Санкт-Петербургский государственный университет

Плеханова Таисия Михайловна

Выпускная квалификационная работа аспиранта

Использование теоретики-игрового подхода для повышения производительности сети MANET

Направление 01.06.01

Математика и механика

Образовательная программа «Прикладная математика и процессы управления»

Научный руководитель,
доктор физико-математических наук,
доцент
Громова Е. В.

Рецензент,
кандидат физико-математических наук,
Янковская Л. А.

Санкт-Петербург

2018

Оглавление

Введение

В данном исследовании рассматривается применение динамической теории игр к беспроводным децентрализованным самоорганизующимся сетям (MANET). MANET состоит из набора узлов/агентов с возможностью использования беспроводной технологии. Существует много приложений для MANET, начиная с обмена файлами в настройках персональной сети и заканчивая потоковой передачей видеоинформации между транспортными средствами . Сеть MANET реализует маршрутизацию с несколькими переходами — это тип связи в радиосетях, в которых зона покрытия сети больше, чем радиус действия одиночных узлов. Поэтому, чтобы передать информацию до конечного пункта, узел может использовать другие узлы в качестве маршрутизаторов. В этом исследовании рассматривается MANET, способствующую координированию действий спасательных отрядов в ситуации аварийного восстановления после природной катастрофы (т. е. после землетрясения, наводнений и т. д.), следовательно, основным условием является быстрота и простота установления связи между участниками спасательных отрядов. Улучшение производительности сети MANET при проведении спасательных операциях является одной из главных задач и в перспективе может спасти жизни.

Работа является продолжением исследования, опубликованного в [3]. В данной работе для улучшения производительности сети, во-первых, рассматривается гексагональная сетка, которая дает более реалистичное представление о передачи в сети. Во-вторых, вводится новый класс стратегий — стратегии «в перспективе», которые приводят к уменьшению длины игры и являются уникальными для предыдущей постановки задачи. В-третьих, задача расположения агентов на сети формируется, как динамическая кооперативная игра.

Постановка задачи

В данной выпускной квалификационной работе для достижения цели повышения производительности сети MANET при проведении спасательных операциях необходимо было решить следующие задачи:

1. расширить область поиска стратегий, предложенной в [3], для того, чтобы найти стратегии, которые имеют больший выигрыш, чем в [3];
2. рассмотреть кооперативный вариант игры [3];
3. написать программноый продукт, позволяющий исследовать игру на гексагональной сетке;
4. провести анализ полученных результатов в Network Simulator 3.

Обзор литературы

При написании выпускной квалификационной работы была использована следующая литература:

1. Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр.

   В данной книге описаны самые основные и актуальные на данный момент направления теории игр. Полно изложены теории кооперативных игр, антагонистических игр, неантагонистических игр. Все главы снабжены многочисленными примерами, демонстрирующие основные положения теорий. При написании работы использовалась Гл. 4 (Многошаговые игры).

2. Gromova E., Gromov D., Timonin N., Kirpichnikova A., Blackway S (2016). A Dynamic Game of Mobile Agent Placement in a MANET

   Данная статья является фундаментом для работы. В ней описана динамическая игра расположения дронов на сети MANET.

3. Винокуров В. М., Пуговкин А. В., Пшенников А. А. ,Ушарова Д. Н., Филатов А. С. Маршрутизация в беспроводных мобильных Ad hoc-сетях

   В этой статье описана сеть Ad hoc, особенности маршрутизации в данной сети. Описаны и промоделированы такие протоколы маршрутизации: DSR, OLSR, AODV, LANMAR, ZRP, OSPFv2.

4. Махмуд А. Ш., Поляков В. М. Оценка производительности протоколов мобильных Ad-hoc сетей (MANET)

   В данной статье приводится описание протоколов DSR и AODV и проводится оценка производительности указанных протоколов при помощи симулятора Network Simulator 2.

5. Климов И. А., Червинская Н. В. Сравнение протоколов маршрутизации для бесповодных AD-hoc сетей

   В данной статье проводится сравнение протоколов маршрутизации OLSR и AODV для сети Manet. Приводится описание работы сетей Mesh.

6. Standart Ecma-262

   Описание мультипарадигменного языка JavaScript.

7. Network simulator 3 documentation

   Описание установки и использования Network simulator 3.

Глава 1. Некооперативный вариант

1.1 Общая сетевая структура

Рассмотрим множество игроков $N = 1, \ldots, n$. Каждый игрок $i \in N$ имеет $M_i \neq \varnothing$ зафиксированных агентов (в процессе игры положение таких агентов на сетке не изменяется), расположенных в подпространстве $W = \{1, \ldots, W_x\} \times \{1, ..., W_y\} \subset R_2$, т. е. в узлах гексагональной сетки, где W_x, $W_y \in R$. Выбрана данная ориентация гексагона (см. рис. 1.1)), для упрощения дальнейших выкладок. Предполагаем, что в одной и той же вершине могут одновременно находиться агенты различных игроков. Позиция каждого агента описывается положительными координатами гексагональной сетки. Агенты обозначаются точками в узлах гексагональной решетки. Будем говорить, что между агентами v_i^p и v_i^s, $s \neq p$, игрока i установлена устойчивая связь (v_i^p, v_i^s), если они находятся в соседних узлах сетки. Полагаем, что между агентами различных игроков связи отсутствуют.

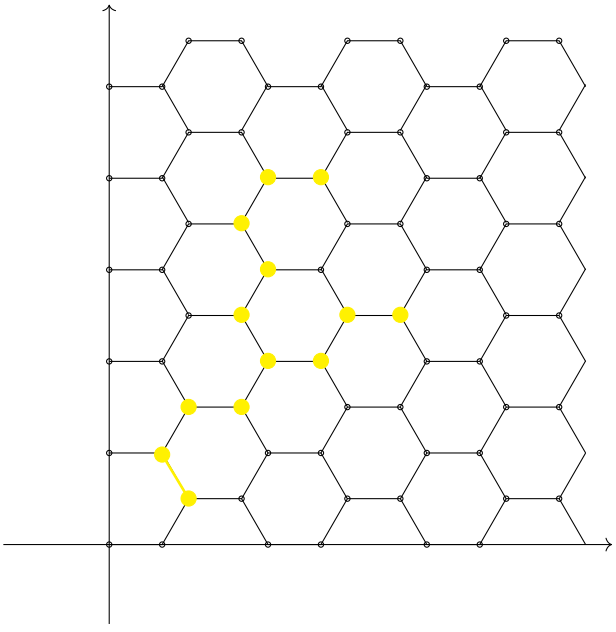

Рис. 1.1. Агенты и связи между ними

Следовательно, множество агентов $v_i^p \in M_i$, $i = 1, \ldots, n$ и связей $e = (v_i^p, v_i^s)$, $v_i^p \in M_i$, $v_i^s \in M_i$, $s \neq p$, $i = 1, \ldots, n$, где V — непустое конечное множество элементов, называемых вершинами (или точками, или узлами), E — конечное множество неупорядоченных пар элементов из V, называемых реб-

рами [10]. Подграфом R_i графа R, будем называть граф, все вершины которого принадлежат V, а все ребра принадлежат E.

Отметим, что $\bigcup_{i=1}^{n} R_i = R$. Граф R определяет общую сетевую структуру игры, в которой имеется $W_x \times W_y$ возможных позиций для агентов на сетке. Количество незанятых позиций агентами $|O|$ может быть оценено:

$$W_x \times W_y - (n_1 + n_2 + ... + n_N) \leq |O| \leq W_x \times W_y - \underset{i \in N}{max}\{n_i\}$$

Пусть подграф R_i связный, это условие гарантирует существование пути между любыми двумя его вершинами, где располагаются агенты игроков.

Путем в подграфе R_i будем называть такую последовательность $e = (e_1, \ldots, e_k,)$ ребер, что конец каждого предыдущего ребра совпадает с началом следующего. Длина пути $e = (e_1, ..., e_k)$ — число $d(e) = k$ ребер последовательности [10]. Длину каждого ребра гексагональной решетки, для упрощения вычислений, положим равной единице, следовательно, длина ребра $e_i = 1$.

Диаметром подграфа $D(R_i)$ будем называть число ребер в кратчайшем простом незамкнутом пути между двумя наиболее удаленными друг от друга вершинами [10] (см. рис.1.2).

$$D(R_i) = \underset{(v_k, v_l) \in V_i \times V_i}{max} d(v_k, v_l),$$

где $d(v_k, v_l)$ — путь между вершинами v_k и v_l.

Из связности подграфа R_i следует, что его диаметр может быть ограничен:

$$2(\sqrt{M_i} - 1) \leq D(R_i) \leq M_i - 1.$$

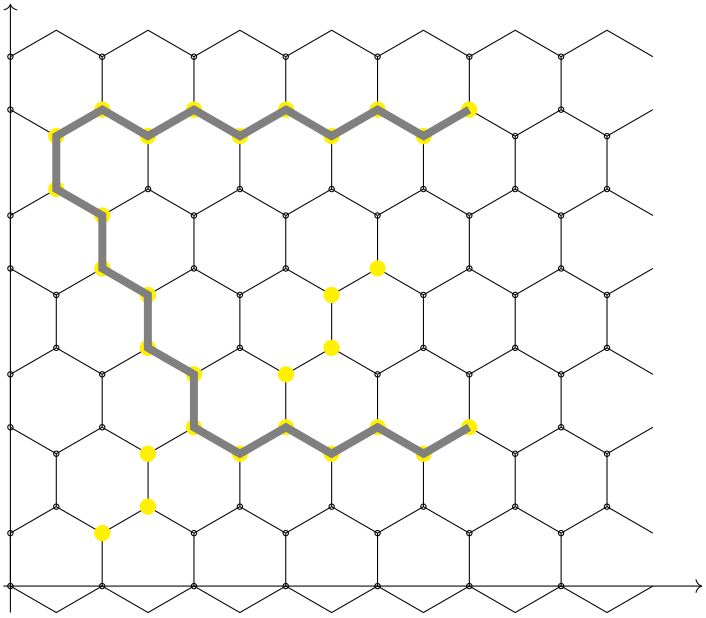

Рис. 1.2. Диаметр подграфа

1.2 Дрон — нефиксированный агент

Каждый игрок имеет одного нефиксированный агента — дрона.

Определение 1.2.1. Под нефиксированным агентом q_i, $i \in N$ будем подразумевать такого агента, положение которого на сетке в процессе игры может изменяться, и который обладает следующими свойством:

1. дрон может устанавливать связь с любым дроном q_i, $i \in N$, участвующем в игре;

2. дрон может устанавливать связь с любым зафиксированным агентом v_i^p, v_i^s, $v_i^s \in M_i$, $v_i^p \in M_i$, $s \neq p$, $i \in N$, участвующем в игре.

Связь дрона с зафиксированными агентами определяется аналогично связи между зафиксированными агентами.

Полагаем, что в начальном состоянии игры все дроны q_i расположены в такой позиции $k = (x, y)$, что не могут устанавливать связь ни с одним агентом ни одного игрока, т. е:

$$d(k, v_i^p) > 1, \tag{1.1}$$

$$d(k, v_i^s) > 1, \tag{1.2}$$

$$v_i^s \in M_i,\ v_i^p \in M_i,\ s \neq p,\ i \in N. \tag{1.3}$$

Подграф R_i^* — расширенный подграф, полученный из подграфа R_i добавлением дронов, множество вершин подграфа R_i^* определяется как:

$$V(R_i^*) = M_i \cup \bigcup_{i=1}^{n} q_i,\ i = 1, ..., n, \tag{1.4}$$

а множество ребер определяется связями на множестве $V(R_i^*)$ (см. рис. 1.3).

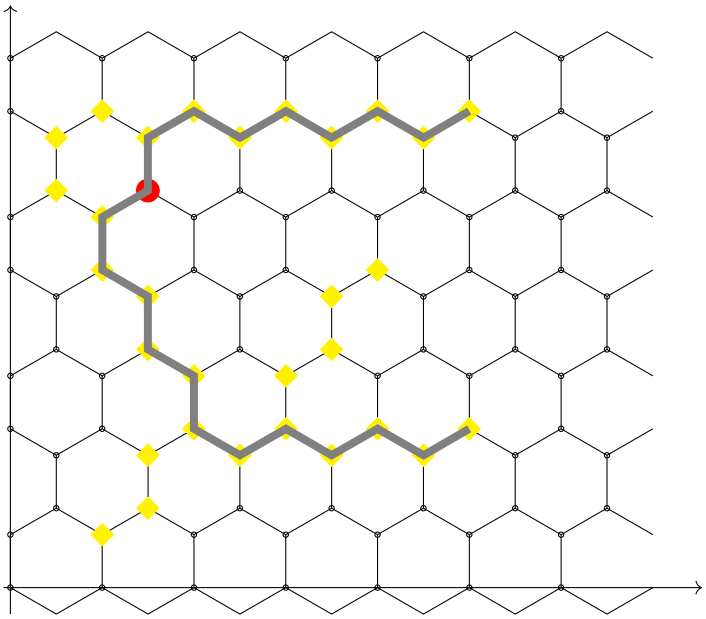

Рис. 1.3. Расширенный подграф

1.3 Стратегии и выигрыш

Каждый игрок ставит перед собой задачу расположить дрона так, чтобы минимизировать диаметр расширенного подграфа.

Игроки ходят последовательно. На каждом шаге игрок i выбирает одни элемент из своего множества альтернатив W_i. Полагаем, что в начальном состоянии игры, каждый игрок знает расположение зафиксированных агентов других игроков, также положим, что игрок «помнит» и «знает» свои предыдущие ходы и предыдущие ходы других игроков. Отметим, что дрон может располагаться не только в вершинах сетки, а также и в центре самого гексагона. Тогда на первом шаге игры под множеством стратегий W_i $(n-1)$ игроков понимается все возможные положения на гексагональной решетке, которые уменьшат диаметр расширенного подграфа R_i^* игрока i по сравнению с начальным диаметром («простые» стратегии), а также такие стратегии, которые уменьшат диаметр подграфа «в перспективе» (см. рис. 1.4). В данном случае под стратегией «в пер-

спективе» подразумевается: все игроки на начальном этапе знают количество дронов и расположение агентов других игроков, следовательно, $(n-1)$ игроков в начале игры могут «просчитать» стратегии на несколько шагов вперед, как будто у игрока не один дрон в управлении, а все. Максимальное количество «таких» шагов первого игрока соответствует количеству дронов (т.к. у каждого игрока только один дрон, следовательно, количеству игроков), второго — количество игроков минус 1, $(n-1)$ игрока — 1 (далее, мы введем понятие стратегии в «в переспективе», которое ограничит количество «таких» шагов до одного). Этот «просчёт» в большинстве случаев приводит к уменьшению длину игры, а также нахождению более выгодных стратегий, т. е. функция выигрыша для стратегии «в переспективе» по окончанию игры будет больше, чем для «простых» стратегии. Стоит отметить, что выбор на первом шаге i игроком таких стратегий «в перспективе» дает «нулевой» выигрыш на первом шаге ($H_i^1 = 0$, $i = 1, \ldots, n-1$).

Далее для игрока n на первом шаге и следующих шагах всех игроков под множеством стратегий W_i игрока i будем понимать только «простые» стратегии. Стоит отметить, что множество стратегий игрока i на каждом шаге зависит от его предыдущих ходов, и от предыдущих ходов других игроков.

Следовательно, каждый игрок i после первого шага решает задачу следующего вида:

$$q_i = \underset{i \in W_i}{argmin}\ D(R_i) \tag{1.5}$$

Так как одно из применений сетей MANET — это спасательные отряды, следовательно, будем полагать, что стратегии игроков будут «благожелательны» в том смысле, что игрок $i \in N$ при совершении своего хода, будучи в равной степени заинтересован в выборе последующих альтернатив, выбирает ту из них, которая более благоприятна для другого игрока [15], т. е. в нашем случае каждый игрок перемещает своего дрона так, чтобы не увеличить диаметры подграфов других игроков. Следовательно, данный процесс может быть сформулирован как конечношаговая игра с полной информацией.

Игра заканчивается когда ни один из игроков не может далее уменьшить диаметра своего расширенного подграфа. Функция выигрыша H_i игрока i за-

дается следующим образом:

$$H_i = d(R_i) - d(R_i^*) \geq 0. \tag{1.6}$$

Зная определение функции выигрыша 1.12, введем определения «простой» и «в перспективе» стратегии.

Определение 1.3.1. Будем говорить, что стратегия w_i является «простой» стратегией в конечношаговой игре n лиц, если функция выигрыша 1.12 имеет строгое неравенство и обладает следующим свойством:

1. «простая» стратегия w_i находится между агентами $v_i(x_i, y_i)$, $v_i(x_{i+1}, y_{i+1})$ игрока i такими, что расстояние между ними удовлетворяет следующему неравенству:

$$\rho(v_i, v_{i+1}) \leq 1 \tag{1.7}$$

Для определения понятия стратегия «в перспективе» введем определения области «в перспективе» и множества стратегий «в перспективе».

Определение 1.3.2. Область U_1^h называется областью «в перспективе» в конечношаговой игре 2 лиц для игрока 1 на первом шаге игры, если она принадлежит области, ограниченной прямыми $x = x_0$ и $x = x_1$ по оси $\overrightarrow{OX}$ и прямыми $y = y_0$ и $y = y_1$ по оси $\overrightarrow{OY}$, где (x_0, y_0), (x_1, y_2) координаты таких агентов $v_1^1(x_0, y_0)$ $v_2^1(x_1, y_2)$ игрока 1, которые удовлетворяют следующим условиям:

1. для любых агентов $v_1^1(x_0, y_0)$ $v_2^1(x_1, y_2)$ игрока 1 существуют агенты $v_1^2(x_0, y_0)$ $v_2^2(x_1, y_2)$ игрока 2;

2. расстояние между агентами (v_1^1, v_2^1) удовлетворяет неравенству:

$$2 < \rho(v_1^1, v_2^1) \leq 3 \tag{1.8}$$

Расширим понятие области «в перспективе» на игру $n \in N$ лиц. Для этого рассмотрим два следующих случая:

1. в вершинах $a = (x_0, y_0)$ и $b = (x_1, y_2)$ находятся k игроков, $k \in N$, $k \leq n$,

расстояние между вершинами:

$$\rho(a,b) \leq n+1-i; \tag{1.9}$$

2. в вершинах $a=(x_0,y_0)$ и $b=(x_1,y_2)$ находятся k игроков, $k \in N$, $k \leq n$, расстояние между вершинами:

$$n+1-i < \rho(a,b) < n+2-i. \tag{1.10}$$

Объединяя неравенства 1.8, 1.9, 1.10, получаем определение области «в перспективе».

Определение 1.3.3. Область U_i^{kh} называется областью «в перспективе» в конечношаговой игре n лиц для игрока i на первом шаге игры, если она принадлежит области, ограниченной прямыми $x=x_0$ и $x=x_1$ по оси $\overrightarrow{OX}$ и прямыми $y=y_0$ и $y=y_1$ по оси $\overrightarrow{OY}$, где (x_0,y_0), (x_1,y_2) координаты таких агентов $v_1^i(x_0,y_0)$ $v_2^i(x_1,y_2)$ игрока i, которые удовлетворяют следующим условиям:

1. для любых агентов $v_1^i(x_0,y_0)$ $v_2^i(x_1,y_2)$ игрока i существуют агенты $v_1^j(x_0,y_0)$ $v_2^j(x_1,y_2)$, $v_1^{j+1}(x_0,y_0)$ $v_2^j(x_1,y_2)$, $\ldots$, $v_1^{j+l}(x_0,y_0)$ $v_2^{j+l}(x_1,y_2)$ игроков j, $j+1$, $\ldots$, $j+l$, где $j \in k \leq n$;

2. расстояние между агентами (v_1^i, v_2^i) удовлетворяет неравенству:

$$2 < \rho(a,b) < n+2-i. \tag{1.11}$$

Определение 1.3.4. Множество $W_i^h = \bigcup_{k=1}^{m} U_i^{kh}$, где m — количество областей «в перспективе» игрока i, называется множеством областей стратегий «в перспективе» игрока i.

Множество областей «в перспективе» для игры 2 лиц изображена на рис. 1.4.

Определение 1.3.5. Будем говорить, что стратегия w_i является стратегией «в перспективе» w_i^h в конечношаговой игре n для игрока i на первом шаге, если w^h принадлежит множеству областей стратегий «в перспективе» W_i^h $(w_i^h \in W_i^h)$ и

функция выигрыша равна нулю, т. е.:

$$H_i(w_i^h) = 0, \text{для } i \leq n-1 \tag{1.12}$$

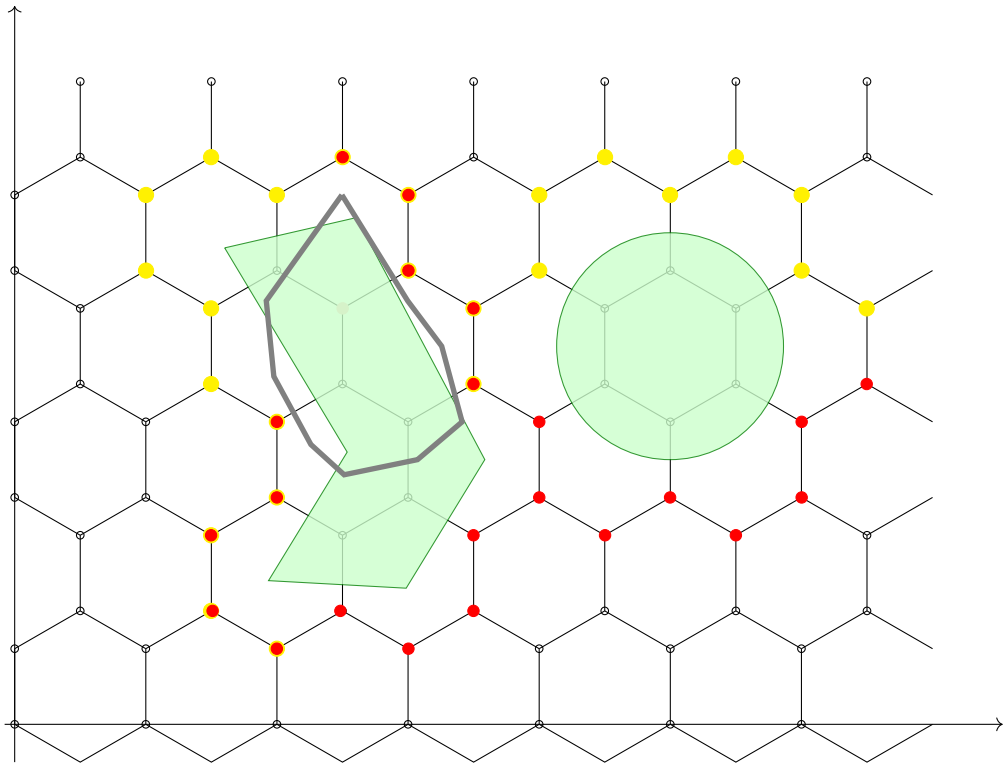

Рис. 1.4. Множество областей «в перспективе»

1.4 Формулы нахождения стратегий

1.4.1 «Простые» стратегии на гексагональной решетке

Найдем формулу для вычисления координат «простых» стратегий. Для этого из определения «простой» стратегии 1.3.1 и постановки задачи (дрон у каждого игрока один, радиус действия дрона равен радиусу действия зафиксированного агента, т. е. 1, возможность устанавливать дрона в центр гексагона) найдем все ситуации расположения агентов на сетке, между которыми можно установить дрона. Возможные ситуации расположения агентов между которыми можно установить дрона продемонстрированы на рис. 1.5, где красная окружность — агент v_{l0} (v_{r0}) игрока, а желтые окружности — агенты $v_{l1},\ldots,v_{l9}$ $(v_{r1},\ldots,v_{r9})$ — возможные ситуации расположения агентов игрока относительно агента v_{l0} (v_{r0}). На рисунке также изображены всевозможные ситуации для

установки дронов (зеленые окружности). В табл. 1.1, 1.2 представлена расшифровка рисунка, где расстояние — это расстояние между агентом v_0^k и агентом из столбца «агент», определяемое следующей формулой:

$$\rho(v_0^k, v_i^k) = \sqrt{(x_0^k - x_i^k)^2 + (y_0^k - y_i^k)^2}, i = 1, \ldots, 9, k = l, r, \qquad (1.13)$$

где (x_0^k, y_0^k) — координаты агента v_{k0}, (x_i^k, y_i^k) — координаты агента v_{ki}. Между агентами v_{k0} и v_{ki} устанавливается дрон под номером из столбца «дрон».

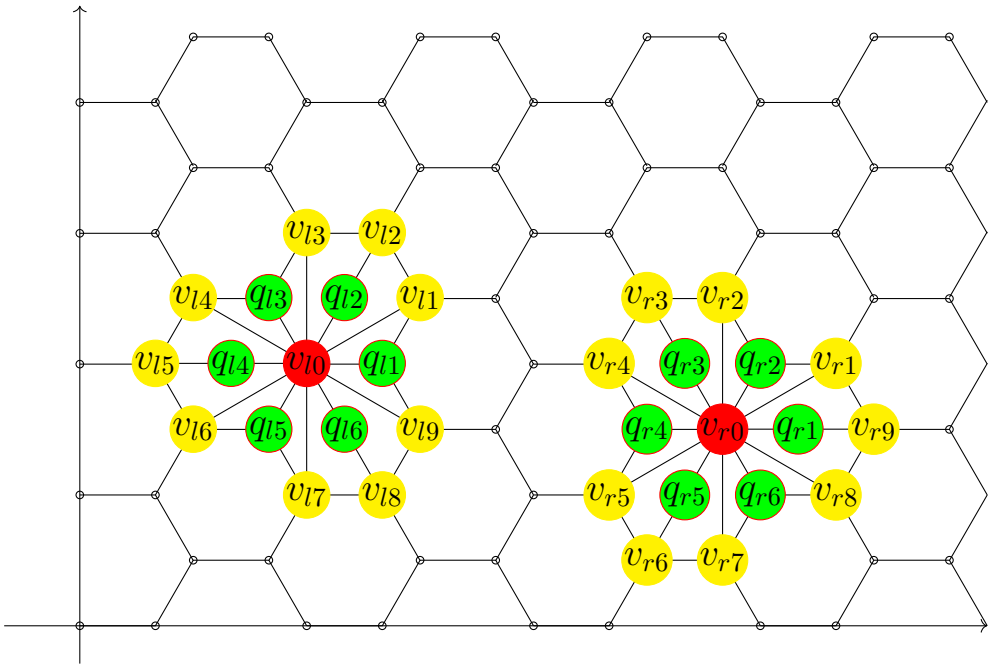

Рис. 1.5. Всевозможные комбинации «простых» стратегий

Агент	Дрон	Расстояние
v_{l1}	q_{l1}, q_{l2}	$\sqrt{3}$
v_{l2}	q_{l2}, q_{l3}	$\sqrt{3}$
v_{l3}	q_{l3}	1
v_{l4}	q_{l3}, q_{l4}	$\sqrt{3}$
v_{l5}	q_{l4}, q_{l5}	$\sqrt{3}$
v_{l6}	q_{l5}	1
v_{l7}	q_{l5}, q_{l6}	$\sqrt{3}$
v_{l8}	q_{l6}, q_{l1}	$\sqrt{3}$
v_{l9}	q_{l1}	1

Таблица 1.1. Расшифровка левой части рис. 1.5

Агент	Дрон	Расстояние
v_1	q_1, q_2	$\sqrt{3}$
v_2	q_2	1
v_3	q_2, q_3	$\sqrt{3}$
v_4	q_3, q_4	$\sqrt{3}$
v_5	q_4	1
v_6	q_4, q_5	$\sqrt{3}$
v_7	q_5, q_6	$\sqrt{3}$
v_8	q_6	1
v_9	q_6, q_1	$\sqrt{3}$

Таблица 1.2. Расшифровка правой части рис. 1.5

Как видно из рис. 1.5 и табл. 1.1, 1.2 все дроны $q_{r1}, \ldots, q_{r6}$ относительно

агента v_{r0} имеют идентичное расположение, как дроны $q_{l1}, \ldots, q_{l6}$ относительно агента v_{l0}, следовательно, для того чтобы найти формулу вычисления «простых» стратегий достаточно рассмотреть правую часть рис. 1.5. Для упрощения записи индекс $k = r$ при v и q в дальнейших выкладках опущен.

Из табл. 1.2 можно сделать вывод, что формулы вычисления и количество дронов будут зависеть от расстояния. Найдем формулы для дронов, где $\rho(v_0, v_i) = 1$. Из рис. 1.5, видно, что вектора $\overrightarrow{v_0v_3}$, $\overrightarrow{v_0v_5}$ получены из вектора $\overrightarrow{v_0v_9}$ поворотом относительно оси $\overrightarrow{OX}$, следовательно, рассмотрим вектор $\overrightarrow{v_0v_9}$ и найдем формулу для вычисления координат дрона q_1 и воспользуемся формулой поворота для вычисления остальных координат:

$$x_n = (x - x_0)\cos\phi - (y - y_0)sin\phi + x_0, \tag{1.14}$$

$$y_n = (x - x_0)\sin\phi + (y - y_0)cos\phi + y_0. \tag{1.15}$$

Формулы координат для дрона q_1 имеют вид:

$$x(q_1) = x_0 + 1, \tag{1.16}$$

$$y(q_1) = y_0 + 0. \tag{1.17}$$

Подставляя равенства 1.16, 1.17 в уравнения 1.14, 1.15 получаем формулы для установки дрона, когда расстояние между агентами $\rho(v_0, v_i) = 1$:

$$x(q_i) = x_0 + \cos\phi, \tag{1.18}$$

$$y(q_i) = y_0 + \sin\phi, \tag{1.19}$$

где ϕ — угол поворота вектора $\overrightarrow{v_0v_9}$ относительно оси $\overrightarrow{OX}$.

Рассмотрим случай, когда $\rho(v_0, v_i) = \sqrt{3}$. Из табл.1.1, 1.2 видно, что в этом случае у нас два возможных варианта для установки дрон. Из рис. 1.5, видно, что вектора $\overrightarrow{v_2v_0}$, $\overrightarrow{v_0v_4}$, $\overrightarrow{v_0v_5}$, $\overrightarrow{v_0v_7}$, $\overrightarrow{v_0v_8}$ получены из вектора $\overrightarrow{v_0v_1}$ поворотом относительно оси $\overrightarrow{OX}$, следовательно, рассмотрим вектор $\overrightarrow{v_0v_1}$ и найдем формулы для вычисления координат дрона q_2. Формулы для q_1 уже найдены. Формулы для q_2 имеют вид:

$$x(q_2) = x_0 + \frac{1}{2}, \tag{1.20}$$

$$y(q_2) = y_0 + \frac{\sqrt{3}}{2}. \tag{1.21}$$

Подставляя равенства 1.20, 1.21 в уравнения 1.14, 1.15 получаем::

$$x(q_i) = x_0 + \frac{1}{2}\cos\phi, \tag{1.22}$$

$$y(q_i) = y_0 + \frac{\sqrt{3}}{2}\sin\phi, \tag{1.23}$$

где ϕ — угол поворота вектора $\overrightarrow{v_1 v_0}$ относительно оси $\overrightarrow{OX}$.

Общий вид формул для координат «простой» стратегии, зная координаты агентов v_0 и v игрока i, расстояние между которыми удовлетворяет неравенству 1.7, имеет вид:

$$x_1 = x_0 + \cos\phi, y_1 = y_0 + \sin\phi, \text{при } \rho(v_0, v) = 1, \rho(v_0, v) = \sqrt{3} \tag{1.24}$$

$$x_2 = x_0 + \frac{1}{2}\cos\phi, y_2 = y_0 + \frac{\sqrt{3}}{2}\sin\phi, \rho(v_0, v) = \sqrt{3}, \tag{1.25}$$

где ϕ — угол поворота вектора $\overrightarrow{v_0 v}$ относительно оси $\overrightarrow{OX}$, а $\rho(v_0, v)$ — расстояние между агентами v_0 и v.

1.4.2 Стратегии «в перспективе» на гексагональной решетке

Найдем формулы вычисления координат стратегий «в перспективе». Для этого из определения стратегии «в перспективе» 1.3.5 и постановки задачи найдем все ситуации расположения агентов на сетке, между которыми можно установить дрона для конечношаговой игры двух лиц. Возможные ситуации расположения агентов между которыми можно установить дрона продемонстрированы на рис. 1.6. Все обозначения на рис. 1.6 аналогичны обозначениям на рис. 1.5. В табл. 1.3 и 1.4 приведена расшифровка рисунка. Отметит только тот факт, что во всех позициях, где находятся агенты игрока 1 ($v_{l0}, \ldots, v_{l11}$, $v_{r0}, \ldots, v_{r11},$), также находятся агенты игрока 2 (п.1 определения 1.3.5). Из рисунка видно, что дроны $q_{l1}, \ldots, q_{l6}$, $q_{r1}, \ldots, q_{r6}$ имеют аналогичные позиции, как дроны $q_{l1}, \ldots, q_{l6}$,

$q_{r1}, \ldots, q_{r6}$ из рис. 1.5, следовательно, для вычисления координат этих дронов можно воспользоваться формулами 1.24, 1.25, с заменой ρ на соответствующие значения из табл. 1.3 получаем:

$$x_1 = x_0 + \cos\phi, y_1 = y_0 + \sin\phi, \text{для } \rho(v_0, v) = 3, \rho(v_0, v) = \sqrt{7} \tag{1.26}$$

$$x_2 = x_0 + \frac{1}{2}\cos\phi, y_2 = y_0 + \frac{\sqrt{3}}{2}\sin\phi, \text{для } \rho(v_0, v) = \sqrt{7}, \tag{1.27}$$

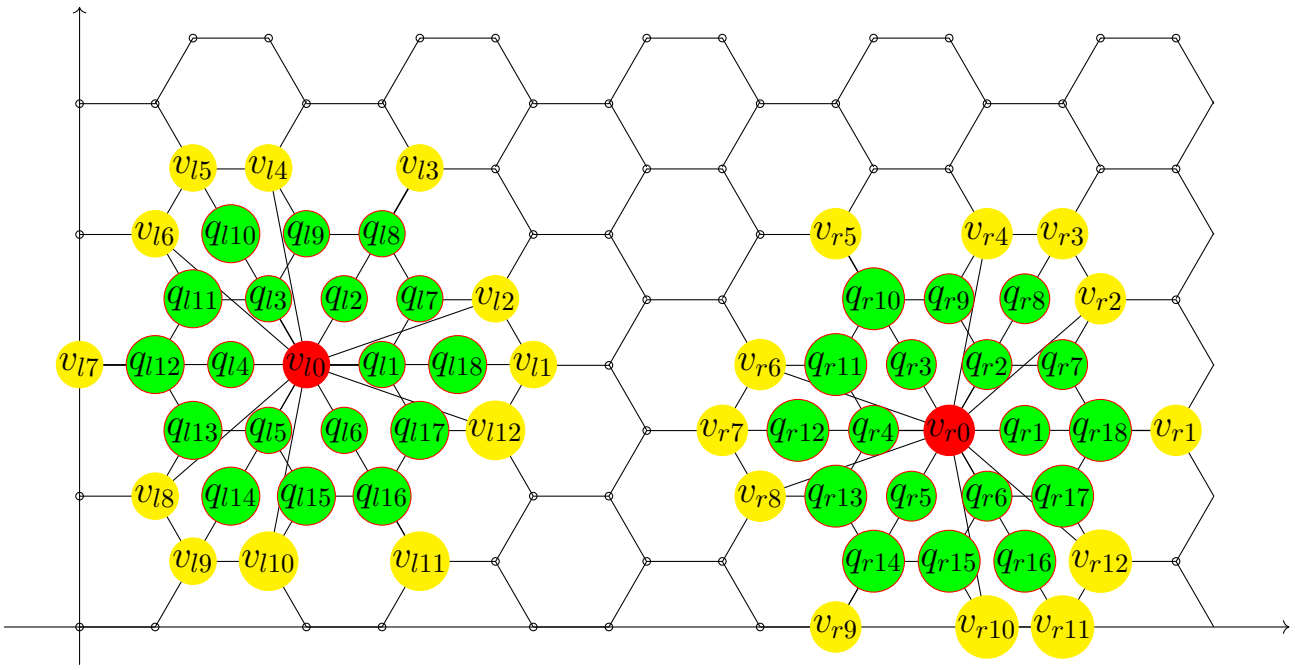

Рис. 1.6. Всевозможные комбинации стратегий «в перспективе» для конечношаговой игры двух лиц

Рассмотрим теперь дроны $q_{l7}, \ldots, q_{l18}$, $q_{r7}, \ldots, q_{r18}$. Из определений 1.3.1, 1.3.3, 1.3.4, 1.3.5 можно сделать предположение, что для игрока 1 не надо их вычислять, потому что игрок 2 сам их вычислит, т. к. для игрока 2 в его ход данные дроны попадут под определение «простых» стратегий. Для понимания хода размышлений рассмотрим следующий пример. Пусть происходит игра 2 лиц. Игрок 1 в своем множестве зафиксированных агентов M_1 имеет агента $v_1^1 \in M_1$ с координатами агента v_{l0} из рис. 1.6 и агента $v_2^1 \in M_1$ с координатами агента v_{l3} из рис. 1.6. Пусть игрок 2 также имеет агента $v_1^2 \in M_2$ с координатами агента v_{l0} из рис. 1.6 и агента $v_2^2 \in M_2$ с координатами агента v_{l3} из рис. 1.6. Пусть игрок 1 на своем первом шаге по формулам 1.26, 1.27 находит координаты дрона (в данном случае координаты дрона q_{l2} на рис. 1.6). По предположению, что координаты дрона q_{l8} не надо вычислять, не вычисляет их и

Агент	Дрон	Расстояние
v_{l1}	q_{l1}, q_{l18}	3
v_{l2}	q_{l1}, q_{l2}, q_{l18}, q_{l7}	$\sqrt{7}$
v_{l3}	q_{l3}, q_{l8}	3
v_{l4}	q_{l2}, q_{l3}, q_{l9}, q_{l10}	$\sqrt{7}$
v_{l5}	q_{l3}, q_{l10}	3
v_{l6}	q_{l3}, q_{l4}, q_{l10}, q_{l11}	$\sqrt{7}$
v_{l7}	q_{l4}, q_{l12}	1
v_{l8}	q_{l4}, q_{l5}, q_{l13}, q_{l14}	$\sqrt{7}$
v_{l9}	q_{l5}, q_{l14}	3
v_{l10}	q_{l5}, q_{l6}, q_{l14}, q_{l15}	$\sqrt{7}$
v_{l11}	q_{l6}, q_{l16}	3
v_{l12}	q_{l6}, q_{l1}, q_{l17}, q_{l18}	$\sqrt{7}$

Таблица 1.3. Расшифровка левой части рис. 1.6

Агент	Дрон	Расстояние
v_{l1}	q_{l1}, q_{l18}	3
v_{l2}	q_{l1}, q_{l2}, q_{l7}, q_{l8}	$\sqrt{7}$
v_{l3}	q_{l2}, q_{l8}	3
v_{l4}	q_{l2}, q_{l3}, q_{l8}, q_{l9}	$\sqrt{7}$
v_{l5}	q_{l3}, q_{l10}	3
v_{l6}	q_{l3}, q_{l4}, q_{l11}, q_{l12}	$\sqrt{7}$
v_{l7}	q_{l4}, q_{l12}	1
v_{l8}	q_{l4}, q_{l5}, q_{l12}, q_{l13}	$\sqrt{7}$
v_{l9}	q_{l5}, q_{l14}	3
v_{l10}	q_{l5}, q_{l6}, q_{l15}, q_{l16}	$\sqrt{7}$
v_{l11}	q_{l6}, q_{l16}	3
v_{l12}	q_{l6}, q_{l1}, q_{l16}, q_{l17}	$\sqrt{7}$

Таблица 1.4. Расшифровка правой части рис. 1.6

заканчивает свой ход, устанавливая своего дрона q_1 в позицию q_{l2}. Ход переходит к игроку 2. По свойствам дрона 1.2.1 и определению множества вершин расширенного подграфа 2.3, игрок 2 может рассматривать дрона q_1, как своего дополнительного зафиксированного агента, следовательно, по формулам 1.24, 1.25 игрок 2 находит координаты, которые соответствуют координатам q_{l8} из рисунка. Расширим предположение на игру n лиц, получим правило и формулу для вычисления стратегий «в перспективе».

Правило вычисления координат стратегий «в перспективе». На первом шаге игры игрок i должен вычислять координаты стратегии «в перспективе» между игроками v_0^i, v^i, удовлетворяющим свойствам 1, 2 определения 1.3.3, по следующим формулам:

$$x_1 = x_0 + \cos\phi, y_1 = y_0 + \sin\phi, \tag{1.28}$$
$$\text{для } 2 < \rho(v_0^i, v^i) < n + 2 - i \text{ и } \rho(v_0^i, v^i) = n + 2 - i,$$
$$x_2 = x_0 + \frac{1}{2}\cos\phi, y_2 = y_0 + \frac{\sqrt{3}}{2}\sin\phi, \tag{1.29}$$
$$\text{для } 2 < \rho(v_0^i, v^i) < n + 2 - i,$$

Глава 2. Кооперативный вариант

2.1 Кооперативный вариант без дронов

Пусть M — множество агентов всех игроков, объединенных вместе (максимальная коалиция), что соответствует их кооперативному поведению. Очевидно, что

$$|M| \leqslant |\bigcup_{i=1}^{n} M_i|, \quad i = 1, \ldots, n,$$

где n — количество игроков. Заметим, что в данной работе рассматривается только кооперация игроков в максимальную коалицию.

Множество агентов $q_l \in M$ и связей $e = (q_p, q_s)$, $q_p \in M$, $q_s \in M$, $s \neq p$, определяют граф $G = (T, K)$, где $T = \bigcup_{i=1}^{n} V$ — непустое конечное множество элементов, K — конечное множество неупорядоченных пар элементов из T. Граф G определяет общую сетевую структуру игры при кооперации игроков в максимальную коалицию.

Определим диаметр графа $D(G)$ при кооперации всех игроков (рис. 3.1), как

$$D(G) = \max_{(g_k, g_l) \in T_i \times T_i} d(g_k, g_l), \tag{2.1}$$

предполагая, что после объединения всех игроков в максимальную коалицию граф G является незамкнутым, т. е. существует хотя бы один подграф $G_i \in G$, который является незамкнутым. Пусть также граф G будет связным. Из связности графа G следует, что его диаметр может быть ограничен:

$$2(\sqrt{M} - 1) \leq D(G) \leq M - 1.$$

Определим функцию выигрыша игрока i следующим образом:

$$H_i^c = D(G) - D(R_i) \geqslant 0, \tag{2.2}$$

где $D(G)$ — диаметр графа при кооперативном варианте, R_i — диаметр подграфа игрока i при некооперативной постановке.

2.2 Кооперативный вариант с дронами

Пусть, как и в некооперативной постановке, каждый игрок i имеет в своем распоряжении управляемого агента — дрона, которого он может устанавливать в узлы решетки. Все характеристики дрона являются аналогичными из некооперативного варианта (опр. 1.2.1). Очевидно, что установка дрона в узле решетки приводит к расширению графа G, который обозначим как G^*. Множество вершин графа G^* определяется, как:

$$V(G^*) = M \cup \bigcup_{i=1}^{n} q_i,\ i = 1, \ldots, n. \tag{2.3}$$

Функцию выигрыша игрока i в данном случае определим, как:

$$H_i^{c*} = d(G^*) - d(R_i) \geqslant 0, \tag{2.4}$$

где $d(G^*)$ — диаметр расширенного графа при кооперативном варианте с установленными дронами.

Глава 3. Пример игры

3.1 Некооперативный вариант

3.1.1 Пример

В рамках данной работы было написано программное обеспечение, позволяющее исследовать описанную игру. Программное обеспечение написано на мультипарадигменном языке JavaScript. Все результаты данной главы (3) получены с помощью написанного программного обеспечения.

Рассмотрим игру для двух участников. Пусть игра Γ происходит на сетевой структуре, изображенной на рис.3.1. Множество игроков N состоит из двух участников: $N = \{1, 2\}$. Зафиксированные агенты первого игрока изображены в виде заполненных кружков желтого цвета (количество агентов $N_{ag}^1 = 18$), а второго игрока (количество агентов $N_{ag}^2 = 18$) — в виде заполненных кружков красного цвета. Дрон первого игрока будет изображаться в виде заполненного кружка чёрного цвета, а второго игрока — в виде заполненного кружка синего цвета. Положение каждого агента задается двумя координатами. В начальном состоянии игры диаметры подграфа первого и второго игроков равны ($d(R_1) = d(R_2) = 17$). Заполненная синим цветом область на рисунке изображает препятствие (например, озеро). Напомним, что в начальном состоянии дроны находятся в такой позиции на гексагональной решетки, что не могут устанавливать связь ни с одним агентом ни одного игрока.

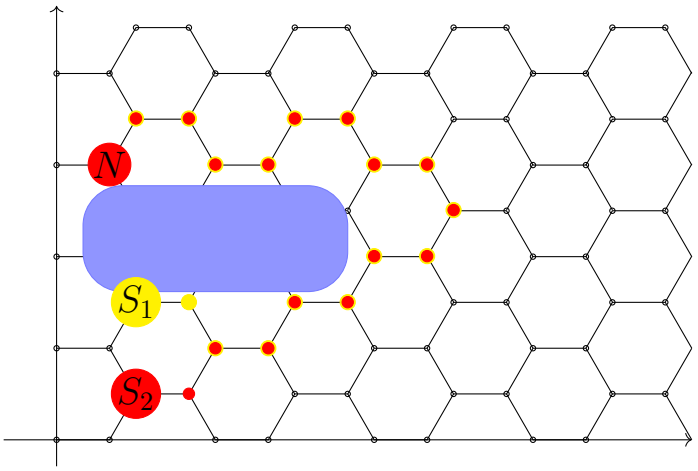

Рис. 3.1. Начальное состояние игры. Диаметры подграфов $d(R_1) = 17$, $d(R_2) = 18$

В табл. 3.1 показаны стратегии игрока 1 на первом шаге. Стратегии 1 и 2 являются стратегиями «в перспективе» для первого игрока, а стратегии $2-6$ — «простые» стратегии.

№ стратегии игрока 1	Координаты дрона	Функции выигрыша
1	(5, 4.47)	(0; 0)
2	(5.5, 4.47)	(0; 0)
3	(7.5, 5.34)	(2; 2)
4	(6.5, 5.34)	(2; 2)
5	(7.5, 7.07)	(1; 1)
6	(3, 6.2)	(1; 1)

Таблица 3.1. Стратегии игрока 1 на первом шаге игры

В табл. 3.2 показаны стратегии игрока 2 на первом шаге. Т. к. 15 агентов игрока 1 имеют позиции на гексагональной сетке, равные позициям 15-ти агентов игрока 2, следовательно, позиции для установки дрона будут для некоторых стратегий второго игрока одинаковыми с некоторыми стратегиями первого игрока. Поэтому, для упрощения записи, в табл. 3.2 в столбце «Координаты дрона 2» в некоторых ячейках номер стратегии из таблицы 3.1, которому соответствуют координаты дрона из столбца «Координаты дрона». На втором шаге игры ни один из игроков не может найти позицию для установки своего дрона, чтобы уменьшить диаметр своего расширенного подграфа R_i^*, следовательно, игра заканчивается на этом шаге.

3.1.2 Выводы

Из табл. 3.1, 3.2 видно, что выигрыш для стратегий «в перспективе» больше, чем для «простых» стратегий и игра заканчивается на 2 шаге.

№ стратегии игрока 1	№ стратегии игрока 2	Координаты дрона игрока 2	Функции выигрыша
1	1	(5.5, 5.34)	(6; 6)
1	2	3	(2; 2)
1	3	4	(2; 2)
1	4	5	(1; 1)
1	5	6	(1; 1)
2	6	(5.5, 5.34)	(6; 6)
2	7	3	(2; 2)
2	8	4	(2; 2)
2	9	5	(1; 1)
2	10	6	(1; 1)
3	11	5	(3; 3)
3	12	6	(3; 3)
4	13	(5.5, 5.34)	(5; 5)
4	14	5	(3; 3)
4	15	6	(3; 3)
5	16	4	(4; 4)
5	17	3	(4; 4)
5	18	6	(2; 2)
6	19	5	(2; 2)
6	20	4	(3; 3)
6	21	3	(3; 3)

Таблица 3.2. Стратегии игрока 2

3.2 Кооперативный вариант

3.2.1 Пример игры

В данном разделе мы рассмотрим кооперативный вариант игры.

Диаметр подграфа для кооперативного варианта игры, изображенной на рис. 3.1, без участия дронов, равен $D(G) = 17$. Функция выигрыша в этом случае имеет вид:

$$H_i^c = (0; 0) \tag{3.1}$$

Как видно из функции выигрыша, в данном случае кооперация игроков не дает выигрыша ни одному из игроков.

Введем в игру дронов. На рис. 3.2 показаны позиции дронов с максимальными функциями выигрыша для обоих игроков. В этом случае функции выигрыша равны:

$$H_i^c = (13; 13) \tag{3.2}$$

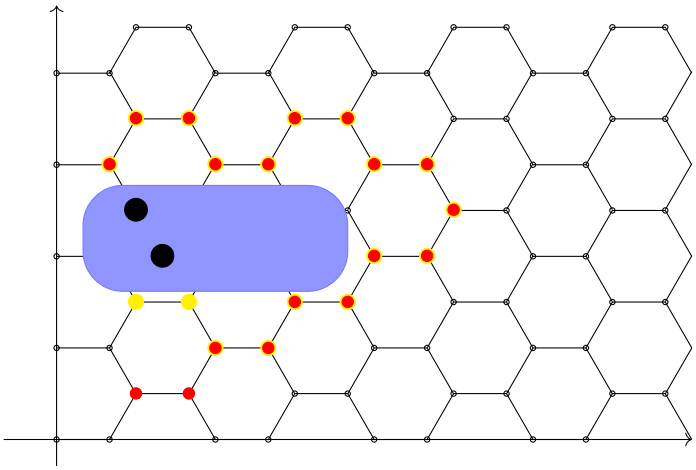

Рис. 3.2. Лучшие расположение дронов при кооперативной игре.

3.2.2 Выводы

Как видно из 3.2 при кооперативной постановке при введении дронов в игру функции выигрыша больше, чем при некооперативной постановке. Также, стоит отметить, что при максимальной кооперации процесс развития игры не зависит от того, который из игроков будет делать первый шаг.

Глава 4. Моделирование в Network Simulator 3

4.1 Конфигурация сети

В данной главе проводится моделирование, полученных результатов в главе 3.1, в Network Simulator 3. Целью было сравнение производительности сети для всех 21 сценариев из табл. 3.2. В табл. 4.1 представлена конфигурация сети, которая использовалась при моделировании. Выбран протокол AODV, потому что этот протокол устанавливает маршруты по требованию, а также AODV является более эффективным в плане пропускной способности, в особенности при увеличении размеров пакета или скорости узлов [13].

Параметр	Значение
Радиус распространения	105м
Количество агентов игрока 1	18
Количество агентов игрока 2	18
Время симуляции	2100 с
Источник сигнала игрока 1	S_1
Источник сигнала игрока 2	S_2
Пункт назначения	N
Физическая модель	DSSS at 11Mbps
Скорость передачи	CBR at 11Mbps
Размер данных	64bits
Протокол маршрутизации	AODV
Транспортный протокол	UDP

Таблица 4.1. Конфигурация сети

4.2 Результаты моделирования

В данной главе представлены результаты моделирования. В табл. 4.2 содержатся, полученные в Network Simulator. Параметр, по-которому идет сравнение — это средняя пропускная способность, т. е. количество пакетов, полученных в единицу времени. Последней строкой в таблице результаты, для сценария, когда дроны не используются.

№ стратегии игрока 1	№ стратегии игрока 2	Средняя пропускная способность	Функции выигрыша
1	1	45,125	(6; 6)
1	2	40,543	(2; 2)
1	3	39,764	(2; 2)
1	4	37,482	(1; 1)
1	5	37,320	(1; 1)
2	6	44,986	(6; 6)
2	7	39,337	(2; 2)
2	8	39,554	(2; 2)
2	9	37,521	(1; 1)
2	10	37,427	(1; 1)
3	11	41,553	(3; 3)
3	12	41,542	(3; 3)
4	13	44,723	(5; 5)
4	14	40,952	(3; 3)
4	15	41,154	(3; 3)
5	16	42,756	(4; 4)
5	17	42,958	(4; 4)
5	18	39,834	(2; 2)
6	19	39,546	(2; 2)
6	20	41,754	(3; 3)
6	21	41,359	(3; 3)
-	-	36,312	(0; 0)

Таблица 4.2. Результаты моделирования

4.3 Выводы

Из таблицы 4.2 видно, что лучшие результаты по пропускной способности у сценариев, которые имеют функции выигрыша больше, а максимума пропускная способность достигает при установке дрона в позиции, соответствующие стратегиям «в перспективе».

Заключение

Анализ улучшения производительности сети MANET при проведении спасательных операций в местах, где произошли природные катастрофы, показал, что одну из главных ролей играет грамотное сотрудничество поисковых команд.

Работа является расширением работы [3] на гексагональную решетку. Был введен новый класс стратегий — стратегии «в перспективе». С помощью нововведения получилось уменьшить длину игры и найти новые позиции для размещения дрона, которые дают лучший результат, чем предыдущий класс стратегий.

Была рассмотрена кооперативная постановка игры управления дронами в сети. Получено, что при кооперации игроков в максимальную коалицию, производительность сети улучшается, в сравнении с некооперативным вариантом.

Был написан программный продукт, позволяющий построить дерево решений для конечношаговых кооперативной и некооперативной игр на гексагональной решетке. Программное обеспечение было протестировано на игре двух лиц.

Была проведено моделирование, полученных результатов, в Network Simulator 3. Было установлено, что применение теоретико-игрового подхода к задаче улучшения производительности сети даёт положительные результаты.

Работа может быть расширена на треугольную решетку. Также следует провести сравнительный анализ для рассмотренных сеток (квадратной, гексагональной), с целью выявления какой тип сетки является предпочтительнее для определенного типа поверхности, на котором ведется поисковая операция. Также стоит рассмотреть кооперацию игроков не в максимальную коалицию. Исследовать другие протоколы маршрутизации в Network Simulator 3, например, DSR, OLSR.

Некоторые результаты исследования отражены в работах [4], [8], [9], [16], [17].

Литература

[1] Anjum S. S., Noor R. M., Anisi M. H. Survey on MANET based communication scenarios for search and rescue operations, in IT Convergence and Security (ICITCS) // 5th International Conference on. IEEE. 2015. P. 1–5.

[2] Standart Ecma-262 [Электронный ресурс]: `https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf` (Дата обращения: 02.02.18).

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

[4] Gromova E., Gromov D., Plekhanova T., Kirpichnikova A., Blakeway S. Increasing network perfomance of a MANET using game-theoretic approach to drone positioning // Ad Hoc Networks.

[5] Han Z., Niyato D., Saad W., Basar T., Hjorungnes A. Game Theory in Wireless and Communication Networks. Theory, Models, and Applications. New York: Cambridge University Press, 2012. 530 p.

[6] LaTeX on Wikibooks [Электронный ресурс]: `URL:http://en.wikibooks.org/wiki/LaTeX` (Дата обращения: 02.06.18).

[7] Network simulator 3 documentation [Электронный ресурс]: `https://www.nsnam.org/documentation/` (Дата обращения: 09.05.18).

[8] Plekhanova T., Gromova E., Kirpichnikova A., Blakeway S. A multistage game of mobile agents placement on a hexagonal grid - MANET // Тезисы XXI международной конференции «Теория игр и менеджмент», 2017.

[9] Plekhanova T., Gromova E., Gromov D., Kirpichnikova A., Blakeway S. The Strategic Placement of Mobile Agents on a Hexagonal Graph using Game Theory // Proc. of the IEEE conference ICAT 2017. doi:10.1109/ICAT.2017.8171635

[10] Robin J. Introduction to Graph Theory // Edinburgh: Oliver and Boyd, 1972. 209 p.

[11] Винокуров В. М., Пуговкин А. В., Пшенников А. А., Ушарова Д. Н., Филатов А. С. Маршрутизация в беспроводных мобильных Ad hoc-сетях // Управление, вычислительная техника и информатика, 2010, С. 288–292.

[12] Климов И. А., Червинская Н. В. Сравнение протоколов маршрутизации для бесповодных AD-hoc сетей // Збірник наукових праць XIII науково-технічної конференції аспірантів та студентів, ДонНТУ, 2013. С. 76–80.

[13] Махмуд А. Ш., Поляков В. М. Оценка производительности протоколов мобильных Ad-hoc сетей (MANET) // DOI:10.18413/2518-1092-2016-1-4-64-71.

[14] Новиков Д. А. Игры и сети // Математическая теория игр и её приложения, 2010. С. 107–124.

[15] Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр, 2012. С. 424.

[16] Плеханова Т. М., Громова Е. В. Кооперативная динамическая игра на сети MANET // Труды «XLIX международная научная конференция аспирантов и студентов «Процессы управления и устойчивость», 2018.

[17] Плеханова Т. М., Громова Е. В. Сравнительный анализ производительности сети MANET при применении теоретико-игрового подхода для различных типов решетки // Тезисы Второй всероссийской междисциплинарной конференции «Социофизика и социоинженерия», 2018.

Приложение

compute.js

```
function computeGame(game, i, j, color) {
    var color = color || 'red';
    var distanceMatrix = getDistanceMatrix(game.situation['
      player' + i].X, game.situation['player'+i].Y);
    var markBefore = computeMark(game.situation['player' + i
      ]);
    var markBefore2 = computeMark(game.situation['player' +
      j]);
    var droneCoord = getDroneCoords(distanceMatrix, game.
      situation['player' + i].X, game.situation['player' + i
      ].Y);
    var distanceAfterDrone = [];
    var distanceAfterDrone2 = [];
    var payOff = [];
    var newDroneCoord = [];
    var count = 0;
    var markAfter = 0;
    var markAfter2 = 0;

    for(count; count < droneCoord.length - 1; count++) {
      markAfter = computeMarkWithDrone(game.situation['
        player' + i], droneCoord[count]);
      if (markAfter < markBefore) {
        distanceAfterDrone.push([markAfter, markBefore -
          markAfter]);
        markAfter2 = computeMarkWithDrone(game.situation['
          player' + j], droneCoord[count]);
        distanceAfterDrone2.push([markAfter2, markBefore2 -
          markAfter2]);
```

```
            payOff.push([markBefore - markAfter, markBefore2 -
                markAfter2]);
            newDroneCoord.push(droneCoord[count]);
        }
    }
    return newDroneCoord;
}
```

draw.js

```
function drawGame(game) {
    game.getCtx().fillStyle = 'white';
    game.getCtx().fillRect(0, 0, game.getWidth(), game.
        getHeight());
    game.drawHexGrid();
    game.drawSqureGrid('hexagonalGrid', game.place.SCALE);
    game.situation.player1.X.forEach(function (item, i, arr)
        {
      arr[i] /= 50000;
    });
    game.situation.player1.Y.forEach(function (item, i, arr)
        {
      arr[i] /= 50000;
    });
        game.drawIniteState(game.situation.player1, 'yellow'
            );
    game.drawIniteState(game.situation.player2, 'red', -4);
    game.drawIniteState(game.situation.player3, 'green', 4);
    game.drawGuidesXY(1, 1);
    game.drawBarrier(game.barrierCoords.barrier1);
    game.drawBarrier(game.barrierCoords.barrier2, 'green');
    game.drawBarrier(game.barrierCoords.barrier3, 'blue');
    game.drawBarrier(game.barrierCoords.barrier4, 'blue');

    return game;
```

```
}
```

function.js

```
function getPerspectiveDrone(X, Y) {
  // possible coordinates
  var possCoord = [['X', 'Y', 'j', 'i']];
  var i = 1;
  var x1, x2, y1, y2;
  // get all possible coordinates
  for (var j = 0; j < X.length; j++) {

    for (i; i < X.length; i++) {

      if (distanceMatrix[j][i] > 2 && distanceMatrix[j][i]
        <= 3) {

        if (Y[i] == Y[j]) {
          x1 = x2 = round((X[i] + X[j]) * 0.5, 3);
          y1 = Y[i] + 0.5;
          y2 = Y[i] - 0.5;
        }

        if (Y[j] > Y[i] && X[j] < X[i]) {
          x1 = X[j];
          x2 = X[i];
          y1 = Y[j] - 1;
          y2 = Y[i] + 1;
        }

        if (Y[j] < Y[i] && X[j] > X[i]) {
          x1 = X[i];
          x2 = X[j];
          y1 = Y[i] - 1;
          y2 = Y[j] + 1;
```

```
        }

        if (Y[j] < Y[i] && X[j] < X[i]) {
          x1 = X[j];
          x2 = X[i];
          y1 = Y[j] + 1;
          y2 = Y[i] - 1;
        }

        if (Y[j] > Y[i] && X[j] > X[i]) {
          x1 = X[i];
          x2 = X[j];
          y1 = Y[i] + 1;
          y2 = Y[j] - 1;
        }

        possCoord.push([x1, y1, j, i], [x2, y2, j, i]);

      }
    }
    i = j + 2;
}
// get drone coordinates
//
var allDroneCoords = []; // drone coordinates with
    hexagon center
var notDrone = false;
// console.log('possCoord', possCoord);
//checking if coordinates have been in array already
for (j = 1; j < possCoord.length; j++) {
  notDrone = X.some(function (Xi, index, array) {
    return Xi == possCoord[j][0] && Y[index] ==
      possCoord[j][1];
```

```
    });
    if (notDrone == false) {
      allDroneCoords.push(possCoord[j]);
    }
  }
  // console.log('allDroneCoords', allDroneCoords);

  // checking if coordinates the same
  allDroneCoords.forEach(function (item, i, array) {
    var arr = allDroneCoords.slice(i + 1);
    var count = 0;
    arr.forEach(function (it, index, arr) {
      if (item[0] == it[0] && item[1] == it[1]) {
        allDroneCoords.splice(i + 1 + index - count, 1);
        count++;
      }
    });
  });

  return allDroneCoords;
}


function drawConnections(coords, ctx) {
  var ctx = ctx || this.getCtx();
  var SCALE = this.place.SCALE;
  var height = this.getHeight();

  // start draw
  ctx.beginPath();

  ctx.strokeStyle = 'red';
  ctx.lineWidth = 2;
```

```
    ctx.moveTo(coords.X[0] * SCALE, height - coords.Y[0] *
      SCALE);

    for (var i = 0; i < coords.X.length - 4; i++) {
      ctx.lineTo(coords.X[i + 1] * SCALE, height - coords.Y[
        i + 1] * SCALE);
    }
    // get line

    ctx.stroke();
    ctx.closePath();
}

/**
 * drawGuidesXY
 * @param {number} X0
 * @param {number} Y0
 * @param {number} SCALE
 * @param {number} ctx
 */
function drawGuidesXY(X0, Y0, SCALE, ctx) {
    var text = '' + X0 + ', ' + Y0;
    var ctx = ctx || this.getCtx();
    var SCALE = SCALE || this.place.SCALE;
    var height = this.getHeight();
    var width = this.getWidth();

    var X0 = (X0 * SCALE) || 0;
    var Y0 = (height - Y0 * SCALE) || height;

    // Drawing of square grid
    ctx.beginPath();
```

```
ctx.strokeStyle = "black";
ctx.lineWidth = 2;
ctx.fillStyle = 'black';
ctx.font = "14px Georgia";
//draw vertical
ctx.moveTo(X0, 0);
ctx.lineTo(X0, height);
// arrows x
ctx.moveTo(X0, 0);
ctx.lineTo(X0 + 5, 10);

ctx.moveTo(X0, 0);
ctx.lineTo(X0 - 5, 10);

ctx.fillText('x', X0 - 15, 15);
// draw horiz
ctx.moveTo(0, Y0);
ctx.lineTo(width, Y0);

// arrows y
ctx.moveTo(width, Y0);
ctx.lineTo(width - 10, Y0 - 5);

ctx.moveTo(width, Y0);
ctx.lineTo(width - 10, Y0 + 5);

ctx.fillText('y', width - 15, Y0 + 15);
// text X0 and Y)

ctx.fillText(text, X0 + 5, Y0 - 5);

ctx.stroke();
```

```
    ctx.closePath();
}

/**
 * add coordinates of drone to arrays X and Y
 * @param {object} XY
 * @param {object} droneCoords
 * @returns {object}
 */
function setDroneToXY(XY, droneCoords) {
    var X = XY.X.slice();
    var Y = XY.Y.slice();
    var index = X.length - 1;

    X.splice(index, 0, droneCoords[0]);
    Y.splice(index, 0, droneCoords[1]);

    XYwithDrone = {
        X: X,
        Y: Y
    };

    return XYwithDrone;
}



/**
 * get Drone Coordinates
 * @param {array} distanceMatrix
 * @param {array} X
 * @param {array} Y
 * @param {number} quantityDrone
 * @returns {Array}
```

```
 */
function getDroneCoords(distanceMatrix, X, Y) {

  // possible coordinates
  var possCoord = [['X', 'Y', 'j', 'i']];
  var i = 1;
  var x1, x2, y1, y2;
  // get all possible coordinates
  for (var j = 0; j < X.length; j++) {

    for (i; i < X.length; i++) {

      if (distanceMatrix[j][i] > 1.71 && distanceMatrix[j
        ][i] <= 1.75) {

        if (Y[i] == Y[j]) {
          x1 = x2 = round((X[i] + X[j]) * 0.5, 2);
          y1 = Y[i] + 0.5;
          y2 = Y[i] - 0.5;
        }

        if (Y[j] > Y[i] && X[j] < X[i]) {
          x1 = X[j];
          x2 = X[i];
          y1 = Y[j] - 1;
          y2 = Y[i] + 1;
        }

        if (Y[j] < Y[i] && X[j] > X[i]) {
          x1 = X[i];
          x2 = X[j];
          y1 = Y[i] - 1;
          y2 = Y[j] + 1;
```

```
        }

        if (Y[j] < Y[i] && X[j] < X[i]) {
          x1 = X[j];
          x2 = X[i];
          y1 = Y[j] + 1;
          y2 = Y[i] - 1;
        }

        if (Y[j] > Y[i] && X[j] > X[i]) {
          x1 = X[i];
          x2 = X[j];
          y1 = Y[i] + 1;
          y2 = Y[j] - 1;
        }

        possCoord.push([x1, y1, j, i], [x2, y2, j, i]);

      }
    }
    i = j + 2;
}
// get drone coordinates
// console.log('possCoordinates\n\t', possCoord.join(';\
   n \t') );
// console.log('possCoord.lenght\n\t', possCoord.length)
   ;

var allDroneCoords = []; // drone coordinates with
   hexagon center
var notDrone = false;
// console.log('possCoord', possCoord);
```

```
    //checking if coordinates of drone == coordinates of
       player agents
    for (j = 1; j < possCoord.length; j++) {
      notDrone = X.some(function (Xi, index) {
        return (Xi + 0.3 > possCoord[j][0]) && (Xi - 0.3 <
           possCoord[j][0]) && Y[index] == possCoord[j][1];
      });
      if (notDrone == false) {
        allDroneCoords.push(possCoord[j]);
      }
    }
    // console.log('allDroneCoords', allDroneCoords);

    // checking if coordinates the same
    allDroneCoords.forEach(function (item, i) {
      var arr = allDroneCoords.slice(i + 1);
      var count = 0;
      arr.forEach(function (it, index) {
        if ((item[0] + 0.3 > it[0]) && (item[0] - 0.3 < it
           [0]) && item[1] == it[1]) {
          allDroneCoords.splice(i + 1 + index - count, 1);
          count++;
        }
      });
    });

    return allDroneCoords;
}



// /**
//  * drawGraph description
//  * @param  {object} brunches
```

```
//  * @param  {object} ctx
//  * @return {type}
//  */
function drawGraph(brunches, ctx) {
  var ctx = ctx || this.getCtx();
  var brunches = brunches || this.brunches;
  var strategy = [[]];
  var vertex = [{
    X: this.getWidth() / 2,
    Y: this.getHeight() - 10,
    angle: 0,
  }];
  var count = 0;
  var text = brunches.strategy1.combineXY();
  var strategies = brunches.strategy1.strategies;
  var nextStrategies = [];

  vertex = vertex.concat(drawPluentyOfBeams(ctx, brunches.
     strategy1.X.length, vertex[0], true,
    text));

  while (strategies.length && count < 10) {

    for (var i = 0; i < strategies.length; i++) {


      text = brunches['strategy' + strategies[i]].
        combineXY();
      vertex = vertex.concat(drawPluentyOfBeams(ctx,
        brunches['strategy' +
        strategies[i]].X.length, vertex[strategies[i] -
          1],
        brunches['strategy' + strategies[i]].player, text)
```

```
            );

        if (brunches['strategy' + strategies[i]].strategies
          != true) {
          nextStrategies = nextStrategies.concat(brunches['
            strategy' + strategies[i]].strategies);
        }

      }
      for (var j = 0; j < nextStrategies.length; j++) {
        strategies[j] = nextStrategies[j];
      }

      nextStrategies = [];
      count++;
    }

    return 0;
}

// /**
// * drawPluentyOfBeams
//   * @param  {type}  context
//   * @param  {type}  quantity
//   * @param  {type}  startCoord
//   * @param  {Boolean} isRound
//   * @param  {array} text
//   * @param  {number} lengthBeam
//   * @return {type}
//   */

function drawPluentyOfBeams(ctx, quantity, startCoord,
  isRound, text, lengthBeam) {
```

```
  var angle = [];
  var finishCoord = [];
  var lengthBeam = lengthBeam || 150;

  var constAngle = (180) / (quantity + 1);

  if (quantity < 10) {
    for (var i = 0; i < quantity; i++) {
      angle[i] = constAngle * (i + 1);
      if (startCoord.angle < 90 && startCoord.angle > 0) {
        angle[i] = angle[i] - startCoord.angle;
      } else if (startCoord.angle > 90) {
        angle[i] = angle[i] + (startCoord.angle - 90);
      }
    }
  } else {
    angle = 360 / (quantity + 1);
  }

  for (var i = 0; i < quantity; i++) {
    finishCoord[i] = drawGraphBeam(ctx, startCoord,
      isRound, angle[i], text[i], lengthBeam);
  }

  return finishCoord;
}
//
// /**
//  * [drawGraphBeam description]
//  * @param {[type]} context [description]
//  * @param {[object ={X:,Y:}]} startCoord [
   description]
//  * @param {Boolean} isRound [description]
```

```
//  * @param  {num} angle      [angle in the degrees]
//  * @param  {[number]}  lengthBeam [default = 30]
//  *
//  * @return {[type]}              [description]
//  */
function drawGraphBeam(ctx, startCoord, isRound, angle,
    text, lengthBeam) {
  var lengthBeam = lengthBeam || 50;
  var finishCoord = {
    X: startCoord.X - lengthBeam * Math.cos(inRad(angle)),
    Y: startCoord.Y - lengthBeam * Math.sin(inRad(angle)),
    angle: angle
  };
  ctx.save();
  ctx.translate(startCoord.X, startCoord.Y);
  ctx.beginPath();
  ctx.lineWidth = 2;
  ctx.strokeStyle = 'black';

  if (isRound) {
    ctx.arc(0, 0, 5, 0, 2 * Math.PI, false);
    ctx.fillStyle = 'black';
    ctx.fill();
  } else {
    var coordRectX = -5;
    var coordRectY = -5;
    // /**
    //  * [context.fill(x, y, width, height) The fillRect
      () method draws a "filled" rectangle.
    //  * The default color of the fill is black. Use the
      fillStyle property to set a color, gradient,
    //  * or pattern used to fill the drawing.]
    //  * @param  {[number]}  x     [The x-coordinate of
```

```
      the upper-left corner of the rectangle]
    //  * @param {[number]} y [The y-coordinate of the
       upper-left corner of the rectangle]
    //  * @param {[number]} width [The width of the
      rectangle, in pixels]
    //  * @param {number} heigth [The height of the
      rectangle, in pixels]
    //  * @return {[type]}
    //  */
    ctx.fillStyle = 'green';
    ctx.fillRect(coordRectX, coordRectY, 10, 10);
}
ctx.rotate(inRad(angle));
ctx.moveTo(0, 0);
ctx.lineTo(-lengthBeam, 0);
// arrows
ctx.moveTo(-lengthBeam, 0);
ctx.lineTo(-lengthBeam + 5, 5);

ctx.moveTo(-lengthBeam, 0);
ctx.lineTo(-lengthBeam + 5, -5);
// /**
//  * [context.fillText The fillText() method draws
    filled text on the canvas.
//  * The default color of the text is black.]
//  * @param {[type]} text [Specifies the text that
     will be written on the canvas]
//  * @param {[type]} x [ The x coordinate
    where to start painting the text (relative to the
    canvas)]
//  * @param {[type]} y [ The y coordinate
    where to start painting the text (relative to the
    canvas)]
```

```
    //   * @param  {[type]} maxWidth [Optional. The maximum
       allowed width of the text, in pixels]
    //   * @return {[type]}           [description]
    //   */
    ctx.font = "14px Georgia";
    ctx.fillText(text, -lengthBeam + 10, -2);

    ctx.stroke();
    ctx.closePath();
    ctx.restore();

    return finishCoord;
}



// http://algolist.manual.ru/maths/graphs/shortpath/
   dijkstra.php
// S array = considerPlayer
// B array = payMatrix
// C array =
// A[i, k] =
// Array have to be sorted
/**
 * Algorithm Dijkstra
 * @param {Array} payMatrix
 * @returns {{space: [number], count: number, flag:
    boolean}/*}
 */
function dijkstra(payMatrix) {

    var LENGTH = payMatrix[0].length;
    var considerPlayer = [0];
    var space = [0];
```

```
for (var i = 1; i < LENGTH; i++) {
  space[i] = Infinity;
  considerPlayer[i] = 0;
}

var maxCount = 0;
var unConsiderPlayer = [];
var unConsiderSpace = [];

while (considerPlayer.some(isConsider) && maxCount <
   300) {
  unConsiderPlayer = [];
  unConsiderSpace = [];

  for (var count = 0; count < considerPlayer.length;
     count++) {
    if(considerPlayer[count] == 0) {
      unConsiderPlayer.push(count);
    }
  }

  for (count = 0; count < unConsiderPlayer.length; count
     ++) {
    unConsiderSpace[count] = space[unConsiderPlayer[
       count]];
  }

  var min = getMinOfArray(unConsiderSpace, 0);
  var indexConsider = unConsiderPlayer[unConsiderSpace.
     indexOf(min)];

  considerPlayer[indexConsider] = 'consider';
```

```
        for (i = 0; i < LENGTH; i++) {
          if (payMatrix[indexConsider][i] == 1) {
            space[i] = Math.min(space[i], payMatrix[
              indexConsider][i] + space[indexConsider]);
          }
        }
      maxCount++;

    }

    var spaceObj = {
      space: space,
      count: maxCount,
      flag: considerPlayer.some(isConsider),
      considerPlayer: considerPlayer
    };
    return spaceObj;
}

function getMinOfArray(numArray, indexSearch) {
    var array = numArray.slice(indexSearch);
    // console.log(array);
    return Math.min.apply(null, array);
}

function isConsider(number) {
    return number == 0;
}

// /**
//  * [getPayMatrix description]
//  * @param  {[type]} distanceMatrix [description]
//  * @return {[type]}                [description]
```

```
// */
function getPayMatrix(distanceMatrix) {
  var length = distanceMatrix[0].length;
  var payMatrix = getMatrix(length, length);

  for (var j = 0; j < length; j++) {
    for (var i = 0; i < length; i++) {
      if (distanceMatrix[j][i] > 0.95 && distanceMatrix[j
        ][i] < 1.05) {
        payMatrix[j][i] = 1;
      } else {
        payMatrix[j][i] = 0;
      }
    }
  }
  return payMatrix;
}



// /**
//  * [getDistanceMatrix description]
//  * @param  {[type]} X [description]
//  * @param  {[type]} Y [description]
//  * @return {[type]}   [description]
// */
function getDistanceMatrix(X, Y) {

  var length = X.length;
  var distMatrix = getMatrix(length, length);

  for (var j = 0; j < length; j++) {
    for (var i = 0; i < length; i++) {
      distMatrix[j][i] = round(getDistance(X[j], X[i], Y[j
```

```
            ], Y[i]), 2);
        }
    }
    return distMatrix;
}

function getDistance(X1, X2, Y1, Y2) {
    return Math.sqrt((X1 - X2) * (X1 - X2) +
        (Y1 - Y2) * (Y1 - Y2));
}

// /**
//  * [getMatrix description]
//  * @param {[type]} rowLength    [description]
//  * @param {[type]} colomnLength [description]
//  * @return {[type]}             [description]
// */

function getMatrix(rowLength, colomnLength) {
    var Matrix = new Array(rowLength);

    for (var j = 0; j < rowLength; j++) {
        Matrix[j] = new Array(colomnLength);
    }

    return Matrix;

}
// /**
//   * [round description]
//   * @param {[type]} number   [description]
//   * @param {[type]} accuracy [description]
//   * @return {[type]}         [description]
```

```
//   */
function round(num, accuracy) {
  accuracy = Math.pow(10, accuracy);

  return Math.round(num * accuracy) / accuracy;
}
// /**
//  * [order description]
//  * @param  {[type]} array [description]
//  * @return {[type]}       [description]
//  */
function order(array) {
  var order = [];
  for (var i = 0; i < array.length; i++) {
    order[i] = i;
  }

  return order;
}
//
// /**
//  * [drawCircle description]
//  * @param  {[type]} x     [description]
//  * @param  {[type]} y     [description]
//  * @param  {[type]} r     [description]
//  * @param  {[type]} text  [description]
//  * @param  {[type]} color [description]
//  * @param  {[type]} ctx   [description]
//  * @return {[type]}       [description]
//  */
function drawCircle(x, y, r, text, color, ctx) {

  var text = text || 0;
```

```
var ctx = ctx || this.getCtx();

ctx.beginPath();
r = r || 4;
// /**
//  * [ctx.arc(x, y, r, sAngle, eAngle, counterclockwise) Creates an arc/curve
//  * (used to create circles, or parts of circles)]
//  * @param {[type]} x      [The x-coordinate of the center of the circle]
//  * @param {[type]} y      [The y-coordinate of the center of the circle]
//  * @param {[type]} r      [The radius of the circle]
//  * @param {[type]} sAngle [The starting angle, in radians (0 is at the 3 o'clock position of the arc's circle)]
//  * @param {[type]} eAngle     [The ending angle, in radians]
//  * @param {[type]} counterclockwise [Optional. Specifies whether the drawing should be counterclockwise or clockwise. False is default, and indicates clockwise, while true indicates counter-clockwise.]
//  * @return {[type]}       [description]
//  */
ctx.arc(x, y, r, 0, 2 * Math.PI, false);
ctx.globalAlpha = 1;
ctx.fillStyle = color || 'black';
ctx.fill();
ctx.lineWidth = 1;
ctx.strokeStyle = color || 'black';

if (text) {
```

```
    ctx.font = "10px Georgia";
    ctx.fillText(text, x + 5, y + 5);
  }

  ctx.stroke();
  ctx.closePath();
};

function drawDrone(x, y, r, text, color, ctx) {
  drawCircle(x, y, r, text, color, ctx);
  ctx.strokeStyle = color || 'black';
};

// /**
// * [combineXY description]
// * @param  {[array]} X [description]
// * @param  {[array]} Y [description]
// * @return {[array]}  combineArr [description]
// */
function combineXY(X, Y) {
  var combineArr = [];
  var X = X || this.X;
  var Y = Y || this.Y;

  for (var i = 0; i < X.length; i++) {
    combineArr[i] = [X[i], Y[i]];
  }

  return combineArr;
}

// /**
//  * [inRad description]
```

```
//  * @param  {[num]} angle [description]
//  * @return {[num]}       [angle in the radian]
//  */
function inRad(angle) {
  return angle * Math.PI / 180;
}

// for <<in the perspective>> alternative
function getDroneCoordsForAll(distanceMatrix, X, Y,
   quantityDrone) {
  var diameter = quantityDrone || 1;
  // possible coordinates
  var possCoord = [['X', 'Y', 'j', 'i']];
  var i = 1;
  var x1, x2, y1, y2;
  // get all possible coordinates
  for (var j = 0; j < X.length; j++) {

    for (i; i < X.length; i++) {

      if (distanceMatrix[j][i] > 2.63 && distanceMatrix[j
         ][i] <= 3.01) {

        if (Y[i] == Y[j]) {
          x1 = x2 = round((X[i] + X[j]) * 0.5, 3);
          y1 = Y[i] + 0.5;
          y2 = Y[i] - 0.5;
        }

        if (Y[j] > Y[i] && X[j] < X[i]) {
          x1 = X[j];
          x2 = X[i];
          y1 = Y[j] - 1;
```

```
            y2 = Y[i] + 1;
        }

        if (Y[j] < Y[i] && X[j] > X[i]) {
            x1 = X[i];
            x2 = X[j];
            y1 = Y[i] - 1;
            y2 = Y[j] + 1;
        }

        if (Y[j] < Y[i] && X[j] < X[i]) {
            x1 = X[j];
            x2 = X[i];
            y1 = Y[j] + 1;
            y2 = Y[i] - 1;
        }

        if (Y[j] > Y[i] && X[j] > X[i]) {
            x1 = X[i];
            x2 = X[j];
            y1 = Y[i] + 1;
            y2 = Y[j] - 1;
        }

        possCoord.push([x1, y1, j, i], [x2, y2, j, i]);

    }
  }
  i = j + 2;
}
// get drone coordinates
//
var allDroneCoords = []; // drone coordinates with
```

```
      hexagon center
    var notDrone = false;
    // console.log('possCoord', possCoord);
    //checking if coordinates have been in array already
    for (j = 1; j < possCoord.length; j++) {
      notDrone = X.some(function (Xi, index, array) {
        return Xi == possCoord[j][0] && Y[index] ==
          possCoord[j][1];
      });
      if (notDrone == false) {
        allDroneCoords.push(possCoord[j]);
      }
    }
    // console.log('allDroneCoords', allDroneCoords);

    // checking if coordinates the same
    allDroneCoords.forEach(function (item, i, array) {
      var arr = allDroneCoords.slice(i + 1);
      var count = 0;
      arr.forEach(function (it, index, arr) {
        if (item[0] == it[0] && item[1] == it[1]) {
          allDroneCoords.splice(i + 1 + index - count, 1);
          count++;
        }
      });
    });

    return allDroneCoords;
}
/**
 * HTMLCanvasElement.prototype.drawFullArea
 * @param {object} coordinates
 * @param {number} color
```

```
 */
HTMLCanvasElement.prototype.drawFullArea = function (
    coordinates, color) {
  var ctx = this.getContext('2d');
  // start draw
  ctx.beginPath();
  ctx.globalAlpha = 0.4;
  ctx.fillStyle = color || 'green';
  ctx.strokeStyle = color || 'green';
  ctx.lineWidth = 4;

  ctx.moveTo(coordinates.X[0] * this.SCALE, this.
      getAttribute('height') - coordinates.Y[0] * this.SCALE
      );

  for (var count = 1; count < coordinates.X.length; count
      ++) {
    ctx.lineTo(coordinates.X[count] * this.SCALE, this.
        getAttribute('height') - coordinates.Y[count] * this
        .SCALE);
  }

  ctx.lineTo(coordinates.X[0] * this.SCALE, this.
      getAttribute('height') - coordinates.Y[0] * this.SCALE
      );

  ctx.fill();
  ctx.stroke();
  ctx.closePath();
};

function getMaxOfArray(numArray) {
  if (Math.max.apply(null, numArray) !== Infinity) {
```

```
    return Math.max.apply(null, numArray);
  } else {
    numArray.splice(numArray.indexOf(Infinity), 1);
    return Math.max.apply(null, numArray);
  }
}

function computeMarkWithDrone(situation, droneCoord) {
  var situationWithDrone = setDroneToXY(situation,
    droneCoord);
  return computeMark(situationWithDrone);
}

function computeMark(situation) {
  var distanceMatrix = getDistanceMatrix(situation.X,
    situation.Y);
  var payMatrix = getPayMatrix(distanceMatrix);
  var spaceObj = dijkstra(payMatrix);
  var markAfter = getMaxOfArray(spaceObj.space);
  return markAfter;
}



function createNodes(X, Y, node) {
  var i;

  for (i = 0; i < X.length; i++) {
    createNode(X[i], Y[i], node);
  }

  return 0;
}
```

```
  function createBarrierNode(X, Y) {
    var i;
    var text = "";

    for (i = 0; i < X.length; i++) {
      text = text + "␣("+ X[i] + ",␣" + Y[i] + ")␣—";
    }

    text = text + "␣("+ X[0] + ",␣" + Y[0] + ")";

    var div = document.createElement('div');
    div.innerHTML = text;
    document.body.appendChild(div);

    return 0;
  }

  function createNode(x, y, node) {
    var div = document.createElement('div');

    div.innerHTML = "␣&#92;" + "node[" + node + "]␣at"  + "␣
      ("+ x + ",␣" + y + ")␣" + "{};";

    document.body.appendChild(div);

    return 0;
  }
```

drawGraph.js

```
;( function(){
  var graph = {
      cvs: document.getElementById("graph"),

      getCtx: function() {
```

```
    return this.cvs.getContext("2d");
  },

  getHeight: function() {
    return this.cvs.getAttribute('height');
  },

  getWidth: function() {
    return this.cvs.getAttribute('width');
  }
};

graph.brunches = {
  strategy1: {
    X: [10, 3, 4],
    Y: [10, 2, 5],
    combineXY: combineXY,
    strategies: [2, 3, 4],
    player: false
  },
  strategy2: {
    X: [10, 2, 7],
    Y: [10, 2, 7],
    combineXY: combineXY,
    strategies: [6],
    player: false
  },
  strategy3: {
    X: [5, 2, 4],
    Y: [5, 2, 7],
    combineXY: combineXY,
    strategies: [9],
    player: false
```

```
  },
  strategy4: {
    X: [7, 2, 10],
    Y: [7, 2, 12],
    combineXY: combineXY,
    strategies: [12],
    player: false
  },
  strategy12: {
    X: [7, 2, 10],
    Y: [7, 2, 12],
    combineXY: combineXY,
    strategies: true,
    player: true
  },
  strategy9: {
    X: [10, 14],
    Y: [10, 14],
    combineXY: combineXY,
    strategies: true,
    player: true
  },
  strategy6: {
    X: [10, 10],
    Y: [10, 12],
    combineXY: combineXY,
    strategies: true,
    player: true
  }

};

// drawPluentyOfBeams(graph.getCtx(), 5, graph.coord,
```

```
        false, 'hello');
    graph.draw = drawGraph;

    console.log(graph.draw());
}());
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Cooperative game on a Manet</title>
<link href="assets/css/style.css">
<!-- Latest compiled and minified CSS -->
<link href="/bootstrap/3.3.7/css/bootstrap.min.css">
<!-- jQuery library -->
<script src="jquery.min.js"></script>
<!-- Latest compiled JavaScript -->
<script src="bootstrap.min.js"></script>
</head>
<body>
<div class="container-fluid">
<!-- canvas -->
<div class="row text-center">
<canvas id="hexagonalGrid" width="1400" height="800"></
    canvas>
</div>
<!-- results -->
<div class="row">
<!-- first player -->
<div id="first-player">
<div id="X1 drone"></div>
<div id="Y1 drone"></div>
<!-- second player -->
```

```
<!--<div id="X2 drone"></div>-->
<!--<div id="Y2 drone"></div>-->

<!--<div class="text-info">Graph Tree</div>-->
<!--<canvas id="graph" width="1000" height="500"></canvas>
    -->
<!--</div>-->

<!-- scripts -->
<script src="assets/js/function.js"></script>
<script src="assets/js/draw.js"></script>
<script src="assets/js/compute.js"></script>
<script src="assets/js/show_results.js"></script>
<script src="assets/js/drawGraph.js"></script>
</body>
</html>
```