Санкт–Петербургский государственный университет

Бусел Владислав Дмитриевич

Выпускная квалификационная работа

Анализ свойств сложных сетей при наличии коалиционных структур

Уровень образования: бакалавриат
Направление 02.03.02
«Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2017
«Программирование и информационные технологии»
Профиль «Автоматизация научных исследований»

Научный руководитель:
профессор, кафедра теории игр и статистических решений,
д.ф. - м.н. Громова Екатерина Викторовна

Рецензент:
преподаватель, университет Стирлинга,
Phd Кирпичникова Анна Сергеевна

Санкт-Петербург
2021 г.

Содержание

Введение

В данной работе рассматривается задача оптимизации передачи информации между группами в самоорганизующихся сетях. Задача решена при использовании теоретико-игрового подхода.

Под самоорганизующейся сетью подразумевается MANET (Mobile Adhoc Network) [1] — беспроводная самоорганизующаяся децентрализованная сеть, состоящая из мобильных узлов, способных устанавливать и поддерживать соединения между узлами. Она не зависит от ранее существующей инфраструктуры, например, от маршрутизаторов в проводных сетях или от точек доступа в инфраструктурных беспроводных сетях. Маршрутизаторы могут свободно перемещаться и организовываться произвольно, из-за этого беспроводная топология сети может изменяться быстро и непредсказуемо. В данном виде сетей каждый узел участвует в маршрутизации путем пересылки данных для других узлов, поэтому определение узлов, пересылающих данные, производится динамически на основе сетевого подключения и используемого алгоритма маршрутизации.

MANET широко применяется во многих проектах. Он хорошо подходит для ситуаций, когда при организации сетей нет необходимости или возможности использовать глобальную сеть. Существует ряд задач, которые требуют быстрого развертывания сети, но при этом нет необходимости в крупной и сложной сети, например, в чрезвычайных ситуациях: при действиях спасательных групп в горах или других труднодоступных местностях; при землетрясении и т. д. Для таких ситуаций при разворачивании сети используется данный вид сети.

В структуре сетей часто можно выделить некоторые скопления узлов, в которых узлы между собой имеют более качественную и устойчивую к изменениям связь в сравнении со связью между узлами разных скоплений. При исчезновении случайной связи в сети вероятность того, что качество связи ухудшится между скоплениями, или по-другому кластерами, гораздо выше,

чем внутри них. Для того чтобы улучшить передачу данных между узлами из разных кластеров, стоит построить более устойчивую связь между ними, когда между скоплениями узлов есть связь. Если связей между ними нет, то может возникнуть необходимость установить связь между ними.

Имеем, что устройства внутри кластера имеют довольно хорошую связь друг с другом, в то время как связь между кластерами слабая или отсутствует в целом. Например, возникают ситуации, когда несколько организаций должны объединиться для достижения общей цели. Внутри каждой организации устройства имеют связь друг с другом, но, несмотря на необходимость поддержания сети между ее участниками, устройства одной организации не имеют возможности передать нужную информацию другой из-за того, что у устройств разных организаций могут различаться настройки сети. В данной работе для установки связи между кластерами или ее улучшения предлагается использовать специальные устройства, которые могут устанавливать связь со всеми организациями, таким образом обеспечивая связь между ними. Далее обычные устройства будем называть агентами, специальные устройства – дронами, организации — игроками.

В работе рассматривается случай, когда между организациями нет связи, при таком обстоятельстве можно очень просто выделить кластер каждого игрока. Из-за этого в данной работе не рассматривается вопрос, каким способом их лучше определять. Таким образом, возникает модель задачи, в основе которой лежит сетевая структура в виде графа. В его вершинах находятся агенты, принадлежащие различным игрокам. Каждый агент сети принадлежит одному из игроков. Агенты могут построить связь только с агентами того же игрока или с дронами, дроны могут взаимодействовать со всеми. Создание связи между игроками происходит с помощью добавления в граф дронов.

В задаче просматривается теоретико-игровой подход в следствии следующих оснований: во-первых, в задаче организации представляют собой игроков, которые могут выбрать позиции дронов. Во-вторых, они могут выбрать различные позиции, значит у них есть возможность оперировать стратегиями.

В-третьих, для выбора оптимальных позиций есть возможность использовать функцию выигрыша игрока, которая будет отображать качество установленной связи между игроками. Для объединения подсетей, игрокам имеет смысл согласовать свои действия, а не действовать поодиночке. Из этого вытекает использование кооперативной структуры в данной задаче. В итоге игроки образуют коалиционную структуру для улучшения передачи информации между друг другом.

В данной работе сформулирована математическая модель для описанной задачи, предложен алгоритм ее решения и проведен анализ различных вариаций сетей с дронами, включая результаты моделирования в симуляторе сетей ns-3 [2]. Вариации сетей образуют связи с разными наборами агентов в каждом случае, и некоторых характеристик графов, соответствующих этим сетям.

Основной целью работы является составление алгоритма, который объединяет обособленные сети в единую сеть с помощью добавления дополнительных узлов, используя теоретико-игровой подход. Набор позиций дронов, который предлагает алгоритм, должен обеспечить лучшие характеристики сети в сравнении с большинством других возможных вариантов. Далее сформулируем задачи, и опишем требования к ожидаемому решению.

Постановка задачи

Как писалось ранее, основной целью работы является составление алгоритма, объединяющего обособленные сети в единую сеть с помощью добавления дополнительных узлов. В связи с поставленной целью работы возникают следующие задачи:

1. Формально описать MANET с использованием теории графов. Сеть должна представляться в виде графа, обладающего необходимыми особенностями сети, такими как положение узлов сети в пространстве и ограниченное расстояние для возможности взаимодействия узлов.

2. Сформулировать кооперативную игру между игроками, в которой они объединяют свои сети оптимальным образом. Под оптимальностью в игре подразумевается решение, дающее наибольший выигрыш. Функция выигрыша должна выражать степень качества связи между игроками. Проанализировать характеристики сетей с различными вариантами расположения дронов в них, чтобы установить качество полученного решения.

3. Описать и программно реализовать алгоритм решения поставленной кооперативной игры. Данный алгоритм должен находить решение, дающее наибольший выигрыш из возможных вариантов. Также следует найти другие варианты для анализа.

4. Провести моделирование полученного решения игры в симуляторе сетей ns-3 и сравнить результат с другими возможными вариантами. Для этого нужно изучить ns-3 и разобраться в устройстве MANET.

Обзор литературы

В [3] описаны самые основные и актуальные направления теории игр: конечные и бесконечные антагонистические игры, бескоалиционные и кооперативные игры, многошаговые и дифференциальные игры. Книга использовалась при формулировке кооперативной однократной игры.

В [4] рассматриваются и структурируются современные тенденции в играх в сетях и сетевых играх. Также в статье вводится определенная классификацая таких игр с точки зрения теории игр и теории графов. Данная статья использовалась при написании работы для выбора постановки игры и подхода ее решения.

В [5] введена постановка некооперативной многошаговой игры управления агентами в сети. В многошаговой игре игроки ходят по очереди, пока игра не закончится. Выигрыш каждого игрока зависит от диаметра подграфа, определенного на вершинах, принадлежащих игроку, сформулировано условие существования абсолютного равновесия по Нэшу. Некоторые результаты данного исследования нашли отражение в [6, 7]. В [6] рассматриваются различные варианты сетки: прямоугольная, треугольная и шестиугольная. В [7] показано, что существование равновесия по Нэшу не гарантировано и сформулированы некоторые условия его существования и рассмотренно два класса многошаговых игр: последовательные и одновременные. При моделировании в симуляторе сетей ns-3 выясняется, что рассмотренные методы дают значительное улучшение работы сети.

В [8] задачей является оптимизация сети внутри компонент связности при помощи добавления новых узлов в сеть. В работе предлагается сформулировать задачу поиска местоположения для дронов в виде однократной игры, а в качестве принципа оптимальности использовать оптимальность по Парето.

В [9, 10] предлагается алгоритм двухуровневой оптимизации, который позволяет сначала решить задачу в кооперативной постановке однократной

игры, а затем отобрать из множества оптимальных решений некоторое подмножество согласно алгоритму ранжирования вершин PageRank.

В [11, 12] рассматривается особенности применения теоретико-игрового подхода при моделировании сетевых коммуникационных структур. В книге [12] присутствует описание меры центральности через вектор Майерсона.

В [13] подробно описаны разные коммуникационные сети, моделирование этих сетей с помощью графов, алгоритмы на них, особенности генерации случайных сетей, а также сетевые метрики и процессы. Книга использовалась при составлении решения задачи оптимизации сети, а также при анализе сети.

В [14] описана беспроводная самоорганизующаяся Ad hoc сеть, описаны особенности маршрутизации в такой сети, представлен обзор и сравнение протоколов маршрутизации, использующихся в этих сетях.

[15] посвящена изучению сетей MANET. В ней выделены ключевые особенности сети, рассмотрено применение MANET в современных технологиях и различных областях, а также проведено практическое исследование организации MANET сети на базе работы [9] с использованием симулятора NS-3.

В [16] строится некооперативная и кооперативная игра на шестиугольной, прямоугольной и треугольной решетках. Предложены подходы для нахождения оптимального расположения дронов.

В [17] используются меры центральности для решения проблемы оптимального позиционирования беспилотного летательного аппарата, улучшенающего работу мобильной сети ad hoc. Показано, что для двух конкретных сетей большинство критериев дают одно и то же решение, демонстрируя тем самым хорошую согласованность в своих прогнозах.

[18] содержит документацию API ns-3, организованную по различным модулям. Документация содержит описание функций, классов и прочих ин-

струментов для взаимодействия с ns-3, а также множество скриптов для моделирования различных сетей.

В данной работе используется теоретико-игровой подход, но в отличие от рассмотренных работ рассматривается задача объединения компонент связности графа сети и оптимизации передачи информации между этими компонентами, узлы графа сети расположены в декартовом двумерном пространстве, а не на сетке, также предлагается способ поиска позиций для установки дрона, который не использует сетку.

Глава 1. Математическая модель

Сеть представляется в виде несвязного графа, вершины которого расположены в некотором пространстве. Граф состоит из подграфов игроков, которые нужно соединить друг с другом оптимальным способом. Некоторые параметры модели можно выбрать в зависимости от условий. Например, можно выбрать пространство, в которых расположены агенты, или функцию выигрыша, обладающие некоторыми желаемыми свойствами. При решении задачи за пространство, в котором расположены вершины графа, обычно берется декартово двумерное пространство. Есть ряд причин для выбора двумерного пространства: простота работы в таком пространстве и отсутствие необходимости в пространстве большей размерности.

Сеть рассматривается в конкретный момент времени, поэтому можно считать, что агенты неподвижны. Это могло быть довольно серьезным ограничением в связи использования MANET, так как его ключевой особенностью является подвижность узлов. Предполагается, что задача решается в определенный промежуток времени, для динамической сети можно находить решение с определенным временным интервалом. Следовательно, алгоритм должен быть достаточно быстрым, чтобы успевать посчитать ответ за ограниченный промежуток времени.

Представим формальное описание игры, ее математическую модель и несколько вариантов функции выигрыша.

1.1 Базовые элементы игры

Определим необходимые для постановки задачи объекты. Пусть задано множество игроков $P = \{1, \ldots, n\}$. Каждый игрок $i \in P$ имеет непустое множество $M_i \neq \varnothing$ агентов, расположенных в множестве X. Множество всех агентов обозначим $M = \bigcup\limits_{i \in P} M_i$. Дронами назовем дополнительных агентов, которые могут быть расположены в $\overline{X} \subset X$. Множество дронов игрока $i \in P$ обозначим $\overline{M_i}$. Они образуют множество дронов $\overline{M} = \bigcup\limits_{i \in P} \overline{M_i}$, $\overline{M} \neq \varnothing$. Обо-

значим функцию расстояния $\rho : X \times X \longrightarrow Y$, где Y — множество значений функции расстояния.

В данной работе используется $\overline{X} = X = \mathbb{R}^2$, ρ — евклидова метрика, представленная в виде (1). В ней n — размерность пространства X, $x, y \in X$. В работе рассмотрена сеть, которая не меняется со временем.

$$\rho(x, y) = \sqrt{\left(\sum_{i=1}^{n} (x_i - y_i)^2 \right)}. \tag{1}$$

1.2 Сетевая структура

Представим сеть в виде графа в пространстве X. Определим его вершины как $v = (a, x)$, где каждому $a \in M$ соответствует единственная $x \in X$. Множество всех вершин обозначим V^*. Между вершинами игрока i v_p^i и v_s^i, где $s \neq p$, установлена устойчивая связь в виде ребра (v_p^i, v_s^i), если расстояние между ними не больше определенной величины $dist_{max}$. Это связано с тем фактом, что узлы сети имеют определенную дальность действия. В связи с этим появляется необходимость модифицировать граф под задачу. Зададим множество ребер функцией $\delta(v_i, v_j)$ (2), которая определяет, есть ли ребро между узлами.

$$\delta(v_i, v_j) = \begin{cases} 1, & \rho(x_i, x_j) \leqslant dist_{max}, \\ & (a, x_i) = v_i, (a, x_j) = v_j, v_i, v_j \in V^*; \\ 0, & \text{иначе.} \end{cases} \tag{2}$$

Таким образом определим граф $G = (V, \delta, \rho)$, $V = \{v : v = (a, x) \in V^*, a \in M \setminus \overline{M}\}$. Подграфом игрока i назовем граф, содержащий в себе всех агентов этого игрока $G_i = (V_i, \delta, \rho) \subset G$. Стоит отметить, что ранее определенный граф не имел дронов. Граф с дронами обозначим через $\overline{G}$. Граф с установленными дронами на определенные позиции назовем расширенным графом G^*.

1.3 Кооперативная игра

Как ранее упоминалось, в поставленной задаче имеет смысл использовать кооперативную структуру, так как объединившись, игроки могут достичь лучших результатов, в сравнении с ситуациями, когда каждый игрок действует самостоятельно. Напомним, что суть игры заключается в установке связи между игроками, поэтому выигрыш должен отображать, насколько установленная связь хорошая.

Представим, что у нас есть функция выигрыша игрока, которая может отобразить качество соединения с остальными игроками. Определение вида функции выигрыша для игрока в данной задаче будет приведено ниже. Обозначим ее как H_i. Стратегией игроков является выбор позиций, в которые будут поставлены дроны. Так как рассматривается кооперативная игра, определим общий выигрыш $H = \sum\limits_{i \in P} H_i$.

Задача состоит в поиске оптимального местоположения дронов, при котором суммарный выигрыш игроков максимизируется.

1.4 Функции выигрыша для кооперативной игры

Для решения игры необходимо определить функцию выигрыша. Стоит уделить внимание ее выбору и рассмотрению альтернативных вариантов, так как в зависимости от выбранной функции может меняться не просто решение игры, но и аспекты, на которые будет делаться акцент. Для данной задачи важными аспектами выделены длина пути между агентами разных игроков и устойчивость сети при удалении случайных агентов. Первый критерий исходит из соображения скорости передачи информации в сети. Очевидно, что чем меньше длина пути между узлами, тем быстрее можно передать информацию. Второй аспект исходит из особенностей MANET. Сеть динамически меняется и любой узел может выйти из сети. Если не учитывать этот факт, то может возникнуть ситуация, в которой выход одного узла может оборвать связь между игроками. Некоторые меры центральности удовлетворяют поставленным требованиям. Рассмотрим меры центральности по степени связ-

ности, посредничества, близости и через вектор Майерсона [12].

Центральность через степень посредничества [13] зависит от количества кратчайших путей, проходящих через вершину: чем их больше, тем больше значение меры в данной вершине. Формула такой меры выглядит следующим образом (3)

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{3}$$

где σ_{st} — общее число кратчайших путей из узла s в узел t, а $\sigma_{st}(v)$ равно числу этих путей, проходящих через v.

В основе меры через вектор Майерсона лежит идея использовать методы кооперативной теории игр для определения центральности вершин в графе. В целях объяснения идеи данной меры рассмотрим игру с ограниченной кооперацией.

Пусть вершины графа представляют игроков, а ребра представляют связь между ними. Игроки могут взаимодействовать напрямую, только если они связаны. Это приводит к коммуникационной (сетевой) игре [4], заданной конечным множеством игроков, характеристической функцией и графом [12]. Хорошо известным решением сетевой игры является вектор Майерсона, который характеризуется компонентной эффективностью и справедливостью. Эффективность означает, что для каждого компонента графа общая выплата игрокам равна ценности этого компонента. Понятие справедливости означает, что удаление связи между двумя игроками приводит обоих к одинаковым изменениям к выплате.

Общая идея заключается в том, что для невзвешенных графов каждый кратчайший путь длины k прибавляет к выигрышу игрока величину $\frac{1}{k}$. То есть один кратчайший путь имеет ценность 1, и узлы, участвовавшие в построении данного маршрута, делят этот выигрыш поровну. Исходя из этого меру центральности через вектор Майерсона можно вычислить следующим

образом:

$$Y(v,x) = \sum_{k=1}^{L} \frac{a_k^v}{k+1} x^k,$$

где a_k^v — число всех кратчайших путей длины k, содержащих вершину v в пути, x — параметр, отвечающий за значимость путей большей длины в сравнении с более короткими путями, $0 < x \leqslant 1$.

Для того, чтобы можно было использовать данную меру при $x > 1$, возьмем за измененную меру центральности через вектор Майерсона обратное число значению функции (4)

$$Y_{mod}(v,x) = \frac{1}{Y(v,x)}, \tag{4}$$

при $x > 1$.

Для данной задачи рассматривались другие известные меры центральности [13], такие как мера центральности по степени (degree centrality) и мера центральности по близости (closeness centrality). Значение первой меры (5) для узла — это доля узлов, к которым он подключен. Значения нормализуются путем деления на максимально возможную степень в простом графе n-1, где n — количество узлов в этом графе.

$$C_d(v) = \frac{deg(v)}{n-1}, \tag{5}$$

где $deg(v)$ — степень вершины v, n — число вершин в графе.

Такая мера центральности относительно грубая. В ней не учитывается степень «центральности» других узлов. Кроме того, недостатком меры является то, что количество связей чаще всего не отражает их качества, а просто свидетельствует о степени коммуникации.

Центральность «по близости» (6) показывает насколько быстро инфор-

мация распространяется в сети от одного участника к остальным, другими словами насколько близок рассматриваемый участник ко всем остальным участникам сети.

$$C_c(v) = \frac{n-1}{\sum\limits_{v \in V} d(v,u)}, \tag{6}$$

где n — число вершин в графе, а V — множество вершин в графе. $\frac{1}{d(v,u)} = 0$, если нет пути между v в u.

Данная мера хорошо подходит для данной задачи, так как для того, чтобы иметь высокую степень данного вида центральности, узел должен не только обладать множеством связей, но и у близких к нему вершинам тоже их должно быть достаточно. Это означает, что узел с высокой степенью центральности по близости через те связи, в которые он включен, получает возможность доступа к большому количеству других участников сети.

Ранее мы обозначили функцию выигрыша игрока через H_i. Формула, по которой ее можно вычислить выглядит следующим образом

$$H_i = \sum_{d \in \overline{M}} C(d)$$

$C(d)$ — значение выбранной меры центральности в вершине d расширенного графа G^*.

Рассмотренные меры центральности в дальнейшем используются при вычислении функции выигрыша, дальше в работе будут определены наиболее подходящие варианты для данной задачи.

Глава 2. Алгоритм

2.1 Описание алгоритма решения задачи

Опишем алгоритм решения задачи. В ходе работы изначально рассматривалась сетка. Существует несколько ее разновидностей, чаще всего люди знакомы с квадратной, треугольной и шестиугольной сетками. Использование сеток определенно имеет свои преимущества, например на их основе можно было бы построить простое решение задачи: перебрать все варианты сочетаний узлов сетки, находящихся не дальше $dist_{max}$ хотя бы от одного из имеющихся агентов. Но такое решение имеет довольно существенный недостаток: большое количество вариантов требует больше времени, возникает желание перебирать не все варианты в ограниченных областях сетки вокруг агентов, а сразу найти варианты расположения дронов, которые объединяют подграфы сети. При использовании сетки с большим размером мы можем пропустить хорошие варианты расположения дронов, поэтому чтобы не пропустить такие позиции, нужно настроить размер сетки. Будем использовать квадратную сетку, чтобы найти возможные варианты соединения дрона с сетью. Также составим алгоритм поиска оптимальных позиций дронов на графе, расположенном в двумерном декартовом пространстве. Для данной цели будет использоваться способ поиска позиций для дрона без использования сетки.

Алгоритм должен выбрать позиции для набора дронов, которые устанавливают связь между игроками наиболее хорошего качества. Обозначим требования, которые алгоритм должен выполнять. Во-первых, он должен объединить все подсети в единую сеть, то есть ситуации, где некоторые игроки изолированы от других, недопустимы. Во-вторых, алгоритм должен быть достаточно быстро выполняться, так как сеть динамическая. Алгоритм можно поделить на две части: отбор наборов позиций, объединяющих подграфы, и выбор из них оптимального. Предложенный вариант алгоритма работает в двумерном пространстве.

Сначала найдем позиции, которые могут установить связь с агентами разных игроков. В основе данного этапа лежит процесс вычисления точек пересечения окружностей. Представим агентов в виде точек на плоскости. Они лежат на двумерной плоскости и имеют свои координаты. Образуем наборы, которые содержат в себе вершины. Расстояние между вершинами в наборах не больше $2dist_{max}$. В таких наборах мы точно знаем, что есть пересечения между парами окружностей, и возможно есть пересечение более двух окружностей. Для определенности будем называть их наборами с центрами взаимопересекающихся окружностей, имеется ввиду, что при выборе любых двух точек, окружности, имеющие центры в этих точках, имеют пересечение.

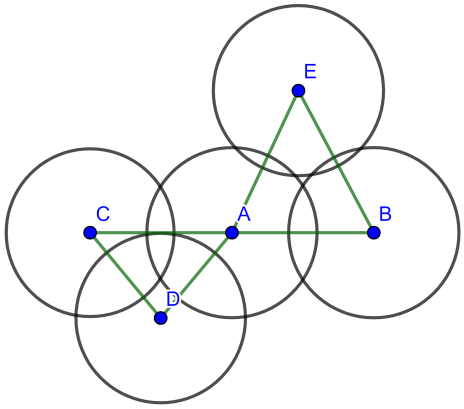

Рис. 1: Узлы, которые могут установить связь с помощью дрона.

Если все агенты имеют одинаковые координаты, то просто выберем эти координаты как позицию для дрона. В этом случае задача не имеет смысла, но все же данный случай нужно было рассмотреть.

Разберем случай, когда хотя бы два агента имеют разные координаты. В такой ситуации найдется хотя бы две окружности, имеющие только две точки пересечения. Для каждого набора с центрами взаимопересекающихся окружностей найдем все точки пересечения окружностей, имеющих центры в разных координатах. Для этого нужно найти пересечения пар окружностей с разными центрами. После вычисления точек пересечения между всеми парами окружностей в наборе мы получаем множество точек пересечений для

данного набора. Обозначим такие множества для каждого набора как B_i, где i — индекс множества. Найти точки пересечения между двумя окружностями с равными радиусами достаточно просто (Рис. 2):

$$AC = BC = dist_{max}$$
$$CD = \sqrt{AC^2 - (\frac{AB}{2})^2}$$
$$r_D = r_A + \frac{r_B - r_A}{2}$$
$$C_x = D_x + \frac{CD}{AB}(y_2 - y_1)$$
$$C_y = D_y - \frac{CD}{AB}(x_2 - x_1)$$
$$E_x = D_x - \frac{CD}{AB}(y_2 - y_1)$$
$$E_y = D_y + \frac{CD}{AB}(x_2 - x_1)$$

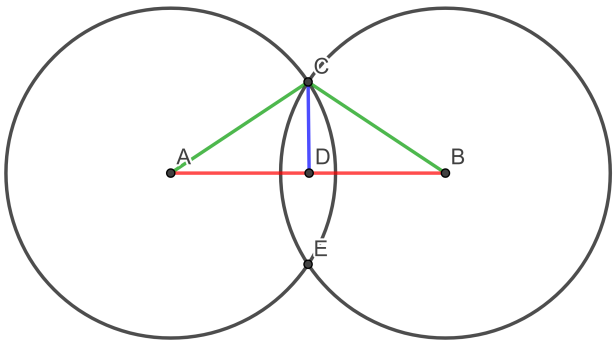

Рис. 2: Пример вычисления точек пересечения.

Для дальнейшего объяснения определим множество доступных агентов для точки $x \in X$

$$A_x = \{a | (a, y) \in V \wedge \rho(x, y) \leqslant dist_{max}\},$$

Определим области, точки внутри которых имеют множество доступных агентов A_j, j — индекс области , которые для множества $A_y, \forall y \in X \setminus A_j$ не является строгим подмножеством $A_j \not\subset A_y$. Для этого отдельно в каждом множестве B_i определим для всех точек $x \in B_i$ множества A_x. То есть сначала находим такие множества для одного множества B_i, потом для следующего. Точки пересечений входят в области, которые мы ищем. Поэтому

найдем точки, у которых множества доступных агентов не являются строгим подмножеством множества доступных агентов для другой точки, и сгруппируем их по этим множествам, точки в одной группе имеют равные множества. Для каждой точки $x \in B_i$ проверяем, нет ли такой точки $y \in B_i, y \neq x$, что $A_x \subset A_y$, если такой точки нет, то добавляем точку x в группу W_j, где j — индекс группы, состоящую из точек с множеством доступных агентов A_j:

$$W_j = \{x | x \in B_i \wedge A_x = A_j \not\subset A_y, \forall y \in B_i\}.$$

Таким образом мы получили группы, с помощью которых можно грубо определить области. Каждая группа определяет свою область. Из области можно взять любую точку, например, в данной работе взята точка, радиус-вектор которой равен усредненной сумме радиус-векторов точек из одной группы (7). В итоге мы нашли потенциальные позиции, которые используем для дальнейшей работы алгоритма.

$$\overrightarrow{r_j} = \frac{1}{|W_j|} \sum_{x \in W_j} \overrightarrow{r_x}. \tag{7}$$

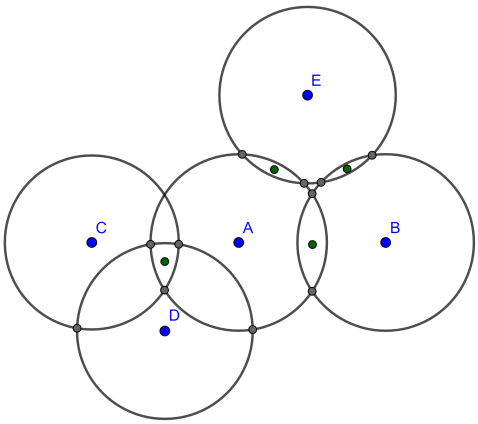

Рис. 3: Потенциальные позиций для дрона (зеленые точки внутри пересечений).

В игре присутствует несколько дронов. Так как игра кооперативная, игроки согласуют свои действия, поэтому нужно выбирать позиции не для каждого дрона по отдельности, а сразу весь набор позиций. Образуем из ра-

нее полученных позиций всевозможные сочетания позиций и оставим только те сочетания, которые объединяют всех игроков в единую сеть.

Далее остается вычислить значение функции выигрыша для каждого найденного набора. Для этого в расширенном графе посчитаем значения функции выигрыша H_i для каждого игрока и просуммируем их, чтобы найти общий выигрыш H. Выберем набор с максимальным общим выигрышем.

Алгоритм можно представить в виде трех основных этапов:

1. Поиск позиций, позволяющих соединить подсети.
2. Сгруппировать позиции во всевозможные наборы, которые объединяют всех игроков.
3. Подсчет суммы значений функции выигрыша в наборах и выбор набора с наибольшим выигрышем.

2.2 Поиск потенциальных позиций при помощи квадратной сетки

В предложенном алгоритме множество потенциальных позиций, у которых мы считаем значение меры центральности, не рассматривает всевозможные варианты графа с дронами, имеющие разное множество ребер. Поэтому для анализа всех возможных вариантов объединения дрона с графом сети будем использовать сетку для поиска потенциальных позиций. Стоит помнить, что для поиска всевозможных вариантов нужно настроить размер сетки.

Для каждой вершины в графе перебираем узлы решетки, расположенные не дальше $dist_{max}$ от этой вершины (Рис. 4). В программной реализации решетка общая для всех вершин. Также существует вариант, когда у каждой вершины имеется своя отдельная решетка, прикрепленная к ней. Для каждого такого узла находим множество доступных агентов, расположенных не дальше $dist_{max}$ от него.

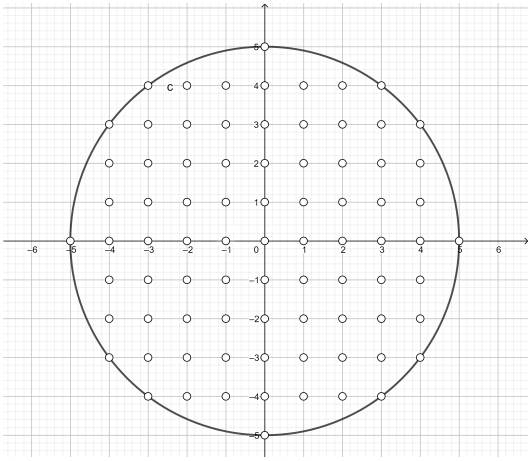

Рис. 4: Пример. Узлы решетки, расположенные не дальше $dist_{max} = 5$ от некоторой вершины.

Если в этом множестве есть агенты двух разных игроков и еще не встречалось узла с равным множеством доступных агентов, то добавляем данный узел к множеству потенциальных позиций. Таким образом можно найти потенциальные позиции дронов (Рис. 5).

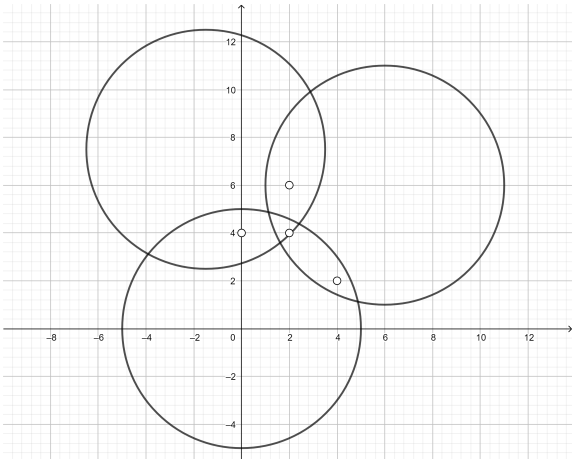

Рис. 5: Пример потенциальных позиций для дрона, найденных с помощью сетки.

Глава 3. Анализ результатов моделирования

В качестве среды для моделирования работы сети был взят ns-3 (Network Simulator 3). Его работа проходит в ограниченном дискретном

промежутке времени, и он является симулятором дискретных событий, каждое событие в нем связано с временем выполнения. Его разработали для того, чтобы обеспечить открытую расширяемую платформу для моделирования сетей, исследований сети и образования. ns-3 предоставляет модели того, как реализованы и работают сети пакетной передачи данных и дает пользователю движок симулятора для проведения имитационных экспериментов. Данный симулятор широко используется как средство проверки для алгоритмов и предлагаемых аналитических моделей, а также для изучения работы больших сетей.

Установленная в ns-3 модель фокусируется на моделировании работы интернет-протоколов и сетей, но в симуляторе также существует возможность использования ns-3 для моделирования таких систем как анализ эпидемиологической обстановки, транспортной системы, поведения людей в обществе и т. д.

Для наглядности моделируемой сети есть возможность ее визуализировать, используя некоторые программные утилиты. NetAnim и ns-3 PyVis являются самыми популярными их них. Оба решения позволяют убедиться в корректности построенной сети, проанализировать состояние узлов, передаваемых и получаемых пакетов.

В ходе работы программы генерируется файл формата flowmon, который использует синтаксис xml и содержит информацию о всех протекающих во время моделирования потоках. Содержащиеся в файле данные можно использовать для анализа сети.

3.1 Преимущества и недостатки MANET

В связи с использованием MANET в данной задаче необходимо разобраться в принципе и особенностях работы данного типа сетей для дальнейшей работы с ним в ns-3. Можно выделить следующие преимущества мобильной ad-hoc сети B [14]:

- Простота и скорость развертывания — сеть развертывается за очень короткий промежуток времени;
- Отказоустойчивость — на случай прекращения функционирования обычно доступных устройств инфраструктуры существует временный резервный механизм;
- Малые затраты на развертывание сети, так как не нужно делать дорогую инфраструктуру для этого.

Также выделим несколько проблем сетей такого типа:

- Требования к отказоустойчивости и качеству обслуживания. MANET должен обеспечивать их в сложных условиях, например, обеспечение минимального уровня обслуживания затрудняется из-за изменяющихся свойств физической линии связи.
- Маршрутизация. Топология сети может меняться со временем, поэтому протокол маршрутизации должен постоянно обновлять маршруты и ссылки. Также маршрутизация может быть поправлена, чтобы решить проблемы потери радиолиний и мобильных устройств.
- Безопасность. Как и остальные беспроводные сети, самоорганизующиеся сети уязвимы к классическим типам атак, тем не менее, они обладают своими особенностями. Из-за отсутствия инфраструктуры неприменимы классические системы безопасности, такие как центральные серверы и центры сертификации. Из-за динамической топологии требуется использовать сложные алгоритмы маршрутизации, которые учитывают вероятность появления скомпрометированной информации от узлов в результате изменения топологии сети. Протоколы маршрутизации могут не иметь четкой картины сети, становится неясно, присоединился какой-то узел к сети или нет, а также является ли присоединенный узел фиктивным или нет. Таким образом, реализация фальсификации в MANET представляется более легкой, чем в сетях других типов.

- Зависимость от автономного питания. Узлы мобильной ad-hoc сети питаются от батарей, что накладывает дополнительные ограничения на устройства, такие как вычислительные возможности узлов, дальность передачи, коммуникационную активность, так как батареи имеют ограниченные емкость и срок службы.

- «Эгоистичность» узлов. Некоторые узлы могут отказаться участвовать в маршрутизации пакетов других узлов, так как питание играет весомую роль. Для выявления и решения таких ситуаций применяются достаточно сложные меры. Одним из вариантов регулирования является отказ пересылки пакетов такого узла другими узлами.

3.2 Протоколы маршрутизации, использующиеся в MANET

Протоколы маршрутизации для мобильных ad-hoc сетей разделяют на три следующих типа B [14]:

  – Проактивные: OLSR, DSDV, GSR и т. д. Они имеют таблицу информации о маршрутизации и узел создает маршруты до того, как в них возникает необходимость;
  – Реактивные: AODV, LAR, DSR и т. д. В них узлы создают маршруты при потребности передачи данных соседнему узлу;
  – Гибридные: ZRP, ZHLS, CEDAR и т. д. Сочетают стратегии проактивных и активных протоколов.

Выбор того или иного вида протокола делается с учётом обстановки и скоростей движения абонентов. Например, для VANET (автомобильной версии MANET) имеет смысл использовать реактивные протоколы.

3.3 Описание моделируемой сети

Чтобы показать состоятельность решений, получаемых с помощью предложенного алгоритма, сравним результаты моделирования некоторых расширенных сетей. Под расширенной сетью подразумевается сеть с установлен-

ными дронами.

Данные передавались от агентов каждого игрока всем агентам других игроков. Использовался проактивный протокол маршрутизации DSDV, так как узлы в рассматриваемой сети неподвижны. Из-за того, что в один момент времени количество потоков может быть довольно большим, некоторые узлы сети могут быть перегружены. Дроны могут стать такими узлами, так как в моделируемой сети мы передаем данные от узлов одного игрока узлам других игроков, а дроны соединяют между собой игроков. Чтобы в сети не возникало такой ситуации, в моделируемой сети в один промежуток времени данные передают не все узлы сразу, а только часть из них. Таким образом получается снизить и привести в норму нагрузку на дроны.

3.4 Примеры

Перед описанием результатов приведем несколько примеров решения задачи. В каждом примере есть два подграфа игроков: один обозначен красным цветом, второй — синим. На столбчатых диаграммах первым столбцом идет результат для позиции, выбранной алгоритмом, остальные позиции соответствуют остальным потенциальным позициям.

Исходные сети, представленные в виде графов, изображены на рисунках 6, 16, 26. Необходимо найти положение для одного дрона.

Промежуточным результатом алгоритма является множество позиций, найденных с помощью пересечения окружностей. Для рассмотрения всех возможных вариантов связи с узлами для дрона используем сетку, так как способ с пересечением окружностей находит наиболее перспективные позиции. Из множества позиций выбирается по одному варианту для каждого возможного набора смежных узлов. Используем сетку для рассмотрения всех возможных вариантов связи с узлами, чтобы продемонстрировать, что алгоритм не пропускает более хорошие решения. Потенциальные позиции, обозначенные белыми кругами с чёрной обводкой, изображены на рисунках 7, 17, 27.

Выбранные алгоритмом позиции для примеров, обозначенные желтым цветом, проиллюстрированы на 8, 18, 28.

В качестве одной из характеристик оценки качества соединения взята средняя длина кратчайшего пути в графе, так как длина пути влияет на время доставки данных. На графиках 9, 19, 29 для каждого примера показана средняя длина кратчайшего пути графа сети для различных позиций дрона.

Другими характеристиками для оценки взяты результаты моделирования в ns-3. На рисунках 10, 11, 20, 21, 30, 31 представлены результаты моделирования в ns-3. Задержка — это время, необходимое для отправки информации из одного узла к другому. Пакеты могут теряться в сети при высокой нагрузке на сеть, то есть с помощью оценки потерянных пакетов можно оценить нагрузку на сеть.

Значения выигрыша при использовании рассмотренных мер центральности можно посмотреть на рисунках 12, 13, 14, 15, 22, 23, 24, 25, 32, 33, 34, 35.

Пример 1.

Сеть, представленная в виде графа, изображена на рисунке 6. Необходимо найти положение для одного дрона.

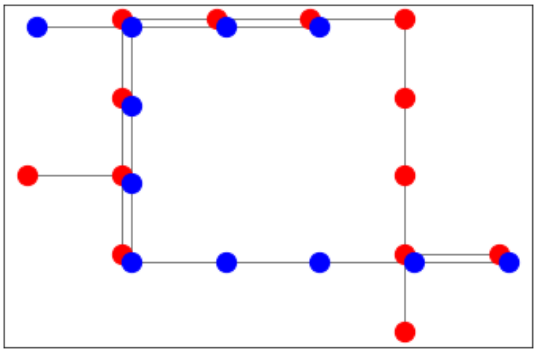

Рис. 6: Пример 1. Исходный граф.

На изображении 7 потенциальные позиции изображены белыми кругами с чёрной обводкой.

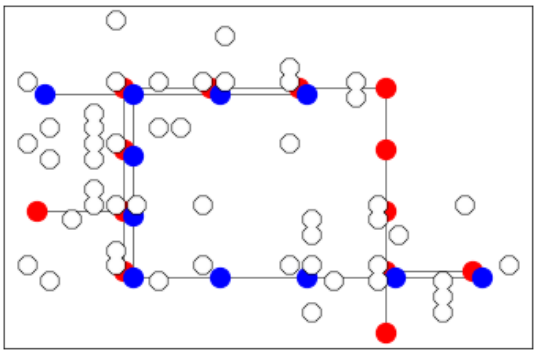

Рис. 7: Пример 1. Граф с потенциальными позициями.

Решение, полученное алгоритмом, продемонстрировано на рисунке 8.

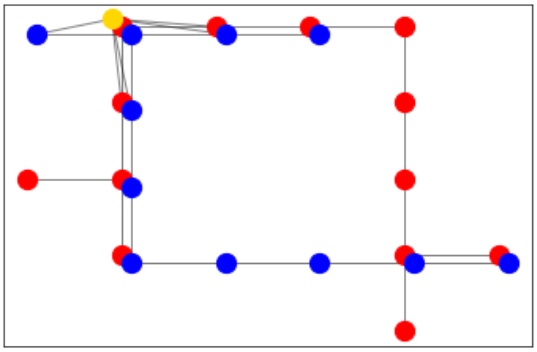

Рис. 8: Пример 1. Решение, найденное алгоритмом.

Одним из критериев сети является средняя длина кратчайших путей в графе сети, так как чем меньше путь, тем быстрее можно передать информацию между узлами. Решение, полученное алгоритмом, имеет наименьшую среднюю длину кратчайших путей в графе в сравнении с другими потенциальными позициями (Рис. 9).

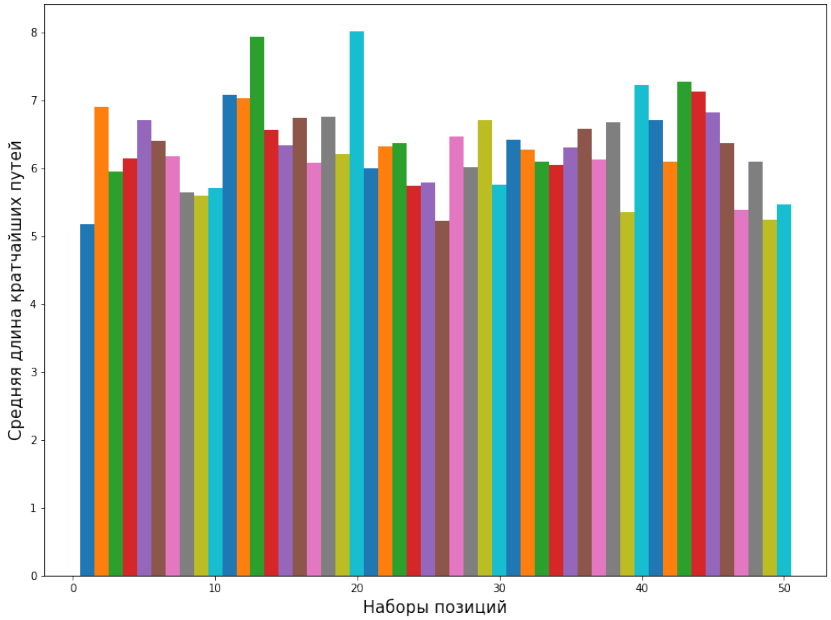

Рис. 9: Пример 1. Средняя длина кратчайших путей графа сети с дроном, расположенном в различных позициях.

При моделировании сети в ns-3 решение дало наименьшую задержку (Рис. 10). Во всех рассмотренных случаях в примерах передавалось одинаковое количесво пакетов, потери пакетов оказались во всех случаях одинаковыми (Рис. 11).

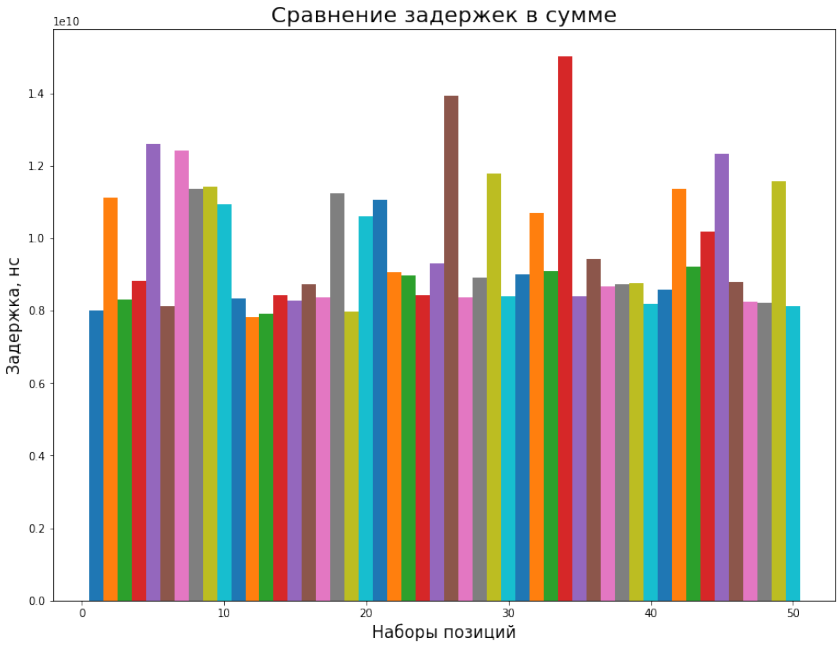

Рис. 10: Пример 1. Сумма задержек для потенциальных позиций.

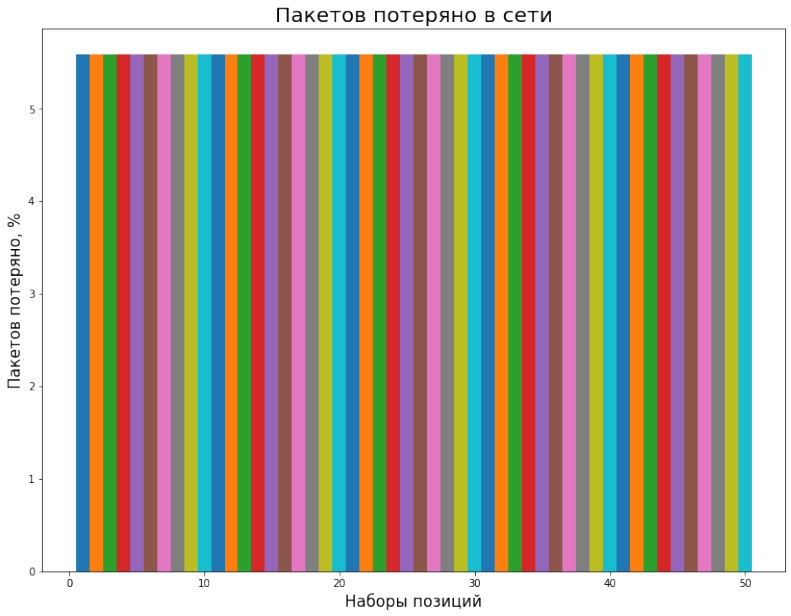

Рис. 11: Пример 1. Процент потерянных пакетов от отправленных для потенциальных позиций.

Далее показаны значения выигрыша при использовании различных мер центральности. Во всех случаях оптимальная позиция, выбранная алгоритмом, дала наибольший выигрыш. В случае с центральностью по степени можно выбирается любой вариант из трех, все три варианта по сравниваемым характеристикам сети дают близкий к оптимальному решению результат.

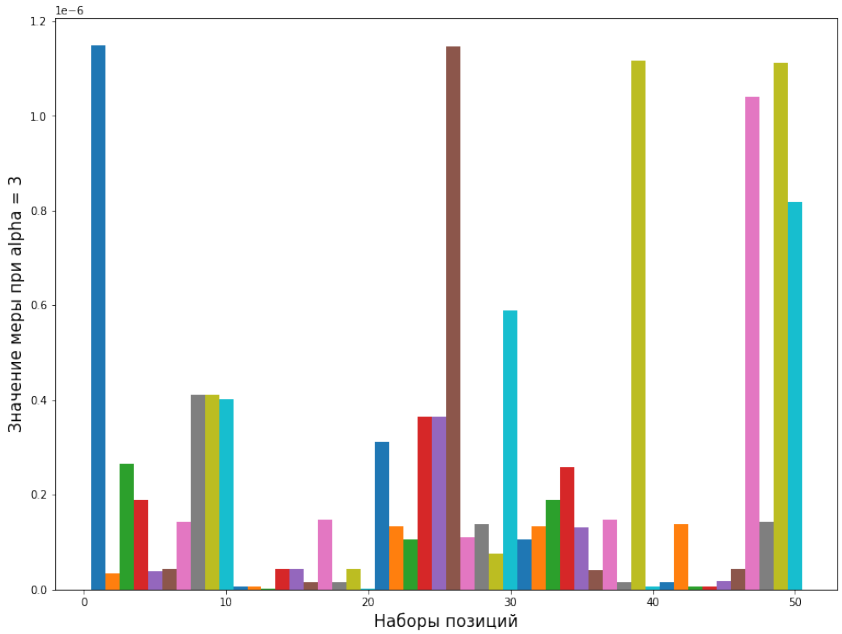

Рис. 12: Пример 1. Значение измененной меры центральности через вектор Майерсона в различных позициях дрона.

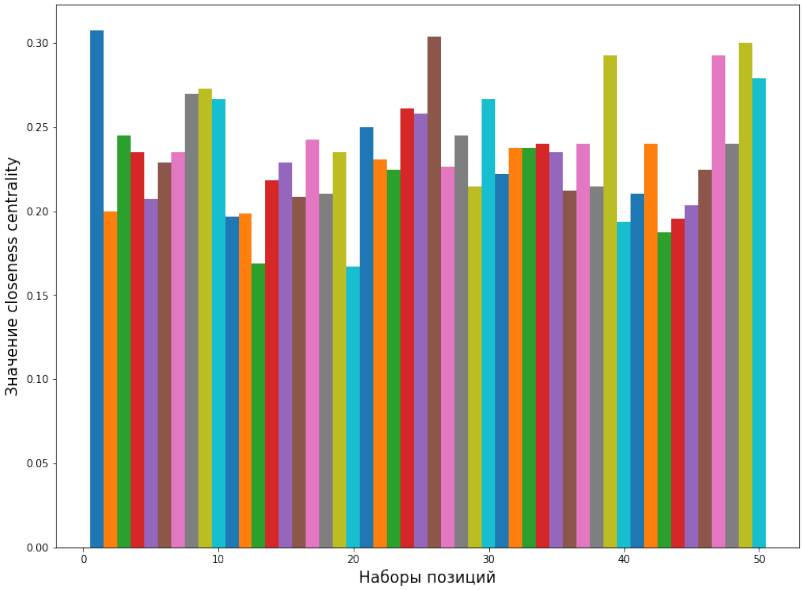

Рис. 13: Пример 1. Значение closeness centrality для дронов, расположенных в потенциальных позициях.

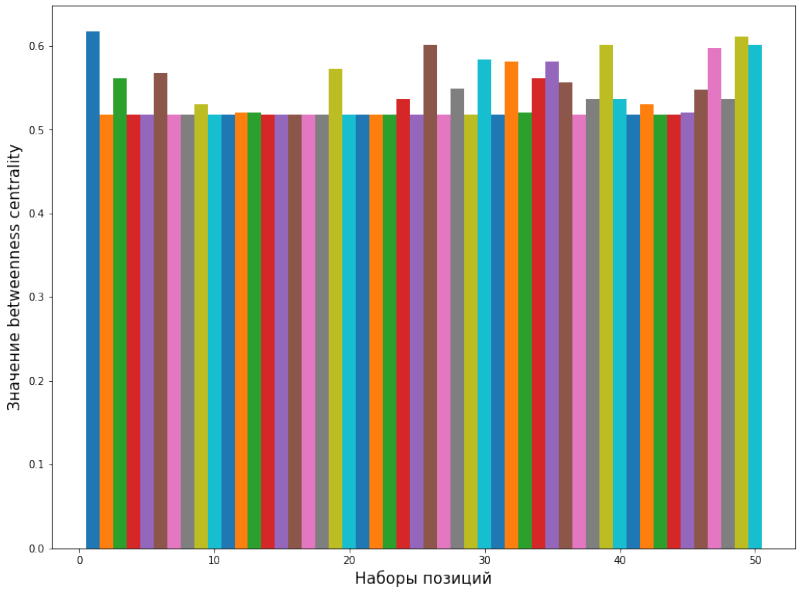

Рис. 14: Пример 1. Значение betweenness centrality для дронов, расположенных в потенциальных позициях.

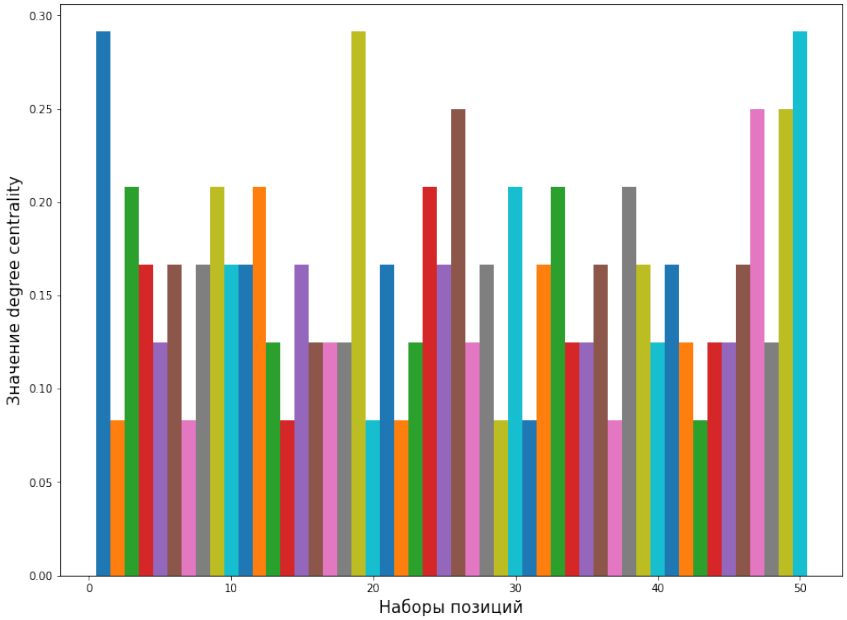

Рис. 15: Пример 1. Значение degree centrality для дронов, расположенных в потенциальных позициях.

Пример 2.

Во втором примере не во всех случаях решения, полученные алгоритмом с использованием мер, дали оптимальный результат. Мера центральности по степени (Рис. 25) может выбрать четвертую позицию, отмеченную красным столбцом. Эта позиция значительно отличается от оптимальной по сумме задержек (Рис. 20) и средней длине кратчайших путей (Рис. 21) в сравнении с другими позициями. Варианты, выбранные остальными мерами центральности, дали схожие результаты, которые практически не отличаются друг от друга.

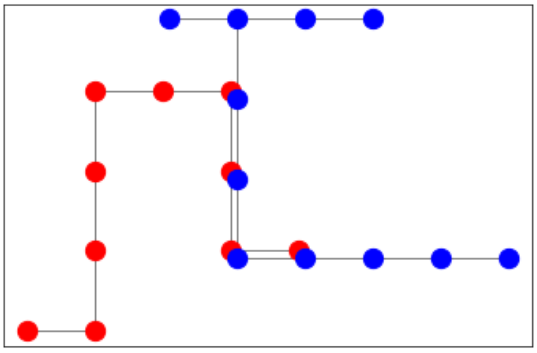

Рис. 16: Пример 2. Исходный граф.

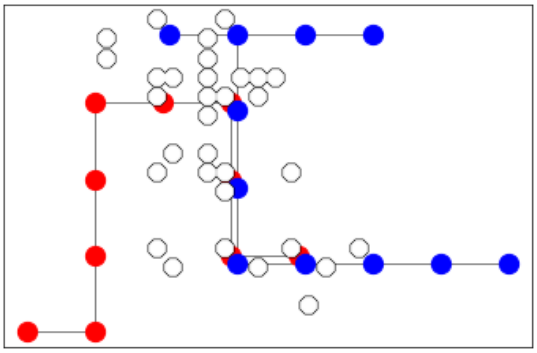

Рис. 17: Пример 2. Граф с потенциальными позициями.

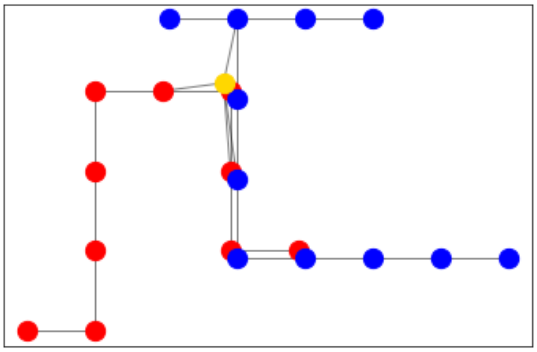

Рис. 18: Пример 2. Решение, найденное алгоритмом.

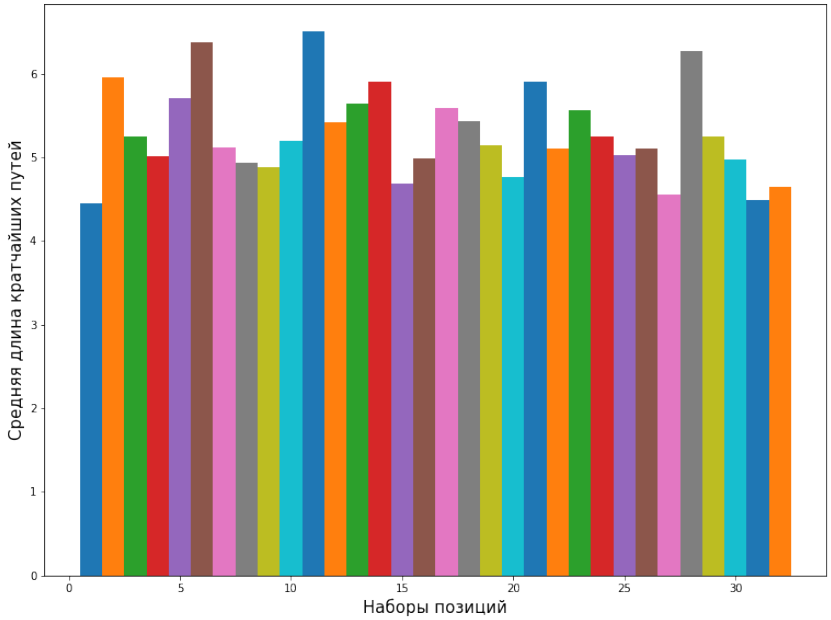

Рис. 19: Пример 2. Средняя длина кратчайших путей графа сети с дроном, расположенном в потенциальных позициях.

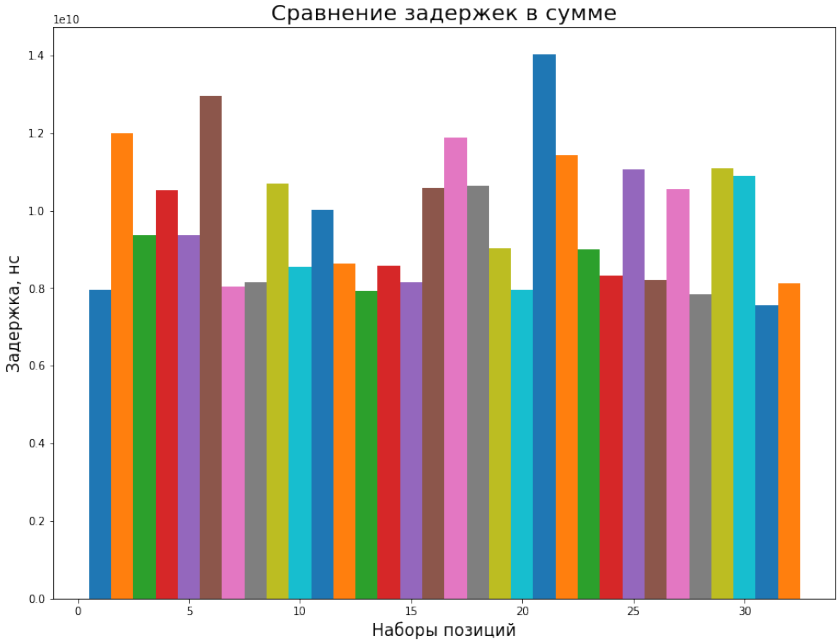

Рис. 20: Пример 2. Сумма задержек для потенциальных позиций.

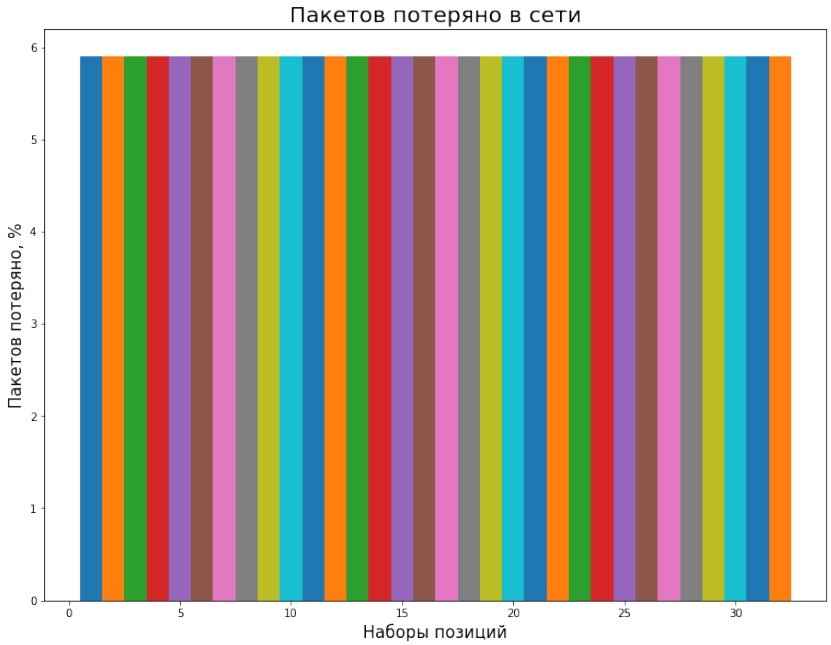

Рис. 21: Пример 2. Процент потерянных пакетов от отправленных для потенциальных позиций.

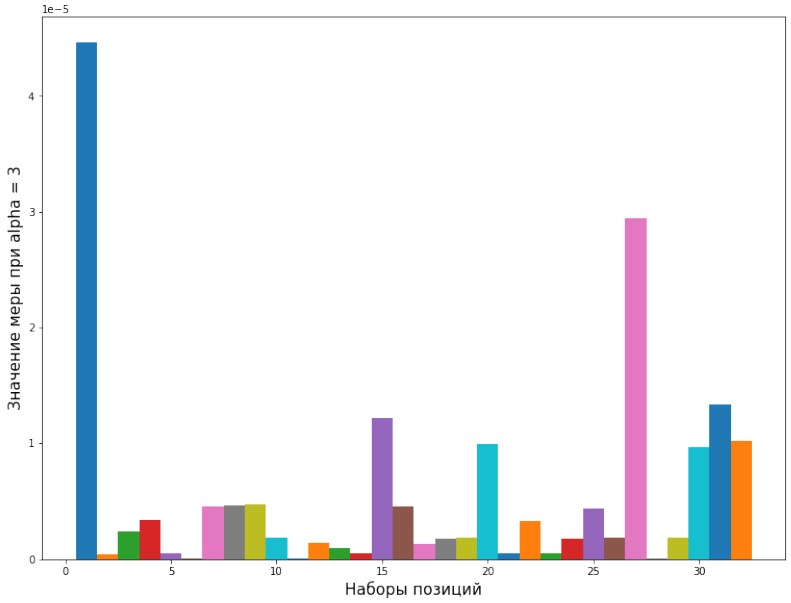

Рис. 22: Пример 2. Значение измененной меры центральности через вектор Майерсона в различных позициях дрона.

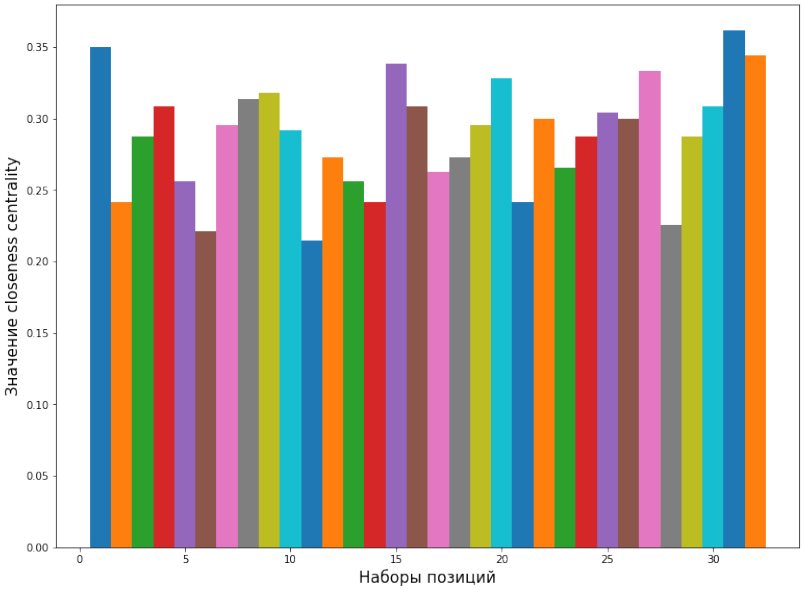

Рис. 23: Пример 2. Значение closeness centrality для дронов, расположенных в потенциальных позициях.

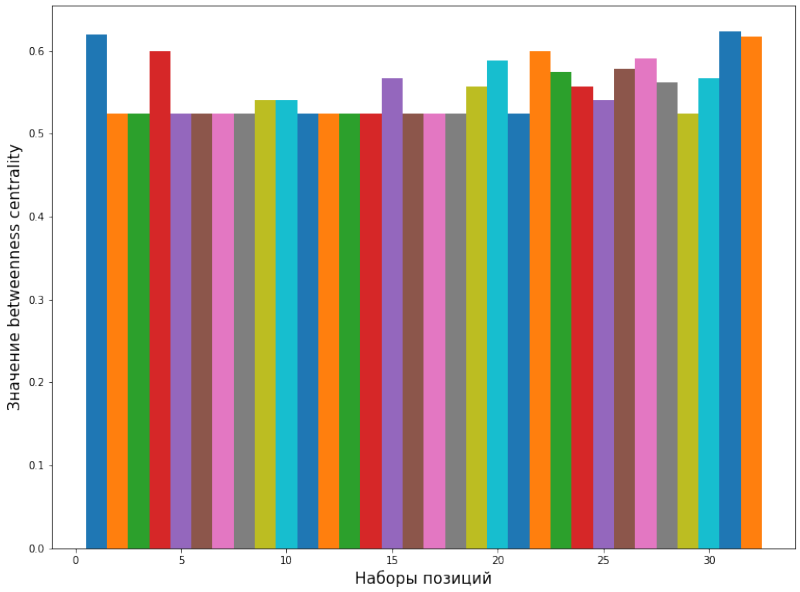

Рис. 24: Пример 2. Значение betweenness centrality для дронов, расположенных в потенциальных позициях.

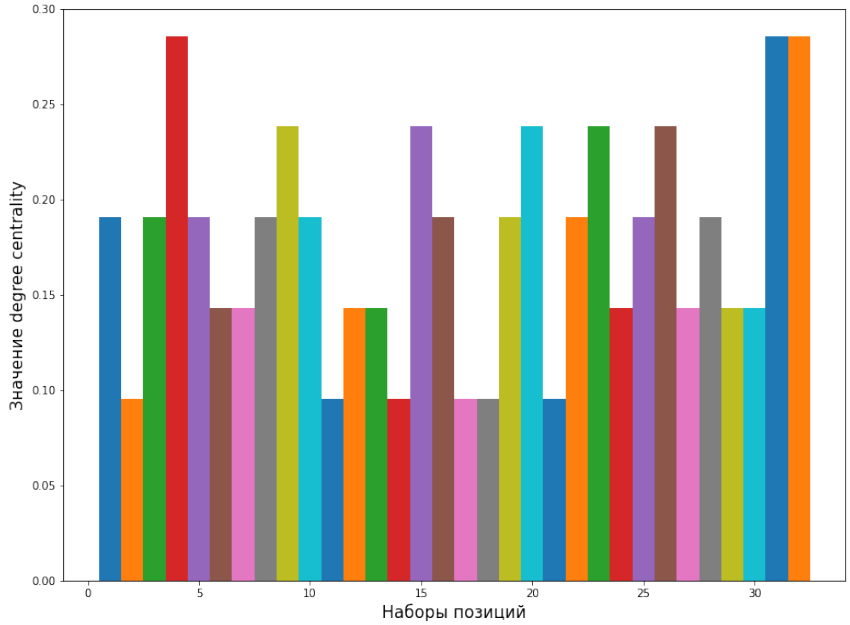

Рис. 25: Пример 2. Значение degree centrality для дронов, расположенных в потенциальных позициях.

Пример 3.

В последнем примере решение с использованием меры центральности по степени, как и во втором примере, может дать значительно отличающийся от оптимального варианта.

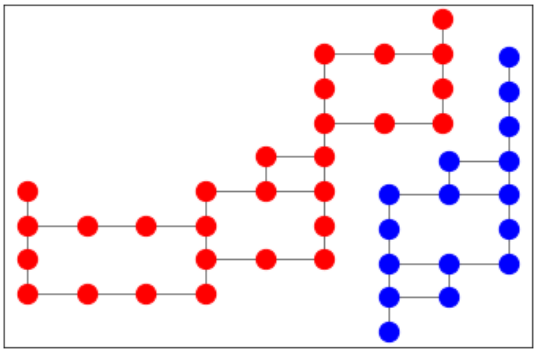

Рис. 26: Пример 3. Исходный граф.

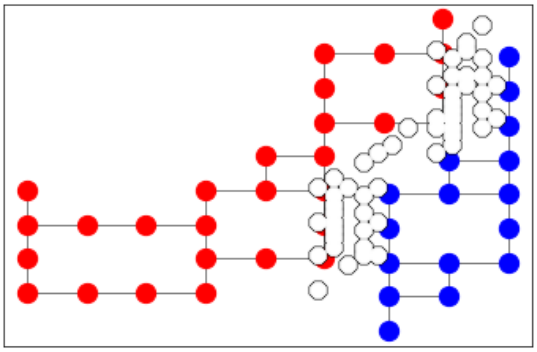

Рис. 27: Пример 3. Граф с потенциальными позициями.

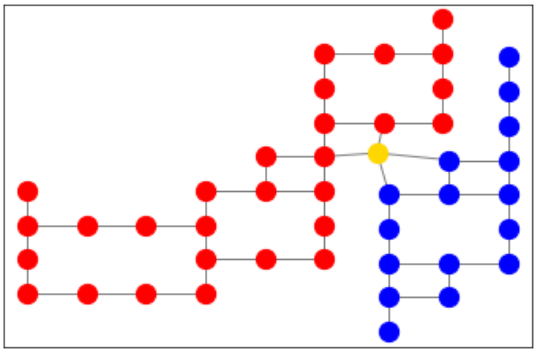

Рис. 28: Пример 3. Решение, найденное алгоритмом.

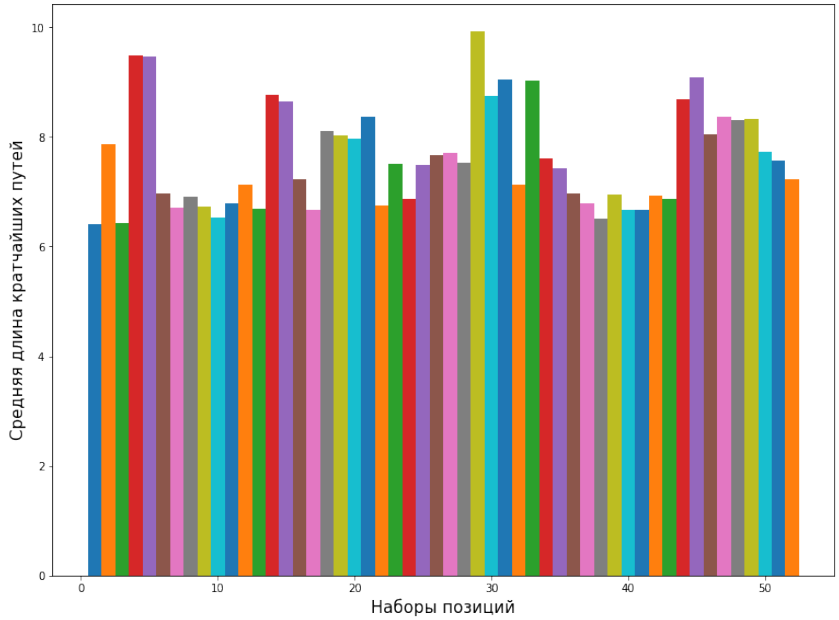

Рис. 29: Пример 3. Средняя длина кратчайших путей графа сети с дроном, расположенном в различных позициях.

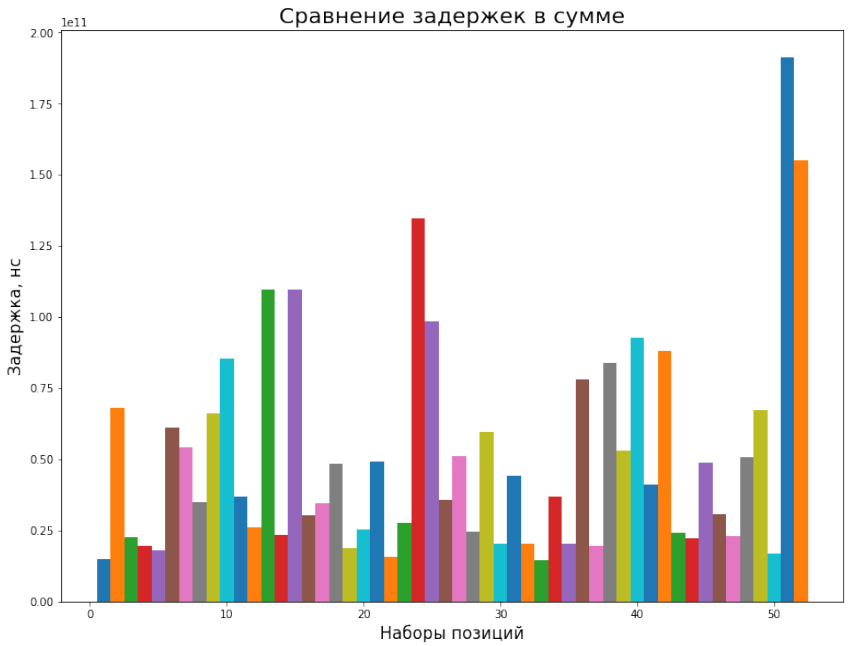

Рис. 30: Пример 3. Сумма задержек для потенциальных позиций.

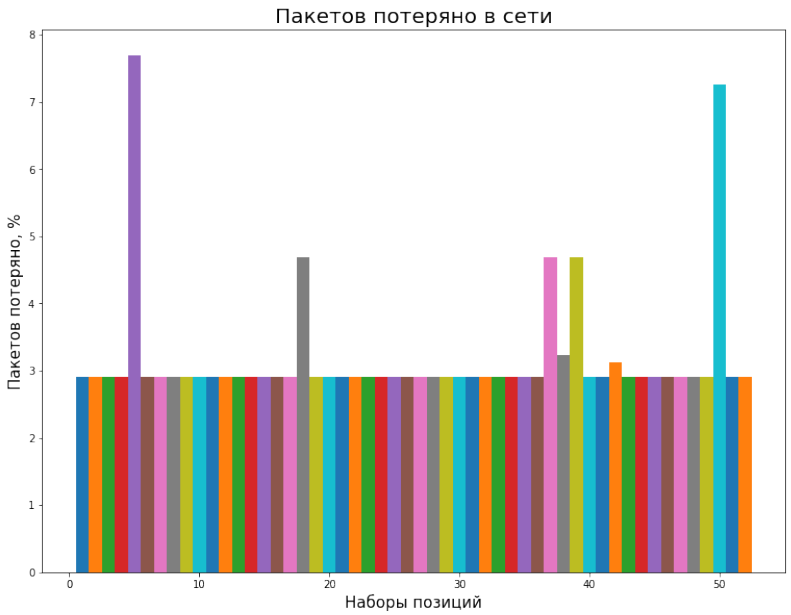

Рис. 31: Пример 3. Процент потерянных пакетов от отправленных для потенциальных позиций.

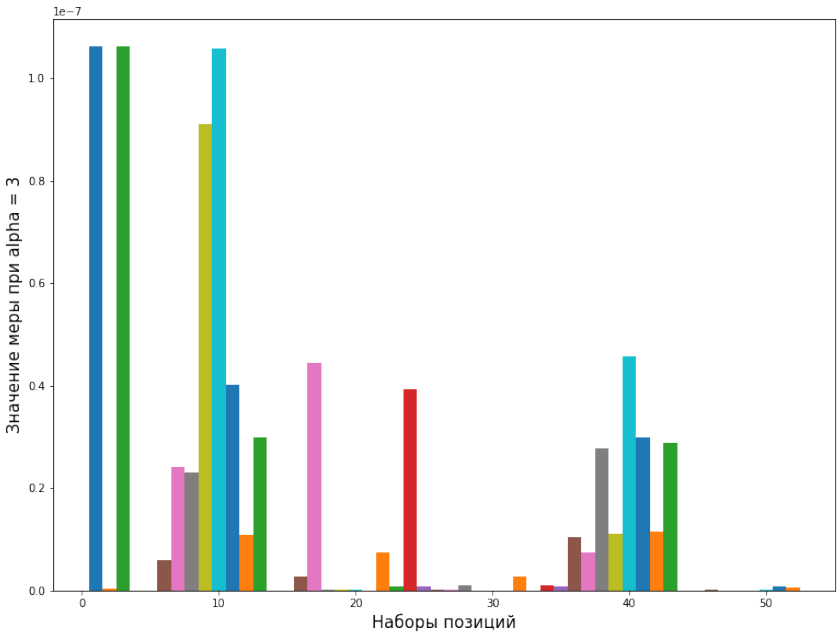

Рис. 32: Пример 3. Значение измененной меры центральности через вектор Майерсона в различных позициях дрона.

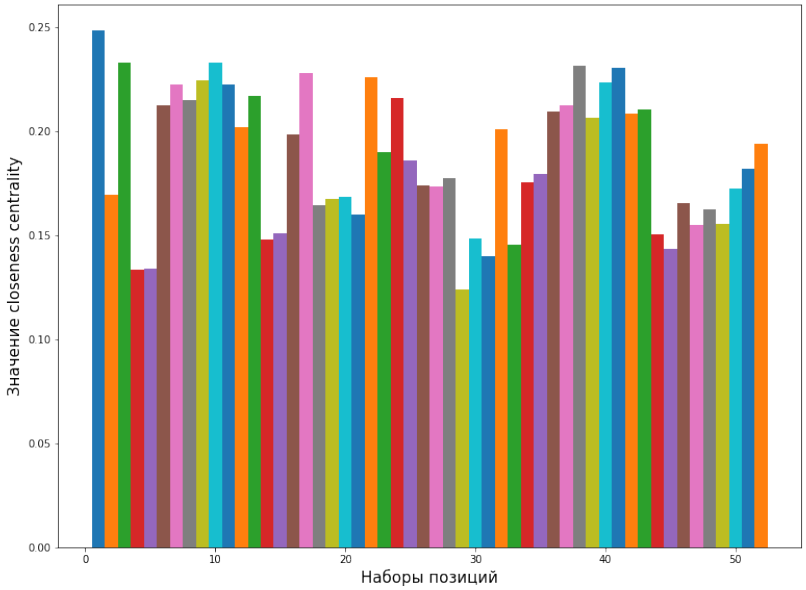

Рис. 33: Пример 3. Значение closeness centrality для дронов, расположенных в потенциальных позициях.

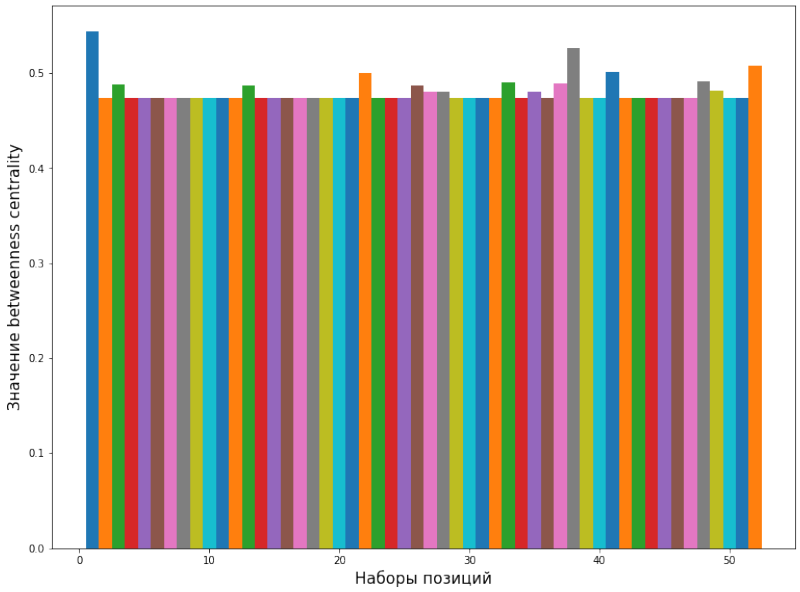

Рис. 34: Пример 3. Значение betweenness centrality для дронов, расположенных в потенциальных позициях.

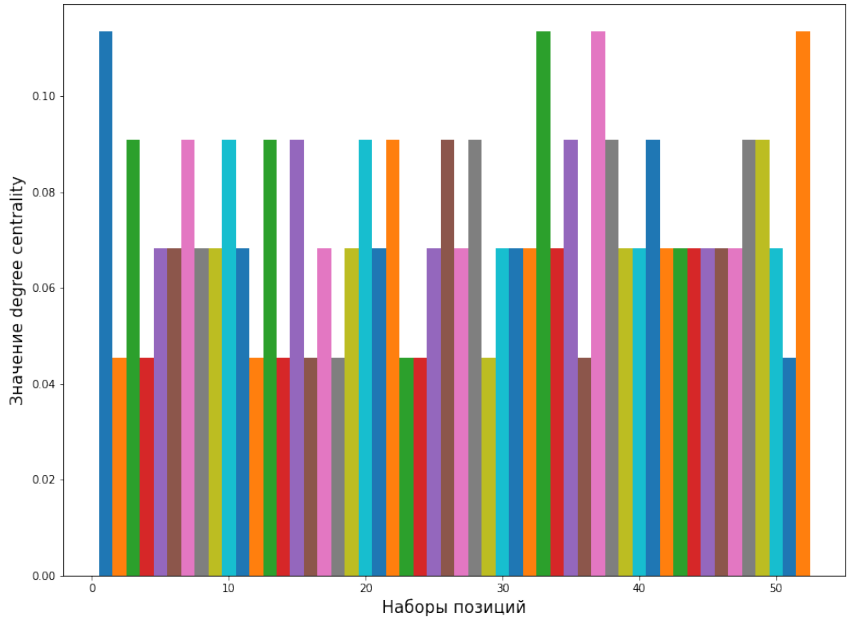

Рис. 35: Пример 3. Значение degree centrality для дронов, расположенных в потенциальных позициях.

В итоге можно сделать вывод, что реализованный алгоритм при использовании рассматриваемых мер центральности, помимо центральности по степени, находит оптимальные или очень близкие к ним решения. Стоит отметить, что при использовании меры центральности через вектор Майерсона можно получить отличающиеся друг от друга решения при разном значении параметра меры, поэтому перед его использованием нужно подобрать оптимальное значение параметра.

Выводы

1. Получена математическая модель задачи. Данная модель описывает MANET через теорию графов в форме кооперативной игры, в которой сети игроков объединяются в единую сеть с помощью дронов оптимальным образом для получения максимального выигрыша.

2. Составлен алгоритм, который находит позиции дронов в сети, представленной в виде графа на двумерной плоскости. Промежуточным результатом алгоритма является список уникальных позиций дронов, имеющих всевозможные разные связи с другими узлами сети. Конечным результатом алгоритма является список позиций, дающих наибольший выигрыш игрокам.

3. Написана программная реализация алгоритма. Программа позволяет быстро построить сеть с игроками, найти потенциальные позиции и конечный результат, выраженный списком позиций. Код достаточно гибкий, чтобы вносить изменения и дополнять программу новыми возможностями. Также программа позволяет увидеть сеть на различных этапах, например: исходную сеть, сеть с потенциальными позициями, сеть с выбранными позициями. Ознакомиться с программным кодом можно в репозитории GitHub: https://github.com/vladbusel/network_optimization.

4. Написан скрипт, моделирующий MANET и передающий данные между узлами этой сети. Результаты моделирования показали, что при использовании мер центральности через вектор Майерсона и по близости выбранные алгоритмом позиции дают более хорошие показатели сети в сравнении с большинством других рассмотренных позиций, в рассмотренных примерах они в основном являются лучшими, а если не являются, то очень близки к лучшим по результатам моделирования.

5. Решения, полученные алгоритмом, дали наименьшую среднюю длину кратчайших путей графа сети на рассмотренных примерах при использовании мер центральности через вектор Майерсона, по близости и по

посреднечеству. Из этого следует, что передача данных между узлами при позиции дрона, выбранной алгоритмом, в среднем должна быть наименьшей.

6. Рассмотрено несколько вариантов мер центральности для вычисления функции выигрыша. Среди них оптимальные или достаточно близкие к ним результаты дали измененная мера центральности через вектор Майерсона, мера центральности по близости и по посреднечеству. Решения, полученные с использованием меры центральности по степени, невсегда дают единственный вариант. Также некоторые полученные варианты решения при использовани центральности по степени дали большую задержку сети при моделировании и большую среднюю длину кратчайших путей в сравнении с остальными рассмотренными мерами.

Заключение

В данной работе получена формулировка задачи оптимизации передачи информации в самоорганизующихся мобильных сетях, используя графы в качестве модели сети. Сформулирован алгоритм, способный решить данную задачу. Данный алгоритм использует некоторые геометрические особенности задачи для ее решения. Для подсчета значения функции выигрыша используются меры центральности. Рассмотрена мера центральности через вектор Майерсона, а также некоторые другие, более известные меры центральности: по степени, близости и посредничеству.

Написана и протестирована программная реализация алгоритма на языке программирования python. Также проведено моделирование различных ситуаций в ns-3, полученные результаты проанализированы. Из них следует, что алгоритм при использовании меры центральности через вектор Майерсона или центральности по близости для подсчета значения функции выигрыша дают близкие к оптимальным или оптимальные решения по результатам моделирования и рассмотренной характеристике графа сети. Другие рассмотренные меры центральности не подошли для данной задачи.

Некоторые результаты, полученные в процессе работы над ВКР, опубликованы в [19].

В итоге все поставленные задачи были выполнены. В дальнейшем планируется реализовать другие способы объединения сети, требующие несколько дронов для образования связи между двумя игроками, рассмотреть случаи, когда кластеры уже имеют связь между друг другом. На данный момент подразумевается, что кластеры известны заранее. В общем решении задачи может потребоваться определить кластеры с помощью алгоритма, поэтому рассматривается возможность выяснить подходящие способы определения кластеров для данной задачи. Определив кластеры и образовав новые связи между ними с помощью дронов можно улучшить качество связи между этими кластерами. Также планируется рассмотреть критерии устойчивости [20] вновь созданной при помощи дрона сети. Связь между узлами сети является устойчивой, если между ними существуют два непересекающихся пути или они связаны напрямую.

Список литературы

[1] Bang, A.O.; Ramteke, P.L. MANET: History, challenges and applications. Int. J. Appl. Innov. Eng. Manag. (IJAIEM) 2013, 2, P. 249–251.

[2] Основной веб-сайт ns-3 [Электронный ресурс]: URL: https://www.nsnam.org/ (дата обращения: 26.05.21).

[3] Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр. 2-е изд. СПб.: БХВ-Петербург, 2014. 432 с.

[4] Novikov, D. A. Games and networks // Automation and Remote Control 2014, 75, 1145–1154 (2014). https://doi.org/10.1134/S0005117914060149.

[5] Тимонин Н. О. Одна динамическая игра управления агентами в сети. URL: https://dspace.spbu.ru/bitstream/11701/4505/1/Diplom.pdf (дата обращения: 26.05.21).

[6] Blakeway S., Gromov D. V., Gromova E. V., Kirpichnikova A. S., Plekhanova T. M. Increasing the performance of a Mobile Ad-hoc Network using a game-theoretic appoach to drone positioning // Вестник Санкт-Петербургского университета. Прикладная математика. Информатика. Процессы управления. 2019. Т. 15. Вып. 1. С. 22–38. https://doi.org/10.21638/11702/spbu10.2019.102.

[7] Gromova E. V., Gromov D. V., Timonin N., Kirpichnikova A. S., Blakeway S. A dynamic game of mobile agents placement on MANET // Proc. of the IEEE conference SIMS 2016. doi:10.1109/SIMS.2016.25.

[8] Воронцов А. О поиске местоположения дронов для оптимизации работы сети типа MANET // Процессы управления и устойчивость. 2019. Т. 6. № 1. С. 404–408.

[9] Киреев С. А. Теоретико-игровая модель передачи данных в беспроводных сетях с различной архитектурой. URL: https://dspace.spbu.ru/bitstream/11701/26456/1/diploma_kireev_serg_3.pdf (дата обращения: 26.05.21).

[10] Киреев С. А. Оптимизация передачи информации в самоорганизующихся сетях // Процессы управления и устойчивость. 2020. Т. 7. № 1. С. 381–386.

[11] Han Z., Niyato D., Saad D., Basar T., Hjorungnes A. Game Theory in Wireless and Communication Networks. Theory, Models and Applications. New York: Cambridge University Press, 2012. 554 p.

[12] Мазалов В. В., Чиркова Ю. В. Сетевые игры. 1-е изд. СПб.: Лань, 2018. 320 с.

[13] Newman M. Networks: An Introduction. 1st Edition. Oxford: Oxford University Press, 2010. 772 p.

[14] Винокуров В. М., Пуговкин А. В., Пшенников А. А., Ушарова Д. Н., Филатов А. С. Маршрутизация в беспроводных мобильных Ad hoc-сетях // Доклады Томского государственного университета систем управления и радиоэлектроники. 2010. № 2-1 (22). С. 288–292.

[15] Голубева М. А. О совместном использовании узлов децентрализованной беспроводной самоорганизующейся сети. URL: `https://dspace.spbu.ru/bitstream/11701/26517/1/Diplom_Golubeva.pdf` (дата обращения: 26.05.21).

[16] Плеханова Т. М. Использование теоретико-игрового подхода для повышения производительности сети MANET. URL: `https://dspace.spbu.ru/bitstream/11701/11944/1/diplom_plekhanova.pdf` (дата обращения: 26.05.21).

[17] Gromova E. V., Kireev S. A., Lazareva A. V, Kirpichnikova A. S., Gromov D. V. MANET Performance Optimization Using Network-Based Criteria and Unmanned Aerial Vehicles // Journal of Sensor and Actuator Networks. doi:10.3390/jsan10010008.

[18] ns-3 Documentation [Электронный ресурс]: URL: `https://www.nsnam.org/doxygen/index.html` (дата обращения: 26.05.21).

[19] Бусел В. Д., Лазарева А. В. Использование различных мер центральности в задаче оптимизации передачи информации в самоорганизующихся сетях // Процессы управления и устойчивость. 2021.

[20] Laclau M. Robust communication on networks. URL: https://arxiv.org/pdf/2007.00457.pdf (дата обращения: 26.05.21).

Приложения

manet_simulation.cc

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/default-deleter.h"
#include "ns3/point-to-point-module.h"

#include <iostream>
#include <fstream>
#include <sstream>
#include "string"
#include "vector"

using namespace ns3;
using namespace std;

NS_LOG_COMPONENT_DEFINE ("manet-routing");

class RoutingExperiment
{
public:
  RoutingExperiment ();
  void Run ();
  std::string CommandSetup (int argc, char **argv);

private:
  Ptr<Socket> SetupPacketReceive (Ipv4Address addr, Ptr<Node> node);
  void ReceivePacket (Ptr<Socket> socket);
  void CheckThroughput ();

  uint32_t port;
  uint32_t bytesTotal;
  uint32_t packetsReceived;

  std::string m_CSVfileName;
  std::string m_protocolName;
  bool m_traceMobility;
  uint32_t m_protocol;
};

RoutingExperiment::RoutingExperiment ()
  : port (9),
    bytesTotal (0),
    packetsReceived (0),
```

```
53      m_CSVfileName ("/home/vlad/jupyter_notebook_project_dir/p2p_manet.output.csv"),
54      m_traceMobility (false),
55      m_protocol (3) // DSDV
56  {
57  }
58
59  static inline std::string
60  PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet, Address senderAddress)
61  {
62    std::ostringstream oss;
63
64    oss << Simulator::Now ().GetSeconds () << " " << socket->GetNode ()->GetId ();
65
66    if (InetSocketAddress::IsMatchingType (senderAddress))
67      {
68        InetSocketAddress addr = InetSocketAddress::ConvertFrom (senderAddress);
69        oss << " received one packet from " << addr.GetIpv4 ();
70      }
71    else
72      {
73        oss << " received one packet!";
74      }
75    return oss.str ();
76  }
77
78  void
79  RoutingExperiment::ReceivePacket (Ptr<Socket> socket)
80  {
81    Ptr<Packet> packet;
82    Address senderAddress;
83    while ((packet = socket->RecvFrom (senderAddress)))
84      {
85        bytesTotal += packet->GetSize ();
86        packetsReceived += 1;
87        NS_LOG_UNCOND (PrintReceivedPacket (socket, packet, senderAddress));
88      }
89  }
90
91  void
92  RoutingExperiment::CheckThroughput ()
93  {
94    double kbs = (bytesTotal * 8.0) / 1024;
95    bytesTotal = 0;
96
97    std::ofstream out (m_CSVfileName.c_str (), std::ios::out);
98
99    out << (Simulator::Now ()).GetSeconds () << ","
100       << kbs << ","
101       << packetsReceived << ","
102       << m_protocolName << ","
103       << std::endl;
104
105
106   out.close ();
107   packetsReceived = 0;
108   Simulator::Schedule (Seconds (1.0), &RoutingExperiment::CheckThroughput, this);
109 }
110
```

```
111 Ptr<Socket>
112 RoutingExperiment::SetupPacketReceive (Ipv4Address addr, Ptr<Node> node)
113 {
114   TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
115   Ptr<Socket> sink = Socket::CreateSocket (node, tid);
116   InetSocketAddress local = InetSocketAddress (addr, port);
117   sink->Bind (local);
118   sink->SetRecvCallback (MakeCallback (&RoutingExperiment::ReceivePacket, this));
119 
120   return sink;
121 }
122 
123 std::string
124 RoutingExperiment::CommandSetup (int argc, char **argv)
125 {
126   CommandLine cmd;
127   cmd.AddValue ("CSVfileName", "The name of the CSV output file name", m_CSVfileName);
128   cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);
129   cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
130   cmd.Parse (argc, argv);
131   return m_CSVfileName;
132 }
133 
134 int main (int argc, char *argv[])
135 {
136   RoutingExperiment experiment;
137   std::string CSVfileName = experiment.CommandSetup (argc,argv);
138 
139   std::ofstream out (CSVfileName.c_str ());
140   out << "SimulationSecond," <<
141   "ReceiveRate," <<
142   "PacketsReceived," <<
143   "NumberOfSinks," <<
144   "RoutingProtocol," <<
145   "TransmissionPower" <<
146   std::endl;
147   out.close ();
148 
149   experiment.Run ();
150 }
151 
152 void
153 RoutingExperiment::Run ()
154 {
155   Packet::EnablePrinting ();
156 
157   std::string rate ("2048bps");
158   std::string phyMode ("DsssRate11Mbps");
159 
160   std::string txt_name ("/home/vlad/jupyter_notebook_project_dir/my_network_with_nodes32.txt");
161   std::string tr_name ("/home/vlad/jupyter_notebook_project_dir/p2p_manet32");
162   m_protocolName = "DSDV";
163 
164   Config::SetDefault  ("ns3::OnOffApplication::PacketSize",StringValue ("64"));
165   Config::SetDefault ("ns3::OnOffApplication::DataRate",  StringValue (rate));
166   Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue (phyMode));
167 
168   uint16_t port = 9;
```

```
169     double txp = 4.6875;
170
171     WifiHelper wifi;
172
173     YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
174     YansWifiChannelHelper wifiChannel;
175     wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
176     wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
177     wifiPhy.SetChannel (wifiChannel.Create ());
178
179     YansWifiPhyHelper wifiPhy2 =  YansWifiPhyHelper::Default ();
180     YansWifiChannelHelper wifiChannel2;
181     wifiChannel2.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
182     wifiChannel2.AddPropagationLoss ("ns3::FriisPropagationLossModel");
183     wifiPhy2.SetChannel (wifiChannel2.Create ());
184
185     // Add a mac and disable rate control
186
187     wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
188                                   "DataMode",StringValue (phyMode),
189                                   "ControlMode",StringValue (phyMode));
190
191
192     wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
193     wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));
194
195     wifiPhy2.Set ("TxPowerStart",DoubleValue (txp));
196     wifiPhy2.Set ("TxPowerEnd", DoubleValue (txp));
197
198     WifiMacHelper wifiMac;
199     wifiMac.SetType ("ns3::AdhocWifiMac");
200     WifiMacHelper wifiMac2;
201     wifiMac2.SetType ("ns3::AdhocWifiMac");
202
203
204           ifstream in;
205           in.open(txt_name);
206
207     if (in.is_open())
208     {
209       cout << "File open" << endl;
210     }
211
212     else
213     {
214       cout << "File open is failed" << endl;
215     }
216     int nodesNumber;
217     int dronsNumber;
218     in >> nodesNumber >> dronsNumber;
219     NodeContainer adhocNodes;
220     adhocNodes.Create(nodesNumber);
221     NodeContainer dronContainer;
222     dronContainer.Create(dronsNumber);
223
224     // Кординаты узлов и их принадлежность игроку
225     double gridScalingFactor = 100.0;
226     Ptr<ListPositionAllocator> positionAlloc = CreateObject <ListPositionAllocator>();
```

```
227     double x, y;
228     int player_id;
229     int playersNodes[nodesNumber];
230     for (int node = 0; node < nodesNumber; node++)
231     {
232       in >> x >> y >> player_id;
233       positionAlloc ->Add(Vector(x*gridScalingFactor, y*gridScalingFactor, 0));
234       playersNodes[node] = player_id;
235     }
236
237     for (int dron = 0; dron < dronsNumber; dron++)
238     {
239       in >> x >> y;
240       positionAlloc ->Add(Vector(x*gridScalingFactor, y*gridScalingFactor, 0));
241     }
242
243     MobilityHelper mobilityAdhoc;
244     mobilityAdhoc.SetPositionAllocator(positionAlloc);
245     mobilityAdhoc.SetMobilityModel("ns3::ConstantPositionMobilityModel");
246     mobilityAdhoc.Install (adhocNodes);
247     mobilityAdhoc.Install (dronContainer);
248
249
250     NodeContainer n_container;
251     NodeContainer n_container2;
252     int container_size = 0;
253     int container2_size = 0;
254     for (int i = 0; i < nodesNumber; i++)
255     {
256         if (playersNodes[i] == 1)
257         {
258           n_container.Add(adhocNodes.Get (i));
259           container_size++;
260         }
261         if (playersNodes[i] == 2)
262         {
263           n_container2.Add(adhocNodes.Get (i));
264           container2_size++;
265         }
266     }
267
268     for (int i = 0; i < dronsNumber; i++)
269     {
270       n_container.Add(dronContainer.Get (i));
271       n_container2.Add(dronContainer.Get (i));
272     }
273
274     Ipv4ListRoutingHelper list;
275     InternetStackHelper internet;
276     DsdvHelper dsdv;
277     list.Add (dsdv, 100);
278     internet.SetRoutingHelper (list);
279     internet.Install(adhocNodes);
280     internet.Install(dronContainer);
281
282     Ipv4AddressHelper address;
283     Ipv4InterfaceContainer adhocInterfaces;
284     address.SetBase ("10.1.1.0", "255.255.255.0");
```

```
285
286     wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
287     NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac, n_container);
288     adhocInterfaces.Add(address.Assign(adhocDevices));
289
290     NetDeviceContainer adhocDevices2 = wifi.Install (wifiPhy2, wifiMac2, n_container2);
291     adhocInterfaces.Add(address.Assign(adhocDevices2));
292
293
294     in.close();
295
296     cout << "PACKETS INITIALIZE\n";
297
298     Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
299
300     OnOffHelper onoff ("ns3::UdpSocketFactory",Address ());
301     onoff.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));
302     onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));
303     double startTime = 25.0;
304     double shift = 0.7;
305
306
307     for (int i = 0; i < container_size; i++)
308     {
309       for (int j = 0; j < container2_size; j++)
310       {
311         Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (container_size+container2_size-1), dronContainer
312
313         AddressValue remoteAddress (InetSocketAddress (adhocInterfaces.GetAddress (container_size+container2_size-1), port)
314         onoff.SetAttribute ("Remote", remoteAddress);
315
316         ApplicationContainer temp = onoff.Install (n_container2.Get (j));
317
318         Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();
319         temp.Start (Seconds (var->GetValue (startTime,startTime+0.03)));
320         temp.Stop (Seconds (startTime+shift));
321       }
322       startTime += shift;
323     }
324
325     for (int i = 0; i < container_size; i++)
326     {
327       for (int j = 0; j < container2_size; j++)
328       {
329         Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (i), n_container.Get (i));
330
331         AddressValue remoteAddress (InetSocketAddress (adhocInterfaces.GetAddress (i), port));
332         onoff.SetAttribute ("Remote", remoteAddress);
333
334         ApplicationContainer temp = onoff.Install (dronContainer.Get (0));
335         Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();
336         temp.Start (Seconds (var->GetValue (startTime,startTime+0.03)));
337         temp.Stop (Seconds (startTime+shift));
338       }
339       startTime += shift;
340     }
341
342     cout << "FLOWMONITOR INITIALIZE\n";
```

```
343     Ptr<FlowMonitor> flowmon;
344     FlowMonitorHelper flowmonHelper;
345     flowmon = flowmonHelper.InstallAll ();
346 
347     NS_LOG_INFO ("Run Simulation.");
348 
349     CheckThroughput ();
350 
351     Simulator::Stop (Seconds (startTime));
352     Simulator::Run ();
353 
354     flowmon->SerializeToXmlFile ((tr_name + ".flowmon").c_str(), true, true);
355 
356     Simulator::Destroy ();
357 
358   }
```

network_game.ipynb

30 мая 2021 г.

[ ]:
```
from itertools import combinations,combinations_with_replacement
import networkx as nx
import matplotlib.pyplot as plt
import subprocess
from xml.etree import cElementTree as ET
import sys
import numpy as np

color_map = {0:'r', 1:'b', 2:'g', 3:'black', 4:'orangered', 5:'m', 6:'y'}

def poly_value(coef_vec,x):
    value = 0
    reverse_vec = coef_vec
    reverse_vec.reverse()
    for coef in reverse_vec:
        value = coef + value*x
    return value

class Pos:

    def __init__(self,i = 0,j = 0):
        self.i = i
        self.j = j

    def __str__(self):
        return f'({self.i},{self.j})'

    def __add__(self, other):
        return Pos(self.i+other.i,self.j+other.j)

    def __sub__(self, other):
        return Pos(self.i-other.i,self.j-other.j)

    def __mul__(self, other):
        try:
            return Pos(self.i*other,self.j*other)
        except:
            print(f'{type(self)} and {type(other)} are not multiply')

    def __rmul__(self, other):
        try:
                return Pos(self.i*other,self.j*other)
        except:
            print(f'{type(self)} and {type(other)} are not multiply')

    def __truediv__(self, other):
        try:
            return Pos(self.i/other,self.j/other)
        except:
            print(f'{type(self)} and {type(other)} are not divisible')

    def __abs__(self):
        return (self.i**2 + self.j**2)**0.5


    def __eq__(self, other):
        if type(self) == Pos and type(other) == Pos:
```

```
            return (self.i == other.i) and (self.j == other.j)

        return False


    def __hash__(self):
        return hash((self.i,self.j))


    def around(self,dist=1,grid_size=1):

        neighbors = [self]
        if dist >= grid_size:
            for x in np.arange(grid_size,dist,grid_size):
                neighbors += [Pos(self.i-x,self.j),Pos(self.i+x,self.j),
                              Pos(self.i,self.j-x),Pos(self.i,self.j+x)]
            if (dist-1) % grid_size == 0:
                neighbors += [Pos(self.i-dist,self.j),
                              Pos(self.i+dist,self.j),
                              Pos(self.i,self.j-dist),
                              Pos(self.i,self.j+dist)]
            for x in np.arange(grid_size,dist+(1e-1),grid_size):
                for y in np.arange(grid_size,
                                   (dist**2 - x**2)**0.5+(1e-1),grid_size):
                    if x**2 + y**2 <= dist**2:
                        neighbors += [Pos(self.i-x,self.j-y),
                                      Pos(self.i-x,self.j+y),
                                      Pos(self.i+x,self.j-y),
                                      Pos(self.i+x,self.j+y),
                                      Pos(self.i-y,self.j-x),
                                      Pos(self.i-y,self.j+x),
                                      Pos(self.i+y,self.j-x),
                                      Pos(self.i+y,self.j+x),]

        return neighbors


    def get_distance(self,other,mode=0):
        if mode:
            return abs(self.i - other.i) + abs(self.j - other.j)

        return ((self.i - other.i)**2 + (self.j - other.j)**2)**0.5



class Agent:

    def __init__(self,player_id = None,pos = None):
        self.player_id = player_id
        self.pos = pos

    def __str__(self):
        return f'<player_id = {self.player_id}, pos = {self.pos}>'

    def is_neighbour(self,other,dist):
        return self.pos.get_distance(other.pos) <= dist

    def neighbors_pos(self,dist=1,grid_size=1):
        return self.pos.around(dist,grid_size)



class Network:

    def __init__(self):
        self.agents = []
        self.drons = []
        self.graph = nx.Graph()
        self.players_id = []
        self.r = 1
        self.grid_size = 1

    def __str__(self):
```

```
        return f'<agents = {list(map(str,self.agents))}>'

    def add_agent(self, new_agent):
        neighbours = []
        for obj in (self.agents):
            if obj.player_id == new_agent.player_id :
                if obj.pos.get_distance(new_agent.pos) <= self.r:
                    neighbours.append(obj)

        for obj in (self.drons):
            if obj.pos.get_distance(new_agent.pos) <= self.r:
                neighbours.append(obj)

        self.graph.add_node(new_agent)

        if new_agent.player_id not in self.players_id:
            self.players_id.append(new_agent.player_id)
        self.graph.add_node(new_agent)

        for neighbour in neighbours:
            if new_agent.player_id == neighbour.player_id:
                self.graph.add_edge(new_agent,neighbour)

        self.agents.append(new_agent)


    def add_agents(self, new_agent_list):
        for new_agent in new_agent_list:
            self.add_agent(new_agent)

    def add_dron(self, new_dron):
        neighbours = []
        for obj in (self.drons+self.agents):
            if obj.pos.get_distance(new_dron.pos) <= self.r:
                neighbours.append(obj)
        self.graph.add_node(new_dron)

        if len(neighbours) > 0:
            for neighbour in neighbours:
                self.graph.add_edge(new_dron,neighbour)


        self.drons.append(new_dron)

    def add_drons(self, new_dron_list):
        for new_dron in new_dron_list:
            self.add_dron(new_dron)

    def del_dron(self, dron):
        if self.drons.count(dron) != 0:
            self.drons.remove(dron)
            self.graph.remove_node(dron)

    def del_drons(self, drons_list):
        for dron in drons_list:
            self.del_dron(dron)

    def create_graph(self):
        self.graph = nx.Graph()
        for agent in self.agents:
            self.graph.add_node(agent)
        for agent in self.agents:
            for other_agent in self.agents:
                if agent.is_neighbour(other_agent) and agent.player_id == other_agent.player_id:
                    self.graph.add_edge(agent,other_agent)

        for dron in self.drons:
            self.graph.add_node(dron)
        for dron in self.drons:
            for other in (self.drons+self.agents):
                if dron.is_neighbour(other):
                    self.graph.add_edge(dron,other)
        return self.graph
```

```
def draw_graph(self, visible_players_list = None,
               possible_positions = None,pngname=None):
    G = self.graph.copy()
    G.remove_nodes_from(self.drons)
    subgraphs = [G.subgraph(c) for c in nx.connected_components(G)]
    subgraphs = {list(subgraph)[0].player_id: subgraph
                 for subgraph in subgraphs}
    if visible_players_list == None:
        visible_players_list = list(subgraphs.keys())
    pos = {agent: (agent.pos.i+agent.player_id*0.1,
                   agent.pos.j-agent.player_id*0.1) for agent in self.agents}
    pos.update({dron: (dron.pos.i,dron.pos.j) for dron in self.drons})
    for i in visible_players_list:
        nx.draw_networkx_nodes(self.graph, pos, node_size=150,
                               nodelist=subgraphs[i], node_color=color_map[i-1])
    nx.draw_networkx_nodes(self.graph, pos, node_size=150,
                           nodelist=self.drons, node_color="gold")
    nx.draw_networkx_edges(self.graph, pos, edgelist = self.graph.edges,
                           alpha=0.5, width=1)

    if possible_positions != None:
        possible_drons = [Agent(pos=possible_position)
                          for possible_position in possible_positions]
        self.graph.add_nodes_from(possible_drons)
        pos = {possible_dron: (possible_dron.pos.i,possible_dron.pos.j)
               for possible_dron in possible_drons}
        nx.draw_networkx_nodes(self.graph, pos, node_size=140,
                               nodelist=possible_drons, node_color="black", node_shape = "8")
        nx.draw_networkx_nodes(self.graph, pos, node_size=110,
                               nodelist=possible_drons, node_color="w", node_shape = "8")
        self.graph.remove_nodes_from(possible_drons)
    plt.axis("on")
    if pngname != None:
        plt.savefig(pngname)
    plt.show()

def get_player_subnet(self, p_id, with_drons=False):
    subgraph = self.graph.copy()
    remove_list = []
    for node in subgraph.nodes:
        if node.player_id != p_id:
            if not with_drons:
                remove_list.append(node)
            elif node.player_id != 0:
                remove_list.append(node)
    subgraph.remove_nodes_from(remove_list)
    if with_drons:
        nodes = []
        for node in list(subgraph.nodes):
            if node.player_id != 0:
                nodes.append(node)
    else:
        nodes = list(subgraph.nodes)
    subnet = Network()
    subnet.add_agents(nodes)
    if with_drons:
        subnet.add_drons(self.drons)
    return subnet

def get_possible_grid_positions(self,grid_size=1):
    positions = set()
    agents_pos = {}
    for agent in network.agents:
        positions.update(agent.neighbors_pos(network.r,network.grid_size))

    possible_positions = {}
    for pos in positions:
        player_set = set()
        agents_list = []
        for agent in network.agents:
            if pos.get_distance(agent.pos) <= network.r:
                player_set.add(agent.player_id)
```

```
                agents_list.append(agent)
        if len(player_set) > 1:
            if possible_positions.get(tuple(agents_list)) == None:
                possible_positions[tuple(agents_list)] = pos

    return list(possible_positions.values())



def get_possible_positions(self):
    a = tuple(set(self.agents))
    intersections = [[],[]]

    for p1 in range(len(a)):
        for p2 in range(p1+1,len(a)):
            if a[p1].player_id != a[p2].player_id:
                dist = a[p1].pos.get_distance(a[p2].pos)
                if dist == 0:
                    intersections[0] += [[]]
                    intersections[1] += [a[p1].pos]

                elif dist < 2*self.r:
                    h = (self.r**2-(dist/2)**2)**0.5
                    p12 = (a[p2].pos-a[p1].pos)
                    center = a[p1].pos + p12/2
                    pos_a = Pos(center.i+(h/dist)*p12.j,
                                center.j-(h/dist)*p12.i)
                    pos_b = Pos(center.i-(h/dist)*p12.j,
                                center.j+(h/dist)*p12.i)
                    intersections[0] += [[],[]]
                    intersections[1] += [pos_a,pos_b]

                elif dist == 2*self.r:
                    intersections[0] += [[]]
                    intersections[1] += [a[p1].pos+(a[p2].pos-a[p1].pos)/2]

    for i in range(len(intersections[1])):
        for agent in a:
            if intersections[1][i].get_distance(agent.pos) <= self.r:
                intersections[0][i].append(agent)

    intersection_dict = {}

    for agent_tuple,point_pos in zip(intersections[0],intersections[1]):
        agent_tuple = tuple(agent_tuple)

        if intersection_dict.get(agent_tuple) == None:
            max_len = 0
            for intersection in intersections[0]:
                if set(agent_tuple).issubset(set(intersection)):
                    if max_len < len(intersection):
                        max_len = len(intersection)

            if len(agent_tuple) == max_len:
                if intersection_dict.get(agent_tuple) == None:
                    intersection_dict[agent_tuple] = [point_pos]
                else:
                    intersection_dict[agent_tuple].append(point_pos)

    values = list(intersection_dict.values())
    return list(map(lambda x: np.mean(np.array(x)),values))



def get_pos_value(self,alpha,position):
    dron = Agent(pos = position)
    self.add_dron(dron)
    value = self.get_value(alpha)
    self.del_dron(dron)
    return value

def get_value(self,alpha):
    len_ways_vec = [0 for i in range(len(self.agents)+len(self.drons))]
```

```
        shotest_ways = []
        for s in self.agents:
            for t in self.agents:
                if s.player_id != t.player_id:
                    if nx.has_path(self.graph, source=s, target=t):
                        for path in list(nx.all_shortest_paths(self.graph,
                                                               source=s, target=t)):
                            shotest_ways.append(path)
        for path in shotest_ways:
            for dron in self.drons:
                if dron in list(path):
                    len_ways_vec[len(path)-1] += 1/len(path)
                    break

        return poly_value(len_ways_vec[1:],alpha)

    def get_value_cc(self):
        cc = nx.closeness_centrality(self.graph)
        val = 0
        for dron in self.drons:
            val += cc[dron]
        return val

    def get_value_bc(self):
        bc = nx.betweenness_centrality(self.graph)
        val = 0
        for dron in self.drons:
            val += bc[dron]
        return val


    def get_value_dc(self):
        dc = nx.degree_centrality(self.graph)
        val = 0
        for dron in self.drons:
            val += dc[dron]
        return val


    def get_best_positions(self,possible_positions,alpha,
                           drons_count=None,mode=0):
        if drons_count == None:
            drons_count = len(self.players_id)-1

        max_val = 0
        best_set = None

        if len(possible_positions) >= drons_count:
            for positions in combinations(possible_positions, drons_count):
                drons = [Agent(pos = position) for position in positions]
                network.add_drons(drons)
                if nx.is_connected(network.graph):
                    if mode == 0:
                        cur_value = network.get_value(alpha)
                    else:
                        cur_value = 1/network.get_value(alpha)
                    if cur_value > max_val:
                        max_val = cur_value
                        best_set = positions
                network.del_drons(drons)

        else:
            for positions in combinations_with_replacement(possible_positions,
                                                           drons_count):
                drons = [Agent(pos = position) for position in positions]
                network.add_drons(drons)
                if nx.is_connected(network.graph):
                    if mode == 0:
                        cur_value = network.get_value(alpha)
                    else:
                        cur_value = 1/network.get_value(alpha)
                    if cur_value > max_val:
                        max_val = cur_value
```

```
                    best_set = positions
            network.del_drons(drons)
        return best_set




    def write_info_file(self,drons_pos=None,
                        filename = 'my_network_with_nodes.txt'):
        if type(drons_pos) is list or type(drons_pos) is tuple:
            drons_list = [Agent(pos = dron_pos) for dron_pos in drons_pos]
            self.add_drons(drons_list)
            adj_matrix = nx.to_numpy_matrix(self.graph,dtype ='int')
        elif type(drons_pos) is Pos:
            dron = Agent(pos = drons_pos)
            self.add_dron(dron)
            adj_matrix = nx.to_numpy_matrix(self.graph,dtype ='int')
        else:
            adj_matrix = nx.to_numpy_matrix(self.graph,dtype ='int')
        with open(filename,'w') as file:
            file.write(str(len(self.agents))+' '+str(len(self.drons))+'\n')
            for agent in self.agents:
                file.write(str(agent.pos.i) + ' ' + str(agent.pos.j) + ' '
                           + str(agent.player_id) +'\n')
            for dron in self.drons:
                file.write(str(dron.pos.i) + ' ' + str(dron.pos.j) + '\n')
            str_matrix = ''
            for i in range(adj_matrix.shape[0]):
                for j in range(adj_matrix.shape[1]):
                    str_matrix += (str(adj_matrix[i,j])+' ')
                str_matrix += '\n'
            file.write(str_matrix)

        subnetA = self.get_player_subnet(1,True)
        if type(drons_pos) is list or type(drons_pos) is tuple:
            for dron in drons_list.copy():
                if len(list(subnetA.graph.neighbors(dron))) < 2:
                    subnetA.del_dron(dron)
        elif type(drons_pos) is Pos:
            if len(list(subnetA.graph.neighbors(dron))) < 2:
                subnetA.del_dron(dron)

        subnetB = self.get_player_subnet(2,True)
        if type(drons_pos) is list or type(drons_pos) is tuple:
            for dron in drons_list.copy():
                if len(list(subnetB.graph.neighbors(dron))) < 2:
                    subnetB.del_dron(dron)
        elif type(drons_pos) is Pos:
            if len(list(subnetB.graph.neighbors(dron))) < 2:
                subnetB.del_dron(dron)

        if type(drons_pos) is list or type(drons_pos) is tuple:
            self.del_drons(drons_list)
        elif type(drons_pos) is Pos:
            self.del_dron(dron)
```

[ ]:
```
def parse_flowmon(info_xml = 'p2p_manet.flowmon'):
    et=ET.parse(info_xml)
    txPackets = []
    rxPackets = []
    lostPackets = []
    delays = []

    for flow in et.iterfind('FlowStats/Flow'):
        txPackets.append(int(flow.get('txPackets')))
        rxPackets.append(int(flow.get('rxPackets')))
        lostPackets.append(int(flow.get('lostPackets')))
        delays.append(float(flow.get('delaySum')[:-2]))

    info_list = [txPackets,rxPackets,lostPackets,delays]
```

```
    return info_list

def plot_sum_bar(info_list,title='',xlabel='',ylabel='',
                 legend_labels=['1','2'],multiplier=2,_width=1,pngname=None):
    fig, ax = plt.subplots(figsize=[6.4*multiplier, 4.8*multiplier])
    plt.title(title, fontsize=20)
    plt.xlabel(xlabel, fontsize=15)
    plt.ylabel(ylabel, fontsize=15)

    if legend_labels != None:
        for i in range(len(info_list)):
            ax.plot(1+i*_width, (info_list[i]), label = legend_labels[i])
        ax.legend(fontsize = 'x-large')

    for i in range(len(info_list)):
        ax.bar(1+i*_width,(info_list[i]), width = _width)

    if pngname != None:
        plt.savefig(pngname)
    plt.show()


def plot_sum_bars(info_list,legend_labels=['Случай без дронов','Первый случай','Второй случай']):

    changed_info = [[],[],[],[]]

    for info in info_list:
        changed_info[0] += [info[3]]
        changed_info[1] += [info[2]]
        changed_info[2] += [info[0]]
        changed_info[3] += [info[1]]

    for info in changed_info:
        for i in range(len(info)):
            info[i] = sum(info[i])

    pngnames = ['delays_sum.png','lostPackets_sum.png',
                'txPackets_sum.png','rxPackets_sum.png']
    plot_sum_bar(changed_info[0],title='Сравнение задержек в сумме',
                 xlabel='Наборы позиций',ylabel='Задержка, нс',
                 legend_labels=legend_labels, pngname=pngnames[0])

    percent_received = []
    for i in range(len(changed_info[0])):
        percent_received.append(100*changed_info[1][i]/changed_info[2][i])
    plot_sum_bar(percent_received,title='Пакетов потеряно в сети',
                 xlabel='Наборы позиций',ylabel='Пакетов потеряно, %',
                 legend_labels=legend_labels,pngname=pngnames[3])


def plot_bar(list1,list2,title='',xlabel='',ylabel='',
             legend_labels=['Первый случай','Второй случай'],
             multiplier=1.5,_width=0.4,pngname=None):

    x1 = [i-_width*0.5 for i in range(len(list1))]
    x2 = [i+_width*0.5 for i in range(len(list2))]
    fig, ax = plt.subplots(figsize=[6.4*multiplier, 4.8*multiplier])

    plt.title(title, fontsize=20)
    plt.xlabel(xlabel, fontsize=15)
    plt.ylabel(ylabel, fontsize=15)

    ax.plot(x1, list1, '', label = legend_labels[0])
    ax.plot(x2, list2, '', label = legend_labels[1])

    ax.legend(fontsize = 'x-large')

    ax.bar(x1, list1, width = _width)
    ax.bar(x2, list2, width = _width)
    if pngname != None:
        plt.savefig(pngname)
    plt.show()
```

```
def plot_bars(info_list,info_list2):
    txPackets,rxPackets,lostPackets,delays = info_list
    txPackets2,rxPackets2,lostPackets2,delays2 = info_list2

    pngnames = ['delays.png','lostPackets.png','txPackets.png','rxPackets.png']
    plot_bar(delays,delays2,title='Сравнение задержек',
             xlabel='Flow id',ylabel='Delays in ns',pngname=pngnames[0])
    plot_bar(lostPackets,lostPackets2,title='Пакетов потеряно',
             xlabel='Flow id',ylabel='Lost packets',pngname=pngnames[1])
    plot_bar(txPackets,txPackets2,title='Пакетов отправлено',
             xlabel='Flow id',ylabel='Transmitted packets sum',pngname=pngnames[2])
    plot_bar(rxPackets,rxPackets2,title='Пакетов принято',
             xlabel='Flow id',ylabel='Received packets sum',pngname=pngnames[3])
```

[ ]:
```
# player1Pos = [[3,3],[4,2],[4,3],[4,4],[4,5],[5,5],
#               [6,5],[7,5],[7,4],[7,3],[7,2],[8,2],[7,1]]

# player2Pos = [[3,5],[4,5],[5,5],[6,5],[4,4],
#               [4,3],[4,2],[5,2],[6,2],[7,2],[8,2]]

player1Pos = [[1,1],[2,1],[2,2],[2,3],[2,4],[3,4],[4,4],[4,3],[4,2],[5,2]]

player2Pos = [[3,5],[4,5],[5,5],[6,5],[4,4],[4,3],[4,2],[5,2],[6,2],[7,2],[8,2]]

# player1Pos = [[-3,-2],[-2,-2],[-1,-2],[0,-2],[-2,0],[-3,-1],[-3,0],[-3,1],[-1,0],
#               [0,-1],[1,-1],[2,-1],[0,0],[0,1],[1,1],[1,2],[2,1],[2,2],[2,3],
#               [2,4],[3,3],[4,3],[4,4],[4,5],[3,5],[2,5],[4,6],[2,0]]

# player2Pos = [[3,-3],[3,-2],[4,-2],[3,1],[4,1],[4,2],[5,2],[5,3],[5,4],[5,5],
#               [3,0],[3,-1],[4,-1],[5,-1],[5,0],[5,1]]



player1AgentsNumber = len(player1Pos)
player2AgentsNumber = len(player2Pos)

player1Agents = [Agent(1,Pos(i,j)) for (i,j) in player1Pos]
player2Agents = [Agent(2,Pos(i,j)) for (i,j) in player2Pos]


network = Network()
network.r = 1
network.grid_size = 0.25
alpha = 3
network.add_agents(player1Agents)
network.add_agents(player2Agents)
network.draw_graph(pngname='orig_graph')



# possible_positions = network.get_possible_grid_positions()
# network.draw_graph(possible_positions = possible_positions,
#                    pngname='example_grid')
# best_positions = network.get_best_positions(possible_positions,2.,1)
# network.draw_graph(possible_positions = best_positions,pngname='example_result')


possible_positions = network.get_possible_positions()
best_positions = network.get_best_positions(possible_positions,alpha,1,mode=1)
# network.draw_graph(possible_positions = possible_positions,
#                    pngname='possible_positions')
network.draw_graph(possible_positions = best_positions)

possible_positions = network.get_possible_grid_positions()
network.draw_graph(possible_positions = possible_positions,
                   pngname='possible_positions')
print(len(possible_positions))


network.write_info_file(best_positions,'my_network_with_nodes.txt')
i = 1
for positions in combinations(possible_positions, 1):
```

```
    if positions != best_positions:
        i += 1
        network.write_info_file(positions,'my_network_with_nodes'+str(i)+'.txt')
```

[ ]:
```
mean_len_list = []
measure_list = []
cc_list,bc_list,dc_list = [],[],[]

drons = [Agent(pos=pos) for pos in best_positions]
network.add_drons(drons)
mean_len_list.append(nx.average_shortest_path_length(network.graph))
measure_list.append(1/network.get_value(alpha))
cc_list.append(network.get_value_cc())
bc_list.append(network.get_value_bc())
dc_list.append(network.get_value_dc())

network.draw_graph(pngname='graph.png')
network.del_drons(drons)

i = 1
for positions in combinations(possible_positions, 1):
    if positions != best_positions:
        drons = [Agent(pos=position) for position in positions]
        network.add_drons(drons)
        print(i)
        i += 1
        mean_len_list.append(nx.average_shortest_path_length(network.graph))
        measure_list.append(1/network.get_value(alpha))
        cc_list.append(network.get_value_cc())
        bc_list.append(network.get_value_bc())
        dc_list.append(network.get_value_dc())
        network.draw_graph(pngname='graph'+str(i)+'.png')
        network.del_drons(drons)
```

[ ]:
```
plot_sum_bar(mean_len_list,title='',xlabel='Наборы позиций',
             ylabel='Средняя длина кратчайших путей',
             legend_labels=None,pngname='mean_len_bars.png')
```

[ ]:
```
plot_sum_bar(measure_list,title='',xlabel='Наборы позиций',
             ylabel='Значение меры при alpha = '+str(alpha),
             legend_labels=None,pngname='mc_value_bars.png')
```

[ ]:
```
plot_sum_bar(cc_list,title='',xlabel='Наборы позиций',
             ylabel='Значение closeness centrality',
             legend_labels=None,pngname='cc_value_bars.png')
```

[ ]:
```
plot_sum_bar(bc_list,title='',xlabel='Наборы позиций',
             ylabel='Значение betweenness centrality',
             legend_labels=None,pngname='bc_value_bars.png')
```

[ ]:
```
plot_sum_bar(dc_list,title='',xlabel='Наборы позиций',
             ylabel='Значение degree centrality',
             legend_labels=None,pngname='dc_value_bars.png')
```

[ ]:
```
info_list = [parse_flowmon('p2p_manet.flowmon')]

for i in range(2,len(possible_positions)+1):
    info_list.append(parse_flowmon('p2p_manet'+str(i)+'.flowmon'))

plot_sum_bars(info_list,legend_labels=None)
```