретных событий. По большей части данная система ориентирована на стандартные проводные и беспроводные сети IP коммуникаций. Данная система широко известна, расширяема и активно поддерживается сообществом своих пользователей.

Для имитационных моделей OMNeT++ применяется язык C++. Так же используется язык высокого уровня NED. Имитационные модели объединяются в крупные компоненты и представляют собой большие системы. В составе симулятора присутствуют графические инструменты для создания моделей и отображения результатов в режиме реального времени.

Симулятор хорошо масштабируем для очень больших сетевых топологий, в этом направлении возможности ограничиваются лишь ресурсами компьютера. В симуляторе определимы задержки для нижних уровней, например, MAC беспроводного канала.

Симулятору не хватает протоколов и должного качества моделирования беспроводных сетей типа MANET. Отсутствует поддержка необходимых протоколов маршрутизации, так как продукт ориентирован в основном на поддержку IP-сетей.

Сравнение перечисленных систем моделирования представлено в таблице 1.

Таблица 1 — Сравнение систем моделирования

|  |  |  |  |
| --- | --- | --- | --- |
| Среда моделирования | Поддержка  ad hoc сетей | Наличие протоколов маршрутизации | Наличие инструментов для обработки результатов моделирования |
| OMNeT++ | - | - | + |
| Ns-2 | + | + | + |
| Ns-3 | + | - | - |

# 3 Общие сведения о графах

Граф — это множество точек или вершин и множество линий или ребер, соединяющих между собой все или часть этих точек. Вершины, прилегающие к одному и тому же ребру, называются смежными.

Если ребра ориентированны, что обычно показывают стрелками, то они называются дугами, и граф с такими ребрами называется ориентированным графом. Если ребра не имеют ориентации, граф называется неориентированным.

Графы обычно изображаются в виде геометрических фигур, так что вершины графа изображаются точками, а ребра — линиями, соединяющими точки. Пример неориентированного и ориентированного графов приведен на рисунке 12.

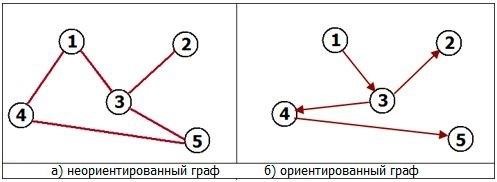


Рисунок 12 — Пример графа

Петля — это дуга, начальная и конечная вершина которой совпадают. Простой граф — граф без кратных ребер и петель. Степень вершины — это удвоенное количество петель, находящихся у этой вершины плюс количество остальных прилегающих к ней ребер. Пустым называется граф без ребер.

Полным называется граф, в котором каждые две вершины смежные.

Путь в ориентированном графе — это последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

Вершины v0, vn называются связанными данным путем (или просто связанными). Вершину v0 называют началом, vn - концом пути. Если v0 = vn, то путь называют замкнутым. Число n называется длиной пути.

Маршрут в графе — путь, ориентацией дуг которого можно пренебречь. Цепь — маршрут, в котором все ребра попарно различны. Цикл — замкнутый маршрут, являющийся цепью. Маршрут, в котором все вершины попарно различны, называют простой цепью. Цикл, в котором все вершины, кроме первой и последней, попарно различны, называются простым циклом.

Подграф графа — это граф, являющийся подмоделью исходного графа, т.е. подграф содержит некоторые вершины исходного графа и некоторые ребра (только те, оба конца которых входят в подграф).

Подграф, порожденный множеством вершин U — это подграф, множество вершин которого — U, содержащий те и только те ребра, оба конца которых входят в U.

Подграф называется остовным подграфом, если множество его вершин совпадает с множеством вершин самого графа.

Граф называется связным, если любая пара его вершин связана. Связными компонентами графа называются подграфы данного графа, вершины которых связаны.

Существует два способа представления графа, в виде списков смежности и в виде матрицы смежности. Оба способа подходят для представления ориентированных и неориентированных графов.

Матрица смежности является удобным для представления плотных графов, в которых количество рёбер (|E|) примерно равно количеству вершин в квадрате (|V|2).

В данном представлении мы заполняем матрицу размером |V| x |V| как представлено в формуле 1.

Aij = (1)

Данный способ подходит для ориентированных и неориентированных графов. Для неориентированных графов матрица A симметрична (то есть A[i][j] == A[j][i], т.к. если существует ребро между i и j, то оно является и ребром из i в j, и ребром из j в i). Благодаря этому свойству можно сократить почти в два раза использование памяти, храня элементы только в верхней части матрицы, над главной диагональю).

Понятно, что с помощью данного способа представления, можно быстро проверить есть ли ребро между вершинами v и u, просто посмотрев в ячейку A[v][u].

С другой стороны, этот способ очень громоздкий, так как требует O (|V|2) памяти для хранения матрицы.

На рисунке 13 приведены представления графов из рисунка 12 с помощью матриц смежности.

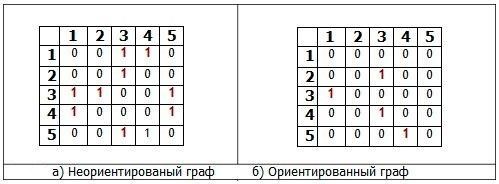


Рисунок 13 — Пример матрицы смежности

Списки смежности больше подходят для разреженных графов, то есть графов у которых количество рёбер гораздо меньше чем количество вершин в квадрате (|E| << |V|2).

В данном представлении используется массив Adj содержащий |V| списков. В каждом списке Adj[v] содержатся все вершины u, так что между v и u есть ребро. Память, требуемая для представления, равна O (|E| + |V|), что является лучшим показателем, чем матрица смежности для разреженных графов.

Главный недостаток этого способа представления в том, что нет быстрого способа проверить существует ли ребро (u, v).

На рисунке 14 приведены представления графов из рисунка 12 с помощью списков смежности.

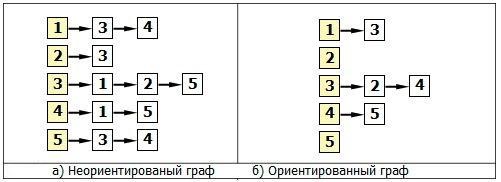


Рисунок 14 — Пример списка смежности

# 4 Практическая реализация 4.1 Представление ad hoc сети и оценка сложности алгоритмов

В реализованной программе достижимость узлов ad hoc сети представлена в виде матрицы смежности = ( ) графа сети, т.е.

квадратной × матрицы, определяемой формулой 2.

Auv =  (2)

Граф генерируется из списка списков объектов, описывающих узел сети: его координаты по осям *X* и *Y*, а также его радиус, которые генерируются в соответствии с заданием.

Определение сильной связности, односторонней связности, слабой связности, несвязности, наряду с поиском пути из узла в узел основаны на алгоритме обхода геометрического графа в глубину, сложность которого может быть оценена как O(n+m), где n — количество вершин, m — рёбер. Алгоритм Тарьяна, использовавшийся для поиска сильных компонент связности имеет подобную сложность(O(n+m)).

## 4.2 Язык программирования Python

В качестве языка программирования был выбран язык Python версии 3.5.2. Он сочетает в себе замечательную мощь с очень четким синтаксисом. В нем есть модули, классы, исключения, динамические типы данных на высоком уровне и динамическая типизация. Существуют интерфейсы для многих системных вызовов и библиотек, а также для различных оконных систем. Новые встроенные модули легко записываются на C или C ++ (или на другие языки, в зависимости от выбранной реализации). Python также можно использовать в качестве языка расширения для приложений, написанных на других языках, для которых требуются простые в использовании сценарии или интерфейсы автоматизации.

## 4.3 Библиотека matplotlib

Matplotlib — это графическая библиотека для языка программирования Python и его численного математического расширения NumPy. Он предоставляет объектно-ориентированный API для встраивания графиков в приложения с использованием универсальных GUI-инструментов, таких как

Tkinter, wxPython, Qt или GTK +. Существует также процедурный интерфейс «pylab», основанный на конечной машине (например, OpenGL), разработанный так, чтобы он напоминал интерфейс MATLAB. SciPy использует matplotlib.

Matplotlib распространяется под лицензией BSD. Сейчас matplotlib поддерживает версии от 2.4 до 3.5 Python. Matplotlib 1.2 — первая версия matplotlib для поддержки Python 3.x. Matplotlib 1.4 является последней версией matplotlib для поддержки Python 2.6.

## 4.4 Интегрированная среда разработки PyCharm

PyCharm - это интегрированная среда разработки (IDE), используемая в

компьютерном программировании, особенно для языка Python, разработанная компанией JetBrains. PyCharm обеспечивает анализ кода, графический отладчик, встроенный тестер модулей, интеграцию с системами контроля версий (VCS) и поддерживает веб-разработку с помощью Django.

PyCharm является кросс-платформенным, имея версии для Windows, MacOS и Linux. Издание Community Edition выпущено под лицензией Apache, а также выпущена Professional Edition с дополнительными функциями, распространяющаяся по лицензии.

## 4.5 Структура программы

#### 4.5.1 Основные методы программы

В процессе реализации программы, имитирующей работу узлов ad hoc сети, были созданы следующие классы:

а) Node имеет свойства x, y, radius. Используется для хранения

информации об узле сети: его координатам по осям *X* и *Y*, его радиусе;

б) Graph имеет свойство matrix, хранящее матрицу смежности, которой

представлен граф. Также имеет следующие методы:

1. конструктор \_\_init\_\_(self, matrix) создает новый граф из матрицы смежности, присваивая ее полю matrix;
2. dfs(self, v, visited) выполняет поиск в глубину, принимая вершину и список посещенных вершин;
3. dfs2(self, v, visited) выполняет модифицированный поиск в глубину, принимая вершину и список посещенных вершин, необходим для алгоритма определения, является ли граф однонаправленным;
4. matrix\_to\_list(self, matrix) трансформирует представление графа в виде матрицы смежности в список смежности;
5. vertexes\_count(self) возвращает количество вершин в графе;
6. edges\_count(self) возвращает количество ребер в графе;
7. get\_transposed(self) возвращает транспонированную матрицу смежности графа;
8. get\_undirected(self) создает неориентированный граф из ориентированного;
9. is\_strongly\_connected(self) проверяет, является ли граф сильно связным;
10. is\_weakly\_connected(self) проверяет, является ли граф слабо связным;
11. is\_unilaterally\_connected(self) проверяет, является ли граф односторонне связным;
12. is\_connected(self) проверяет, является ли граф связным.

Помимо вышеназванных методов в программе существуют следующие функции:

а) generate\_nodes(area\_radius, n\_nodes, is\_node\_radius\_random, radius\_deviation) генерирует заданное количество узлов с заданным радиусом в заданной окружности;

б) line\_length(x1, y1, x2, y2) находит длину отрезка с координатами

(x1; y1), (x2, y2);

в) are\_circles\_intersect(x1, y1, r1, x2, y2, r2) проверяет, пересекаются ли окружности с заданными координатами и радиусами;

г) is\_point\_in\_circle(p\_x, p\_y, c\_x. c\_y. c\_radius) проверяет, находится ли точка с координатами (p\_x; p\_y) в окружности с координатами (c\_x; c\_y) и радиусом c\_radius;

д) get\_random\_point\_in\_area(area\_radius) возвращает случайную точку в

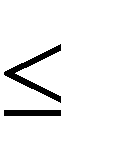
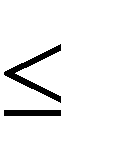
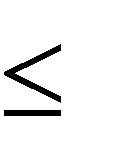
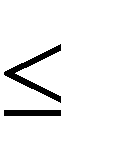
окружности с заданным радиусом и координатами (0; 0);

е) graph\_from\_nodes(nodes) генерирует граф достижимости узлов ad

hoc сети из списка узлов с координатами в виде матрицы смежности.

#### 4.5.2 Описание работы программы

Была реализована программная имитация ad hoc сети с заданными согласно заданию параметрами. В начале работы программа устанавливает такие параметры, как размер сети, количество генерируемых ориентированных графов для каждой конфигурации сети. Для каждого значения 2 *n* 10 количества узлов в сети выполнение последующего алгоритма разбивается на две части. Первая часть выполняется для случайного, взаимно независимого, равномерно распределенного на [1 – *c*, 1 + *c*] значения радиусов узлов со средним значением 1,0. Анализ проводится для различных значений 0,2 *c* 0,8 с шагом 0,1. Вторая часть выполняется для радиуса R = 1 и необходима для сравнительных оценок.



Для каждого значения радиуса происходит генерация 105 узлов сети с переменными параметрами. Для этого создается пустой список узлов и в соответствии с размером сети создаются и затем добавляются в список узлы со случайными координатами и, в зависимости от шага, либо случайным радиусом от [1 – *c*, 1 + *c*], либо константным, равным 1. Генерация случайных координат учитывает проверку нахождения в пределах обозначенной заданием окружности. Проверка заключается в сравнении суммы квадратов разниц координат по оси X и разниц координат по оси Y. Если эта сумма меньше квадрата радиуса окружности, то координаты находятся внутри этой окружности.

В дальнейшем из узлов сети строятся графы достижимости, после чего инкрементируется счетчик разновидностей графов, необходимый в дальнейшем для составления таблицы вероятностей появления орграфов, которая соответствует таблице диаграмм орграфов, приведенной в книге Ф.Харари «Теория графов», а также перечня наиболее и наименее вероятных орграфов. Граф строится путем прохода по списку узлов и поиска других узлов, которые находятся в области покрытия узла. Полученную матрицу смежности мы можем использовать для дальнейших расчетов.

На следующем шаге выполняется проверка каждого графа на сильную связность, слабую связность, одностороннюю связность, несвязность и увеличиваются соответствующие счетчики в случае положительного результата.

Для проверки графа на сильную связность в программе используется алгоритм Косарайю, который по данному ориентированному графу, определяет, сильно ли он связан или нет. Ориентированный граф сильно связан, если существует путь между любыми двумя парами вершин.

Например, граф на рисунке является сильно связным.

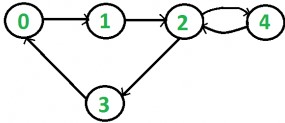


Рисунок 15 — Сильно связный граф

Алгоритм прост для неориентированного графа, можно просто сделать поиск в глубину (DFS) или в ширину (BFS), начиная с любой вершины. Если BFS или DFS посетили все вершины, то неориентированный граф связан. Этот подход не будет работать для ориентированного графа. Например, рассмотрим граф на рисунке 16, который не сильно связан. Если мы начинаем DFS (или BFS) с вершины 0, мы можем достичь всех вершин, но если мы начнем с любой другой вершины, мы не сможем достичь всех вершин.



Рисунок 16 — Граф, не являющийся сильно связным

Для ориентированного графа можно использовать алгоритм поиска сильных компонент. Мы можем найти все сильные компоненты за O(V+E) времени, где V — количество вершин, а E — количество ребер. Если число сильных компонент равно единице, то граф сильно связан. Алгоритм для поиска сильных компонент делает дополнительную работу, поскольку находит все сильные компоненты.

Следующий алгоритм, основанный на DFS на основе Косарайю, выполняет два обхода графика DFS:

а) инициализировать все вершины как не посещенные;

б) провести обход в глубину графа, начиная с любой произвольной вершины v, если обход в глубину не посещает все вершины, тогда завершить алгоритм и вернуть ложь;

в) обратить все дуги (или найти обратный граф);

г) отметить все вершины в обратном графе как не посещенные;

д) провести обход в глубину в обратном графе, начиная с той же вершины v, что и на шаге б); если обход DFS не посещает все вершины, то вернуть ложь, в противном случае — истину.

Идея состоит в том, что если каждый узел может быть достигнут из вершины v, и каждый узел может достигнуть v, то граф сильно связан. На шаге б) мы проверяем, достижимы ли все вершины из v. На шаге г) мы проверяем, могут ли все вершины достигать v в обратном графе, если все вершины достижимы из v, то все вершины могут достигать v в исходном графе.

Для проверки, является ли граф слабо связным, в программе учитывается тот факт, что если построенный из ориентированного графа неориентированный граф является связным, то исходный ориентированный граф является слабо связным.

Для того, чтобы узнать, является ли граф односторонне связным, в программе используется модифицированный поиск в глубину. Мы можем начать поиск из любой вершины. Необходимо отметить все вершины, достижимые из первой вершины, по мере посещения. Затем мы проходим через все вершины. Для каждой непосещенной вершины мы проводим поиск в глубину, ведя список, содержащий все непосещенные вершины, которые мы прошли. Если во время поиска в глубину мы не посещаем ни одной посещенной во время предыдущего поиска в глубину вершины, то мы можем сразу сделать вывод, что ориентированный граф имеет несколько связных компонент и не является односторонним. В противном случае мы помечаем все вершины в списке, полученном во время поиска, как посещенные и продолжаем. Как только мы пройдем через все вершины графа, и все поиски в глубину пройдут некоторые посещенные вершины, мы можем заключить, что граф является односторонне связанным.

Для проверки графа на связность мы можем использовать поиск в глубину. Мы можем выбрать одну из вершин и пометить ее. Затем пометить все вершины, смежные с ней. На следующем шаге помечаем все вершины, смежные с этими вершинами, кроме уже помеченных. И так далее до тех пор, пока на очередном шаге мы не сможем поставить ни одну новую метку. Если помечены все вершины графа, то граф связный. В противном случае граф несвязный, и мы только что пометили один из компонентов связности.

После генерации необходимого количества графов и их проверки мы можем посчитать, сколько процентов составляет доля каждого типа графа, и добавить эти значения в список для дальнейшей обработки и вывода в виде графика.

Мы формируем список для каждого типа графов для каждого размера сети n, а затем списки этих списков для каждого типа графов. В итоге результирующие списки хранят списки для каждого размера сети n.

Для удобного отображения на графиках списки списков необходимо представить в виде двумерного массива и транспонировать.

Затем мы строим список вероятностей появления ориентированных графов, список наиболее вероятных орграфов и список наименее вероятных ориентированных графов.

После этого нам осталось вывести графики полученных результатов:

вероятность появления сильно связного, слабо связного, односторонне связного и несвязного ориентированных графов.

Полный исходный текст программы находится в приложении А.

4.5.3 Результаты работы программы

В результате работы программы на основе достаточного (105) числа экспериментов мы получили графики вероятностей появления различных видов связности орграфов, соответствующих расположению сети.

На рисунке 17 изображена вероятность появления сильной связности орграфа.

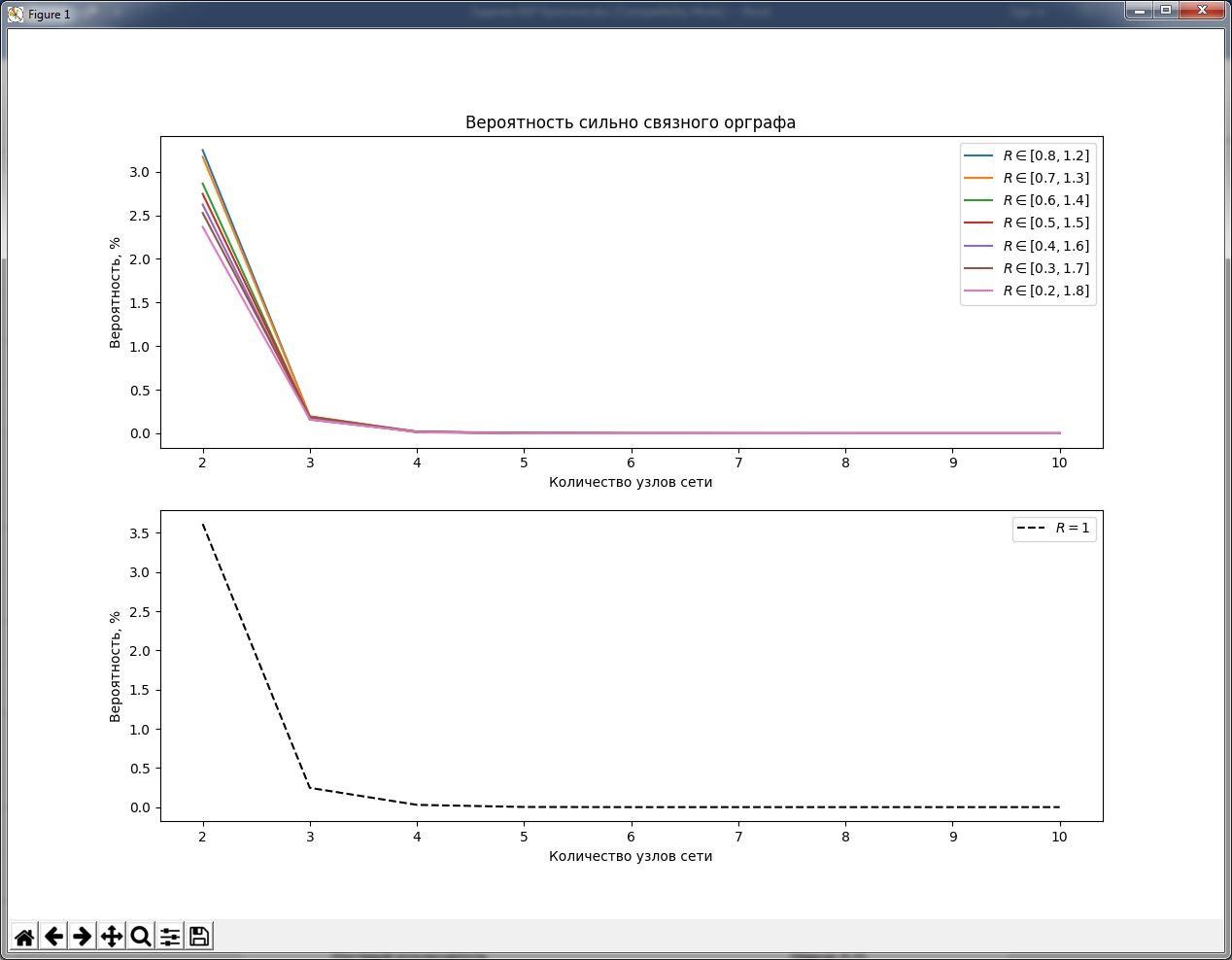


Рисунок 17—Вероятность появления сильно связного орграфа

Можно заметить, что при количестве узлов сети, равном двум, процент сильно связных графов остается очень небольшим — около 2,5–3%, — а при увеличении количества узлов сети до 3 падает до 0,2%. При количестве узлов, равном 4 и более, процент сильно связных графов падает практически до нуля.

Также замечается обратная небольшая зависимость от радиуса покрытия узлов. Чем больше разброс радиуса покрытия, тем ниже вероятность появления сильной связности графа.

Для сравнения, при отсутствии радиуса покрытия при радиусе, равном 1, вероятность появления сильной связности орграфа при выше.

Например, она равна 3,6% и 0,4% для двух и трех узлов соответственно.

График вероятности появления слабой связности орграфа, показанный на рисунке 18, очень похож на график вероятности появления сильной связности графа.

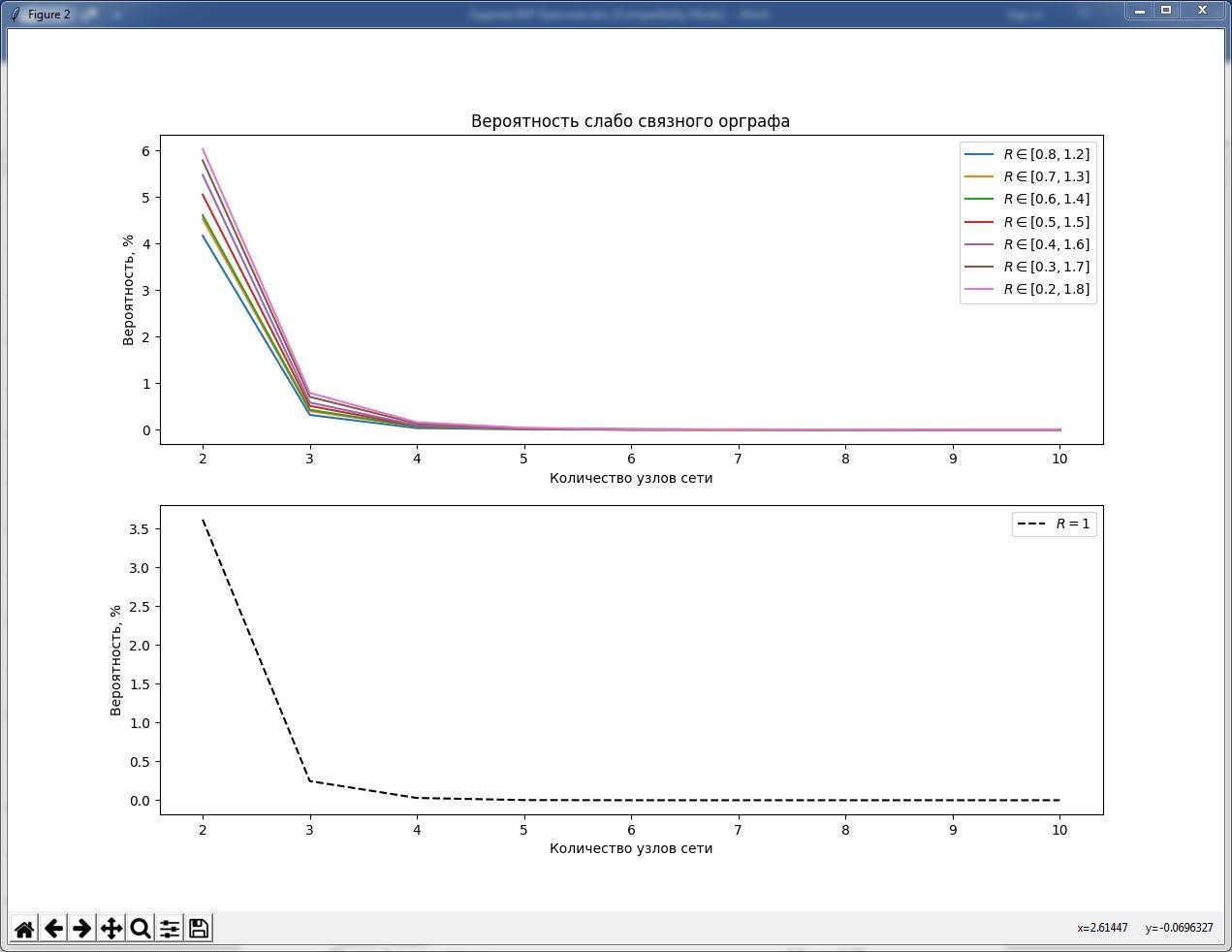


Рисунок 18 — Вероятность появления слабо связного орграфа

Его отличительная черта состоит в том, что зависимость вероятности появления слабой связности графа от разброса радиуса покрытия узлов не обратная, а прямая. При R = 1 мы имеем практически те же результаты вероятности, что и для сильно связных графов. Однако, при увеличении разброса радиусов, вероятность появления увеличивается вплоть до 6% при двух узлах и R  [0,2; 1,8] и до 1,8% при тех же радиусах и трех узлах.

На рисунке 19 представлена вероятность появления односторонне связного орграфа.

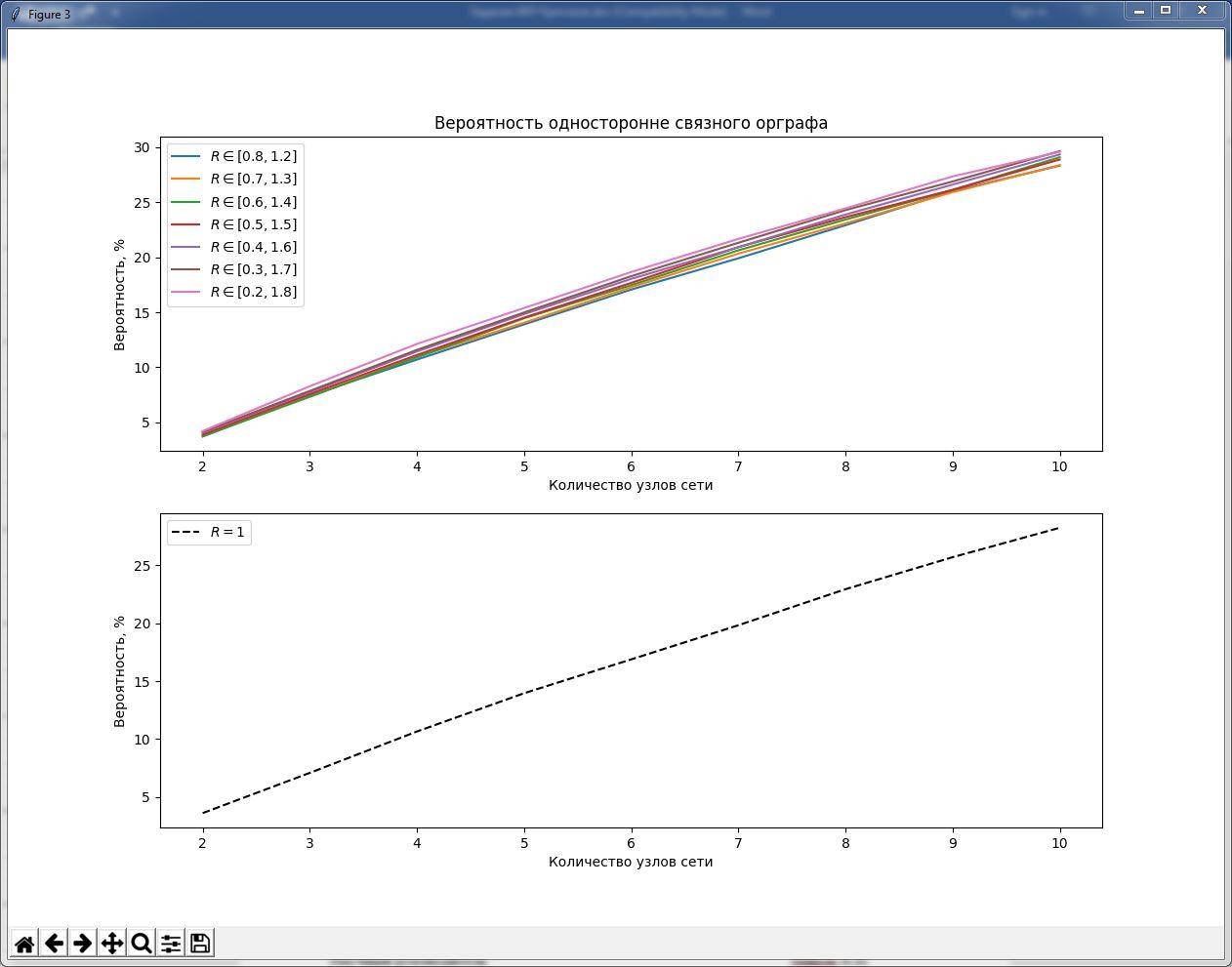


Рисунок 19 — Вероятность появления односторонне связного графа

Прослеживается прямая линейная зависимость вероятности от количества узлов сети — от 2% при двух узлах до 28% при десяти узлах. Также заметна небольшая прямая зависимость вероятности от разброса радиусов — чем разброс больше, тем вероятность выше.

Вероятность появления несвязного орграфа представлена на рисунке

20.

Видна зависимости прямой вероятности появления несвязного графа от узлов сети — чем узлов больше, тем вероятность выше. При этом вероятность уже для двух узлов очень высока — около 96,3%, а для трех близка к 100% — 99,8%. Прослеживается небольшая обратная зависимость от разброса радиусов — чем разброс больше, тем вероятность ниже.

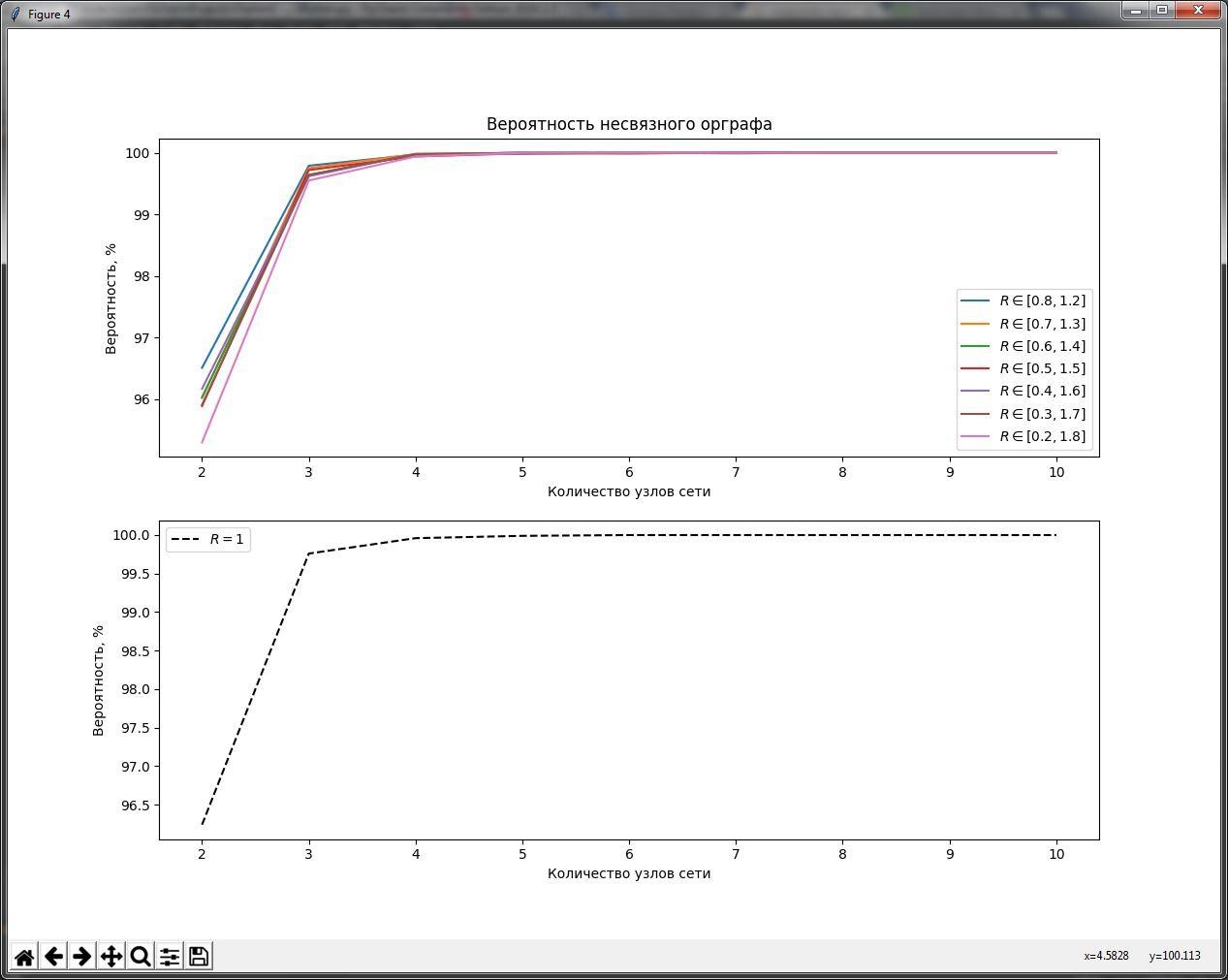


Рисунок 20 — Вероятность появления несвязного графа

В таблице 2 представлены вероятности появления орграфов соответствующая таблице диаграмм орграфов, приведенной в книге Ф. Харари «Теория графов», 2 *p* 4, 0 *q* 12, где *p —* количество вершин, а *q* — количество ребер. Так как при *p* = 4 существует 218 орграфов, для *p* = 4 в таблице представлен один орграф для каждого 0 *q* 12.

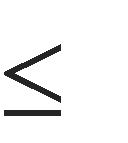
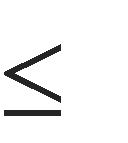
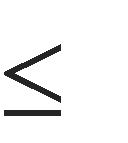
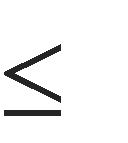
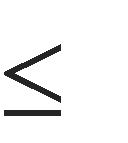
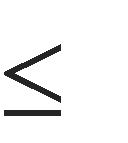


Таблица 2 — Вероятности появления орграфов

|  |  |  |  |
| --- | --- | --- | --- |
| Количество вершин | Количество ребер | Орграф | Вероятность, % |
| 2 | 0 |  | 95,05275 |
| 2 | 1 |  | 2,01025 |
| 2 | 2 |  | 2,937 |
| 3 | 0 |  | 85,93425 |
| 3 | 1 |  | 5,634275 |
| 3 | 2 |  | 8,084125 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 |  | 0,0575 |
| 3 | 2 |  | 0,066 |
| 3 | 2 |  | 0,01925 |
| 3 | 3 |  | 0,039125 |
| 3 | 3 |  | 0,0195 |
| 3 | 3 |  | 0,00475 |
| 3 | 3 |  | 0,0231 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 4 |  | 0,02575 |
| 3 | 4 |  | 0,00025 |
| 3 | 4 |  | 0,0165 |
| 3 | 4 |  | 0,009125 |
| 3 | 5 |  | 0,006 |
| 3 | 6 |  | 0,0605 |
| 4 | 0 |  | 73,71575 |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 1 |  | 9,39875 |
| 4 | 2 |  | 4,65 |
| 4 | 3 |  | 0,01825 |
| 4 | 4 |  | 0,00625 |
| 4 | 5 |  | 0,000125 |
| 4 | 6 |  | 0,00375 |
| 4 | 7 |  | 0,005 |
| 4 | 8 |  | 0,00125 |
| 4 | 9 |  | 0,0025 |
| 4 | 10 |  | 0,00125 |
| 4 | 11 |  | 0,0025 |
| 4 | 12 |  | 0,00625 |

В таблице 3 представлен перечень наиболее вероятных орграфов.

Видно, что наиболее вероятным графом для количества узлов в сети 2 n 10 является граф без ребер, а вероятность его появления снижается с увеличением n, падая с 95,05275 % при n = 2 до 9,94375 % при n = 10.

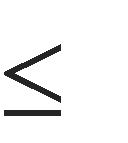
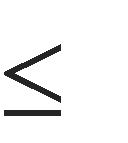
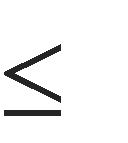
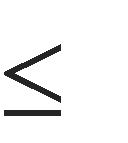


Таблица 3 — Перечень наиболее вероятных орграфов

|  |  |  |
| --- | --- | --- |
| Количество вершин | Наиболее вероятный орграф | Вероятность, % |
| 2 |  | 95,05275 |

|  |  |  |
| --- | --- | --- |
| 3 |  | 85,93425 |
| 4 |  | 73,71575 |
| 5 |  | 60,04125 |
| 6 |  | 46,37 |
| 7 |  | 34,0465 |
| 8 |  | 23,921625 |
| 9 |  | 15,8345 |
| 10 |  | 9,94375 |

Перечень наименее вероятных орграфов представлен в таблице 4. Видно, что для количества узлов в сети 3 n 10 вероятность появления наименее вероятного орграфа падает до 0,000125 %. Эта вероятность означает, что при 800000 экспериментах был обнаружен всего один такой граф согласно формуле 3.



(3)

Таблица 4 — Перечень наименее вероятных орграфов

|  |  |  |
| --- | --- | --- |
| Количество вершин | Наименее вероятный орграф | Вероятность, % |
| 2 |  | 1,002375 |
| 3 |  | 0,000125 |
| 4 |  | 0,000125 |
| 5 |  | 0,000125 |

|  |  |  |
| --- | --- | --- |
| 6 |  | 0,000125 |
| 7 |  | 0,000125 |
| 8 |  | 0,000125 |
| 9 |  | 0,000125 |
| 10 |  | 0,000125 |

Сложность использованных алгоритмов определения связности графов от размера сети *n* представлена в таблице 5.

Таблица 5 — Сложность использованных алгоритмов определения связности орграфа

|  |  |  |  |
| --- | --- | --- | --- |
| Сильно связный | Слабо связный | Односторонне связный | Несвязный |
| O(*n*) | O(*n*) | O(*n*) | O(*n*) |

Сложность алгоритмов определения сильной связности, слабой связности, односторонней связности и несвязности орграфа одинакова и равна *O(V+E)*, где *V* — количество вершин, а *E* — количество ребер. Таким образом, сложность всех алгоритмов в зависимости от размера сети *n* будет равна *O(n)*.

ЗАКЛЮЧЕНИЕ

В рамках настоящей выпускной квалификационной были изучены принципы построения ad hoc сетей и их работы. Разработаны алгоритмы и программы для исследования, и в последствие проведены исследования того, какие именно орграфы (с точки зрения теории графов) соответствуют возможным конфигурациям ad hoc сетей. При этом анализировалась не одна конфигурация сети, а множество различных конфигураций, рассматриваемых как реализации случайной сети — случайный геометрический ориентированный граф.

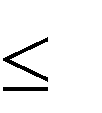
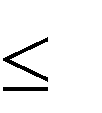
С помощью языка программирования Python были реализован рассмотренные алгоритмы. Получены данные, отображающие полученные результаты. Для вывода графиков использована библиотека matplotlib, написанная на python.

Мы получили результаты исследования, представлены в виде графиков и таблиц, показывающих зависимости для заданных исходных данных.

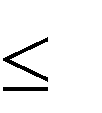
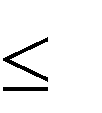
В полученных данных имеются:

* графики вероятности сильной связности, односторонней связности, слабой связности, несвязности в зависимости от величины *n* и величины *c;*
* таблица вероятностей появления орграфов (с учетом изоморфизма), соответствующая таблице диаграмм орграфов, приведенной в книге

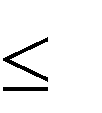
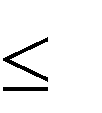
Ф. Харари «Теория графов», 2 *n* 4;



* перечень наиболее вероятных орграфов для 2 *n* 10;



* перечень наименее вероятных орграфов для 2 *n* 10;



* оценка сложности разработанных алгоритмов как функции *n* — размера сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Миков А.И., Мезенцева А.С. Характеристики геометрических графов, моделирующих AD HOC сети // Информатизация и связь. — 2012. — № 5. — с. 85—88.
2. Харари Ф. Теория графов. М.: Эдиториал УРСС, 2003.
3. Миков А.И. Связность автономных беспроводных компьютерных сетей в местностях с плохой инфраструктурой. // Экологический вестник научных центров ЧАЭС, 2014, № 1, с. 70 – 75.
4. Зыков А. А. Основы теории графов. // М.: Наука, 1986. — 456 с.
5. Белов В.В., Воробьев Е.М., Шаталов В.Е. Теория графов. // М.:

Высш.шк., 1976. — 392 с.

1. Емеличев В.А., Мельников О.И., [и др.] Лекции по теории графов. // М.: Наука, 1990. — 384 с.
2. Оре О. Графы и их применение. // М.: URSS, 2006. — 268 с.
3. Хабрахабр [Электронный ресурс]. URL: https://habrahabr.ru/post/197860/ (дата обращения: 01.02.2017).
4. Томский государственный университет систем управления и радиоэлектроники [Электронный ресурс]. URL: http://old.tusur.ru/filearchive/reports-magazine/2010-2-1/288.pdf (дата обращения: 07.03.2017).
5. Trinity College Dublin [Электронный ресурс]. URL: https://www.scss.tcd.ie/~omahony/jemu-iei.pdf (дата обращения: 04.04.2017).

ПРИЛОЖЕНИЕ А

Программный код

**import** random **import** math **import** pylab **import** networkx **import** matplotlib.pyplot **as** plt

**class** Node:

**def** \_\_init\_\_(self, x, y, r):

self.x = x self.y = y self.r = r

**class** Graph: **def** \_\_init\_\_(self, matrix): self.matrix = matrix

**def** dfs(self, v, visited): adj\_list = self.matrix\_to\_list() visited[v] = **True for** i **in** adj\_list[v]: **if not** visited[i]: self.dfs(i, visited)

**def** dfs2(self, v, visited2, l):

adj\_list = self.matrix\_to\_list() visited2[v] = **True** l.append(v) ret = **False for** i **in** adj\_list[v]: **if** visited2[i]: ret = **True else**:

r = self.dfs2(i, visited2, l) ret = ret **or** r

**return** ret

**def** matrix\_to\_list(self):

graph = {} **for** i, node **in** enumerate(self.matrix): adj = [] **for** j, connected **in** enumerate(node): **if** connected: adj.append(j) graph[i] = adj

**return** graph

**def** serialize(self):

ret = 0 **for** i **in** range(0, len(self.matrix)): **for** j **in** range(0, len(self.matrix[i])): **if** self.matrix[i][j] == 1: ret |= 1 << (i \* len(self.matrix[i]) + j) **return** ret

@staticmethod **def** deserialize(number, vertexes): matrix = [[0 **for** \_ **in** range(0, vertexes)] **for** \_ **in** range(0, vertexes)] **for** i **in** range(0, vertexes): **for** j **in** range(0, vertexes): matrix[i][j] = (number >> (i \* len(matrix[i]) + j)) & 1 **return** matrix

**def** vertexes\_count(self): **return** len(self.matrix)

**def** edges\_count(self):

count = 0 **for** i **in** self.matrix: **for** j **in** i: **if** j == 1: count += 1

**return** count

**def** get\_transposed(self): g = [[0 **for** \_ **in** range(len(self.matrix))] **for** \_ **in** range(len(self.matrix))] **for** i **in** range(0, len(self.matrix)):

**for** j **in** range(0, len(self.matrix[i])):

g[j][i] = self.matrix[i][j]

**return** Graph(g)

**def** is\_weakly\_connected(self):

**return** self.get\_undirected().is\_connected()

**def** get\_undirected(self):

transposed = self.get\_transposed() undirected = [[0 **for** \_ **in** range(len(self.matrix))] **for** \_ **in** range(len(self.matrix))] **for** i **in** range(0, len(self.matrix)): **for** j **in** range(0, len(self.matrix[i])):

undirected[i][j] = self.matrix[i][j] **or** transposed.matrix[i][j] **return** Graph(undirected)

**def** is\_strongly\_connected(self):

visited = [**False**] \* len(self.matrix) self.dfs(0, visited) **if** any(i **is False for** i **in** visited): **return False** gr = self.get\_transposed() visited = [**False**] \* len(self.matrix) gr.dfs(0, visited) **if** any(i **is False for** i **in** visited): **return False**

**return True**

**def** is\_unilaterally\_connected(self): adj\_list = self.matrix\_to\_list() vertices = list(adj\_list.keys()) visited = [**False**] \* len(self.matrix)

self.dfs(0, visited) **for** v **in** vertices: **if not** visited[v]: visited\_list = [**False**] \* len(self.matrix) encountered\_previously\_visited = self.dfs2(0, visited, visited\_list) **if** encountered\_previously\_visited:

**for** i **in** visited\_list: visited[i] = **True else**:

**return False**

**return True**

**def** is\_connected(self): visited = [**False**] \* len(self.matrix) self.dfs(0, visited) **for** i **in** range(0, len(self.matrix)): **if not** visited[i]: **return False**

**return True**

**def** print(self):

**for** i **in** range(0, len(self.matrix)): **for** j **in** range(0, len(self.matrix[i])): print(self.matrix[i][j], end=**' '**) print()

print()

**def** generate\_nodes(area\_radius, n\_nodes, is\_node\_radius\_random, radius\_deviation=0):

nodes = [] **for** \_ **in** range(n\_nodes): **if** is\_node\_radius\_random: radius = random.uniform(1 - radius\_deviation, 1 + radius\_deviation) **else**:

radius = 1

x, y = get\_random\_point\_in\_area(area\_radius) nodes.append(Node(x, y, radius)) **return** nodes

**def** line\_length(x1, y1, x2, y2): **return** math.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)

**def** are\_circles\_intersect(x1, y1, r1, x2, y2, r2): **return** line\_length(x1, y1, x2, y2) < r1 + r2

**def** is\_point\_in\_circle(p\_x, p\_y, c\_x, c\_y, c\_radius): **return** (p\_x - c\_x) \*\* 2 + (p\_y - c\_y) \*\* 2 <= c\_radius \*\* 2

**def** get\_random\_point\_in\_area(area\_radius): **while True**: x = random.uniform(-area\_radius, area\_radius) y = random.uniform(-area\_radius, area\_radius) **if** is\_point\_in\_circle(x, y, 0, 0, area\_radius):

**return** x, y

**def** graph\_from\_nodes(nodes):

nodes\_len = len(nodes) graph = [[0 **for** \_ **in** range(nodes\_len)] **for** \_ **in** range(nodes\_len)] **for** i **in** range(0, nodes\_len): **for** j **in** range(i + 1, len(nodes)): graph[i][j] = int(is\_point\_in\_circle(nodes[j].x, nodes[j].y, nodes[i].x, nodes[i].y, nodes[i].r))

graph[j][i] = int(is\_point\_in\_circle(nodes[i].x, nodes[i].y, nodes[j].x, nodes[j].y, nodes[j].r))

**return** Graph(graph)

**def** main():

area\_radius = 5 total\_iterations = 10 \*\* 5

random.seed()

nmax = 10 nmax2 = 4

debug = 0

xlist = []

**for** n\_nodes **in** range(2, nmax + 1):

xlist.append(n\_nodes)

slists = [] wlists = [] ulists = [] dlists = [] graphs = {}

**for** n\_nodes **in** range(2, nmax + 1):

graphs[n\_nodes] = {}

slist = [] wlist = [] ulist = [] dlist = [] print(**'\nn ='**, n\_nodes)

print(**'1)'**) **for** radius\_deviation\_times\_10 **in** range(2, 9):

radius\_deviation = radius\_deviation\_times\_10 / 10

print(**'c ='**, radius\_deviation) strongly\_connected = 0 weakly\_connected = 0 unilaterally\_connected = 0 disconnected = 0 **for** it **in** range(0, total\_iterations): nodes = generate\_nodes(area\_radius, n\_nodes, **True**, radius\_deviation) graph = graph\_from\_nodes(nodes)

serialized = graph.serialize() **if** serialized **in** graphs[n\_nodes]: graphs[n\_nodes][serialized] += 1 **else**:

graphs[n\_nodes][serialized] = 1

**if** graph.is\_strongly\_connected():

strongly\_connected += 1 **if** graph.is\_weakly\_connected():

weakly\_connected += 1 **if** graph.is\_unilaterally\_connected():

unilaterally\_connected += 1 **if not** graph.is\_connected():

disconnected += 1

strongly\_pct = strongly\_connected / total\_iterations \* 100 weakly\_pct = weakly\_connected / total\_iterations \* 100 unilaterally\_pct = unilaterally\_connected / total\_iterations \* 100 disconnected\_pct = disconnected / total\_iterations \* 100

print(**'Strongly connected:'**, strongly\_pct, **'%'**) print(**'Weakly connected:'**, weakly\_pct, **'%'**) print(**'Unilaterally connected:'**, unilaterally\_pct, **'%'**) print(**'Disconnected:'**, disconnected\_pct, **'%'**)

slist.append(strongly\_pct) wlist.append(weakly\_pct) ulist.append(unilaterally\_pct) dlist.append(disconnected\_pct) print(**'2)'**)

print(**'c ='**, 1)

strongly\_connected = 0 weakly\_connected = 0 unilaterally\_connected = 0

disconnected = 0

**for** it **in** range(0, total\_iterations): nodes = generate\_nodes(area\_radius, n\_nodes, **False**) graph = graph\_from\_nodes(nodes) serialized = graph.serialize() **if** serialized **in** graphs[n\_nodes]: graphs[n\_nodes][serialized] += 1 **else**:

graphs[n\_nodes][serialized] = 1 **if** graph.is\_strongly\_connected():

strongly\_connected += 1 **if** graph.is\_weakly\_connected():

weakly\_connected += 1 **if** graph.is\_unilaterally\_connected():

unilaterally\_connected += 1 **if not** graph.is\_connected():

disconnected += 1

strongly\_pct = strongly\_connected / total\_iterations \* 100 weakly\_pct = weakly\_connected / total\_iterations \* 100 unilaterally\_pct = unilaterally\_connected / total\_iterations \* 100 disconnected\_pct = disconnected / total\_iterations \* 100

print(**'Strongly connected:'**, strongly\_pct, **'%'**) print(**'Weakly connected:'**, weakly\_pct, **'%'**) print(**'Unilaterally connected:'**, unilaterally\_pct, **'%'**)

print(**'Disconnected:'**, disconnected\_pct, **'%'**)

slist.append(strongly\_pct) wlist.append(weakly\_pct) ulist.append(unilaterally\_pct)

dlist.append(disconnected\_pct)

slists.append(slist) wlists.append(wlist) ulists.append(ulist) dlists.append(dlist)

slists = list(map(list, zip(\*slists))) wlists = list(map(list, zip(\*wlists))) ulists = list(map(list, zip(\*ulists))) dlists = list(map(list, zip(\*dlists)))

print(**'\nНаиболее вероятные орграфы:'**)

**for** n\_nodes **in** range(2, nmax + 1): print(**'p ='**, n\_nodes, **','**, end=**''**) m = 0 mi = 0 **for** serialized, count **in** graphs[n\_nodes].items(): **if** count > m: m = count

mi = serialized

g = Graph(Graph.deserialize(mi, n\_nodes)) print(m / 8 / total\_iterations \* 100, **'%'**) g.print() gr = networkx.DiGraph() **for** i **in** range(0, len(g.matrix)): **for** j **in** range(0, len(g.matrix[i])): **if** g.matrix[i][j] == 1: gr.add\_edge(str(i+1), str(j+1)) **if** gr.number\_of\_nodes() == 0: gr.add\_nodes\_from(list(range(1, len(g.matrix) + 1)))

networkx.draw\_networkx(gr, with\_labels=**True**, label=str(m / 8 / total\_iterations \* 100) +

**'%'**)

plt.title(**'Наиболее вероятный при n = '** + str(n\_nodes) + **', '** + str(m / 8 / total\_iterations

* 100) + **'%'**)

plt.show()

print(**'q ='**, mi, **'-'**, m / 8 / total\_iterations \* 100, **'%'**)

print(**'\nНаименее вероятные орграфы:'**) **for** n\_nodes **in** range(2, nmax + 1): print(**'p ='**, n\_nodes, **', '**, end=**''**)

* 1. = 99999999
  2. = 99999999

**for** serialized, count **in** graphs[n\_nodes].items():

**if** count < m: m = count

mi = serialized

g = Graph(Graph.deserialize(mi, n\_nodes)) print(m / 8 / total\_iterations \* 100, **'%'**) g.print() gr = networkx.DiGraph() **for** i **in** range(0, len(g.matrix)):

**for** j **in** range(0, len(g.matrix[i])): **if** g.matrix[i][j] == 1:

gr.add\_edge(str(i+1), str(j+1)) **if** gr.number\_of\_nodes() == 0: gr.add\_nodes\_from(list(range(1, len(g.matrix) + 1)))

networkx.draw\_networkx(gr, with\_labels=**True**, label=str(m / 8 / total\_iterations \* 100) +

**'%'**)

plt.title(**'Наименее вероятный при n = '** + str(n\_nodes) + **', '** + str(m / 8 / total\_iterations

* 100) + **'%'**)

plt.show()

print(**'q ='**, mi, **'-'**, m / 8 / total\_iterations \* 100, **'%'**)

print(**'q ='**, mi **if** mi != 999 **else** 0, **'-'**, m / 8 **if** m != 999 **else** 0, **'%'**)

print(**'\nВероятности появления орграфов:'**) **for** n\_nodes **in** range(2, nmax2 + 1):

print(**'p ='**, n\_nodes)

**for** serialized, count **in** graphs[n\_nodes].items(): g = Graph(Graph.deserialize(serialized, n\_nodes)) print(count / 8 / total\_iterations \* 100, **'%'**) g.print() gr = networkx.DiGraph() **for** i **in** range(0, len(g.matrix)): **for** j **in** range(0, len(g.matrix[i])): **if** g.matrix[i][j] == 1: gr.add\_edge(str(i + 1), str(j + 1)) **if** gr.number\_of\_nodes() == 0: gr.add\_nodes\_from(list(range(1, len(g.matrix) + 1)))

networkx.draw\_networkx(gr, with\_labels=**True**, label=str(count / 8 / total\_iterations \*

100) + **'%'**)

plt.title(str(count / 8 / total\_iterations \* 100) + **'%'**) plt.show()

cm = [**'0.8'**, **'0.7'**, **'0.6'**, **'0.5'**, **'0.4'**, **'0.3'**, **'0.2'**] cp = [**'1.2'**, **'1.3'**, **'1.4'**, **'1.5'**, **'1.6'**, **'1.7'**, **'1.8'**]

pylab.figure(1) pylab.subplot(211)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.title(**'Вероятность сильно связного орграфа'**)

**for** i **in** range(0, len(slists)-1):

pylab.plot(xlist, slists[i], label=**'$R \\in [{a}, {b}]$'**.format(a=cm[i], b=cp[i])) pylab.legend(loc=**'best'**)

pylab.subplot(212)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**) pylab.plot(xlist, slists[len(slists) - 1], **'k--'**, label=**'$R = {x}$'**.format(x=1)) pylab.legend(loc=**'best'**)

pylab.figure(2) pylab.subplot(211)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.title(**'Вероятность слабо связного орграфа'**) **for** i **in** range(0, len(wlists)-1):

pylab.plot(xlist, wlists[i], label=**'$R \\in [{a}, {b}]$'**.format(a=cm[i], b=cp[i])) pylab.legend(loc=**'best'**)

pylab.subplot(212) pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.plot(xlist, wlists[len(wlists) - 1], **'k--'**, label=**'$R = {x}$'**.format(x=1)) pylab.legend(loc=**'best'**)

pylab.figure(3) pylab.subplot(211)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.title(**'Вероятность односторонне связного орграфа'**) **for** i **in** range(0, len(ulists)-1): pylab.plot(xlist, ulists[i], label=**'$R \\in [{a}, {b}]$'**.format(a=cm[i], b=cp[i])) pylab.legend(loc=**'best'**)

pylab.subplot(212)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.plot(xlist, ulists[len(ulists) - 1], **'k--'**, label=**'$R = {x}$'**.format(x=1)) pylab.legend(loc=**'best'**)

pylab.figure(4) pylab.subplot(211)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.title(**'Вероятность несвязного орграфа'**) **for** i **in** range(0, len(dlists)-1): pylab.plot(xlist, dlists[i], label=**'$R \\in [{a}, {b}]$'**.format(a=cm[i], b=cp[i])) pylab.legend(loc=**'best'**)

pylab.subplot(212)

pylab.xlabel(**'Количество узлов сети'**) pylab.ylabel(**'Вероятность, %'**)

pylab.plot(xlist, dlists[len(dlists) - 1], **'k--'**, label=**'$R = {x}$'**.format(x=1)) pylab.legend(loc=**'best'**)

pylab.show()

**if** \_\_name\_\_ == **"\_\_main\_\_"**: main()