Санкт-Петербургский государственный университет

Кафедра математической теории игр и статистических решений

Тимонин Николай Олегович

Выпускная квалификационная работа бакалавра

Одна динамическая игра управления агентами в сети

Направление 010400

Прикладная математика и информатика

Научный руководитель,

кандидат физ.-мат. наук,

доцент

Громова Екатерина Викторовна

Санкт-Петербург

2016

Содержание

Введение

В работе введена новая постановка некооперативной многошаговой игры управления агентами в сети. Сетевая структура описывается графом, где вершинами являются агенты, а ребра определены как устойчивые связи между агентами [1]. Выигрыш каждого игрока зависит от диаметра подграфа, определенного на вершинах, принадлежащих игроку. Диаметром графа является длина кратчайшего пути между двумя наиболее удаленными друг от друга вершинами. Диаметр графа используется для оценки максимального времени, требующегося для доставки пакета информации от одного агента к другому.

В данной работе будем предполагать, что каждый каждый игрок имеет в своем распоряжении одного подвижного агента, который может быть расположен в любой допустимой позиции для образования нового подграфа. Целью каждого игрока является уменьшение диаметра результирующего расширенного подграфа.

Актуальность данного подхода определяется практической значимостью рассмотренной задачи в приложениях, связанных с самоорганизующимися одноранговыми мобильными сетями (MANET) [2]. MANET - сеть, без заранее определенной структуры, которая состоит из набора беспроводных мобильных узлов или хостов (роль которых, в данной работе, берут на себя агенты игроков). Правильная организация сети MANET играет ключевую роль в таких задачах как восстановление связи в местах природных катастроф (землетрясений, наводнений и т. п.), координирование действий спасательных отрядов и т. д. Одной из самых важных проблем, связанных с построением данной сети, является разумное использование ограниченных ресурсов, таких как электромагнитный спектр частот радиоволн, время работы аккумуляторов и т. п. Неразумное использование этих ресурсов может привести к ухудшению производительности всей сети в целом. Большая часть исследований, проведенных с применением теории игр в области мобильных сетей, связано с обнаружения вредоносных или не приносящих пользу узлов. В данном же исследовании используется другой подход: рассматривается совместный сценарий поведения узлов для достижения максимальной выгоды.

Основной особенностью данного типа сетей является возможность реорганизовывать структуру сети (топологию, пути передачи данных, распределение спектра и т. д.), основываясь на знаниях предыдущих этапов построения. Таким образом, с течением времени общая производительность сети (скорость доставки пакетов информации, расходы на доставку пакета и т. п.) будет расти.

Все узлы сети имеют некоторую область взаимодействия с другими узлами. Таким образом, последовательность узлов может образовывать

канал (путь) для передачи данных. Большое количество промежуточных узлов в таком пути может отрицательно сказаться на скорости передачи данных. Или, в некоторых случаях, препятствия на пути (озера, обломки и т. п.) могут вызвать его разрыв. Одним из эффективных способов уменьшения нагрузки на канал или его восстановления является использование беспилотных летательных аппаратов - дронов.

Предметом данного исследования является поиск наилучшего места для внедренных мобильных агентов (дронов). Под наилучшим местом понимается такое место, что помещение туда дрона приводит к увеличению производительности сети, а именно уменьшение длины канала передачи данных [3]. В данной задаче жестко ограничено число доступных беспилотных летательных аппаратов. В процессе самоорганизации сеть самостоятельно определяет оптимальное место для помещения в него подвижного узла. В итоге, две части одного маршрута, состоящие из промежуточных неподвижных узлов, будут соединены между собой беспилотником.

Так как сеть MANET предполагает динамическую топологию: в каждый момент времени сеть самостоятельно определят как использовать беспилотник наилучшим образом, было принято решение, в качестве инструмента исследования, применять теорию позиционных игр [4], [5]. Особенностью данного класса игр является учет динамики конфликтно - управляемого процесса. Класс конечношаговых игр с полной информацией является одним из классов позиционных игр. Данный класс игр теоретически позволяет построить граф игры, наглядно демонстрирующий поведение игроков в конкретных ситуациях. Для класса конечношаговых игр вводится понятие абсолютного равновесия по Нэшу, что является усилением понятия равновесия по Нэшу для игр в нормальной форме.

Постановка задачи

В данной выпускной квалификационной работе была предложена новая постановка некооперативной многошаговой игры. Необходимо было решить следующие задачи:

1. исследовать данную игру с применением методов теории позиционных игр;
2. построить дерево решений и сформулировать условие существования абсолютного равновесия по Нэшу;
3. написать программное обеспечение, позволяющее исследовать данную игру;
4. протестировать программный продукт на примере игры двух лиц.

Обзор литературы

При написани работы была использована литература:

1. Новиков Д. А. Игры и сети. Математическая теория игр и её приложения.

   В данной статье описаны общие положения и структура современных направлений в играх на сетях. Введена система классификации игр на сетях с точки зрения теории игр и теории графов. Приведены общие положения теории сетевых игр. Изложение материала построено таким образом, чтобы обеспечить целостное представление об играх на сетях. При написании работы использовались материалы из раздела 2 (Игры на сетях) и 3 (Задача управления).

2. Петросян Л. А., Седаков А. А. Многошаговые сетевые игры с полной информацией.

   В статье рассматриваются многошаговые сетевые игры с полной информацией, в которых в каждый момент игры задается сетевая структура, соединяющая игроков. Демонстрируется метод нахождения оптимального поведения игроков.

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

   В этой книге описаны общие положения математической теории игр как составной части исследования операций. Изложение материала построено таким образом, что понятие модели конфликта (игры) развивается от простой (матричные игры) до наиболее сложной (дифференциальные игры). Во всех главах имеется большое количество примеров, иллюстрирующих основные положения теории. В частности, подробно описана теория многошаговых игр с полной информацией. При написании работы использовались материалы из гл. 4 (Позиционные игры).

4. Wilson R. J. Introduction to Graph Theory.

   Данная книга является вводным курсом в теорию графов, но вместе с тем затрагивает ряд интересных и сложных задач. Полезна для решения прикладных задач теории графов. При написании работы использовалась для описания взаимодействия между агентами игрока с точки зрения теории графов.

5. Yu F. R. Cognitive Radio Mobile Ad Hoc Networks.

   В данной книге подробно описаны технологии MANET сетей. Описаны общие принципы построения сетей и дальнейшее изменения их структуры для увеличения производительности сетей. При написании работы книга использовалась как дополнение к описанной выше литературе

при построении общей архитектуры сети, описания поведения подвижных агентов и методов взаимодействия агентов, как узлов сети.

6. Zhu Han and Dusit Niyato and Walid Saad and Tamer Başar and Are Hjørungnes. Game Theory in Wireless and Communication Networks.

   Данная книга позволяет получить общее представление о теории игр, её использование в построении безпроводных коммуникаций и решение проблем, связанных с построением сетей. В частности, описаны 3G сети, LAN, сенсорные и самоорганизующиеся сети. Также материал книги покрывает техническое моделирование, построение архитектуры и анализ сетей с применением теории игр.

7. Dijkstra E. W. A note on two problems in connexion with graphs.

   В данной статье описан алгоритм, позволяющий находить кратчайшие пути от одной вершины графа, до всех остальных.

Глава 1. Основные понятия

1.1 Многошаговая игра с полной информацией

Многошаговая игра с полной информацией определяется на древовидном конечном графе $G = (X, F)$, где X - некоторое конечное множество, а F - отображение из X в X - некоторое правило, которое каждому элементу $x \in X$ ставит в соответствие некоторое множество $F_x \subset X$. В частности, возможна ситуация когда $F_x = \emptyset$.

Элементы множества X будем изображать точками на плоскости. Пары точек x и y, для которых выполняется $y \in F_x$, будем соединять непрерывной линией со стрелкой, направленной от x к y. В дальнейшем, каждый элемент из X будем называть вершиной (узлом) графа, а пару элементов (x, y), для которой $y \in F_x$ - дугой графа. Вершины x и y для дуги $p = (x, y)$ являются граничными вершинами, где x - начало, а y - конец дуги. Различные дуги, имеющие общую граничную точку, будем называть смежными. Все множество дуг в графе будем обозначать P.

Путем в графе $G = (X, F)$ будем называть такую последовательность $p = (p_1, p_2, \dots, p_k, \dots)$ дуг, в которой конец каждой предыдущей дуги совпадает с началом следующей. Длина пути $p = (p_1, \dots, p_s)$ - число $l(p) = s$ дуг последовательности.

Под ребром графа $G = (X, F)$ будем понимать множество из двух элементов $x, y \in X$, для которых выполняется $(x, y) \in P$, или $(y, x) \in P$. Заметим, что в ребре ориентация роли не играет, в отличии от дуги. В дальнейшем ребра будем обозначать p, q, а все множество ребер - P. Цепью будем называть такую последовательность ребер $(p_1, p_2, \dots)$, в которой у каждого ребра p_k одна из граничных вершин является также граничной для p_{k-1}, а другая - граничной для p_{k+1}. Циклом графа $G = (X, F)$ будем называть конечную цепь, начинающуюся и оканчивающуюся в одной и той же вершине.

Древовидный граф - конечный связный граф без циклов, имеющий не менее двух вершин, в котором существует единственная вершина x_0, такая, что $F_{x_0} = X$. Данную вершину x_0 будем называть начальной вершиной графа G.

Подграфом G_z древовидного графа $G = (X, F)$, будем называть граф вида (X_z, F_z), где $z \in X$ и $X_z = F_z$, а $F_z x = F_x \cap X_z$. Отображение F_z является сужением отображения F на множество X_z. Данный подграф будем обозначать $G_z = (X_z, F_z)$.

Далее перейдем к определению многошаговой игры на древовидном конечном графе. Разобьем множество вершин X на $n + 1$ подмножество $X_1, \dots, X_{n+1}$, такое что

$$\bigcup_{i=1}^{n+1} X_i = X, X_k \cap X_l = \emptyset, k \neq l,$$

где $F_x = \emptyset$ для $x \in X_{n+1}$. Множество $X_i, i = 1, \ldots, n$ является множеством очередности игрока i, а множество X_{n+1} - множеством окончательных позиций. На множестве окончательных позиций X_{n+1} определены n вещественных функций $H_1(x), \ldots, H_n(x), x \in X_{n+1}$, которые будем называть функциями выигрышей игроков.

Далее опишем процесс игры. Имеется некоторое множество игроков N, пронумерованных натуральными числами. Если $x_0 \in X_{i_1}$, тогда в вершине x_0 «ходит» игрок i_1 и выбирает вершину $x_1 \in F_{x_0}$. Далее, если $x_1 \in X_{i_2}$, то в вершине X_1 «ходит» игрок i_2 и выбирает следующую вершину $x_2 \in F_{x_1}$. И так далее, если на k-ом шаге $x_{k-1} \in X_{i_k}$, то «ходит» игрок i_k, выбирая следующую вершину из множества $F_{x_{k-1}}$. Игра заканчивается при достижении окончательных вершины $x_l \in X_{n+1}$, т. е. такой, для которой $F_{x_l} = \emptyset$.

Последовательный выбор вершин однозначно реализует последовательность $x_0, \ldots, x_k, \ldots, x_l$, определяющую путь в древовидном графе G, с началом в x_0 и концом в одной из окончательных позиций игры. Этот путь будем называть партией.

Так как граф G древовидный любая партия заканчивается в некоторой окончательной позиции x_l и, наоборот, любая окончательная позиция x_l однозначно определяет некоторую партию. В окончательной позиции x_l каждый из игроков получает выигрыш. Игрок i при совершении выбора в вершине $x \in X_i$ знает позицию x, а следовательно, из-за древовидности графа G может восстановить и все предыдущие позиции. В этом случае говорят, что игроки имеют полную информацию.

Стратегией игрока i будем называть однозначное отображение u_i, которое ставит в соответствие каждой вершине $x \in X_i$ некоторую вершину $y \in F_x$. Таким образом, для любой позиции x из множества его очередности X_i, стратегия игрока i предписывает ему однозначный выбор следующей позиции.

Ситуацией в игре будем называть упорядоченный набор

$$u = (u_1, u_2, \ldots, u_n), u_i \in U_i,$$

где U_i - множество всевозможных стратегий игрока i. Множеством ситуаций в игре будем называть декартово произведение $U = \prod_{i=1}^{n} U_i$.

Заметим, что каждая ситуация $u = (u_1, u_2, \ldots, u_n)$ в игре однозначно соответствует партии игры $x_0, x_1, \ldots, x_l$ и, как следствие, однозначно определяет выигрыш каждого игрока

$$K_i(u_1, u_2, \ldots, u_n) = H_i(x_l), i = 1, \ldots, n.$$

Таким образом, значение выигрыша каждого игрока K_i в ситуации u равно значению выигрыша H_i в окончательной позиции партии $x_0, x_1, \ldots, x_l$, которая соответствует ситуации $u = (u_1, u_2, \ldots, u_n)$. В итоге мы получили некоторую игру в нормальной форме

$$\Gamma = (N, \{U_i\}_{i \in N}, \{K_i\}_{i \in N}),$$

где N - множество игроков, U_i - множество стратегий игрока i, K_i - функция выигрыша игрока i.

Ситуацию $u^* = (u_1^*, u_2^*, \ldots, u_n^*)$ будем называть равновесием по Нэшу, если выполняется неравенство:

$$K_i(u_1^*, \ldots, u_{i-1}^*, u_i^*, u_{i+1}^*, \ldots, u_n^*) \geq K_i(u_1^*, \ldots, u_{i-1}^*, u_i, u_{i+1}^*, \ldots, u_n^*),$$

где $u_i \in U_i, i \in N$

Подыгрой Γ_z будем называть игру определенную на подграфе $G_z = (X_z, F), z \in X$. Ей соответствует подыгра в нормальной форме

$$\Gamma_z = (N, \{U_i^z\}_{i \in N}, \{K_i^z\}_{i \in N}),$$

где $u_i \in U_i, i \in N$, а функции выигрыша $K_i^z, i = 1, \ldots, n$ определены на декартовом произведении $U^z = \prod_{i=1}^{n} U_i^z$.

Ситуацию равновесия по Нэшу основной игры $u^* = (u_1^*, u_2^*, \ldots, u_n^*)$ будем называть абсолютным равновесием по Нэшу в игре Γ, если для любого $z \in X$ ситуация $(u^*)^z = ((u_1^*)^z, (u_2^*)^z, \ldots, (u_n^*)^z)$, где $(u_i^*)^z$ - сужение стратегии u_i^* на подигру Γ_z, является ситуацией равновесия по Нэшу в подигре Γ_z.

1.2 Алгоритм Дейкстры

Данный алгоритм позволяет найти кратчайшие пути от некоторой вершины v графа $G = (V, E)$ до всех остальных вершин этого графа [6].

Каждой вершине графа сопоставляется метка, означающая минимальное расстояние от v до этой вершины. Работа алгоритма выполняется пошагово: на каждом шаге берется одна вершина и выполняется попытка уменьшения значения метки. Алгоритм заканчивает свою работу после перебора всех вершин.

1. Начальное состояние. Значение метки самой вершины v устанавливается в 0, а значение меток других вершин - в бесконечность (что, по сути, отражает неизвестность расстояния от v до других вершин). Все вершины графа помечаются как нетронутые.

2. Шаг алгоритма. Если все вершины были взяты, алгоритм завершается. В противном случае, из множества не взятых вершин выбирается

вершина s, у которой минимальное значение метки. Рассматриваются всевозможные маршруты, для которых s является предпоследним пунктом. Вершины, к которым ведут ребра из s, будем называть соседями этой вершины. Для каждого не взятого соседа s рассмотрим новую длину пути, которая равна сумме значений текущей ветки s и длины ребра, соединяющего s с этим соседом. Если полученное значение длины меньше значения метки соседа, тогда заменим значение метки полученным значением длины. После рассмотрения всех соседей, пометим вершину s как взятую. Повторим шаг алгоритма.

Недостаток данного алгоритма заключается том, что он будет некорректно работать если граф имеет дуги отрицательного веса.

Глава 2. Постановка игры управления агентами в сети

2.1 Общая сетевая структура

Задано множество игроков $N = \{1, \ldots, n\}$. Каждый игрок i имеет не пустое множество M_i агентов, расположенных в подпространстве

$$W = \{1, \ldots, W_x\} \times \{1, \ldots, W_y\} \subset R^2,$$

т. е. в узлах целочисленной решетки, где $W_x, W_y \in N$. Возможно расположение агентов разных игроков в одном узле решетки. Позиция каждого агента описывается положительными координатами в ДПСК. Далее агентов будем обозначать точками на плоскости. Будем говорить, что между агентами v_p^i и $v_s^i, s \neq p$, игрока i установлена устойчивая связь (v_p^i, v_s^i), если они находятся в соседних узлах целочисленной решетки. Связи будем обозначать на плоскости непрерывным отрезком, соединяющим точки (см. рис. 1).

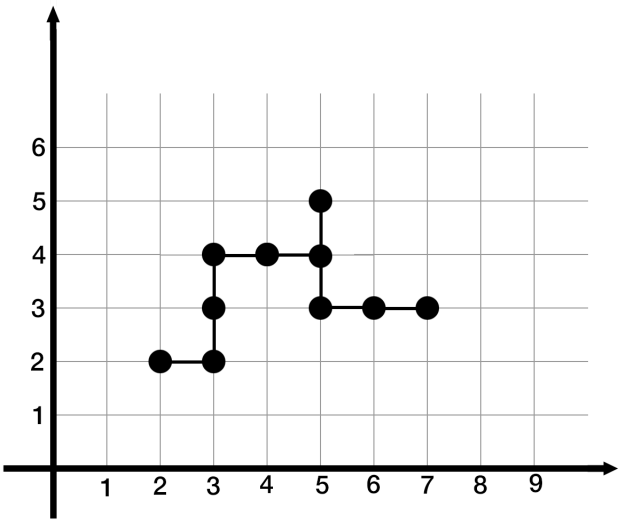

Рис. 1: Агенты и связи между ними

Таким образом множество агентов (точек) $v_p^i \in M_i, i = 1, \ldots, n$, и связей (отрезков) $e = (v_p^i, v_s^i), v_p^i \in M_i, v_s^i \in M_i, s \neq p, i = 1, \ldots, n$, определяет подграф $R_i = (V_i, E_i) \in R = (V, E)$, где V — не пустое конечное множество элементов, называемых вершинами (или точками), E — конечное множество неупорядоченных пар элементов из V, называемых ребрами (или линиями) [7]. Подграфом R_i графа R, будем называть граф, все вершины которого принадлежат V, а все ребра принадлежат E.

Заметим, 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

следующим образом

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

В дальнейшем, будем считать, что подграф R_i связный. Это гарантирует существование пути между любыми двумя его вершинами. Заметим отсутствие связей между элементами разных подграфов, т. е. неподвижные агенты одного игрока не взаимодействуют с неподвижными агентами других игроков в процессе передачи информации.

Путем в подграфе R_i будем называть такую последовательность $e = (e_1, \ldots, e_k, \ldots)$ ребер, что конец каждого предыдущего ребра совпадает с началом следующего. Длина пути $e = (e_1, \ldots, e_k)$ — число $d(e) = k$ ребер последовательности. Очевидно, что длина каждого такого ребра равна единице.

Диаметром подграфа $D(R_i)$ будем называть число ребер в кратчайшем простом незамкнутом пути между двумя наиболее удаленными друг от друга вершинами (см. рис. 2).

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

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

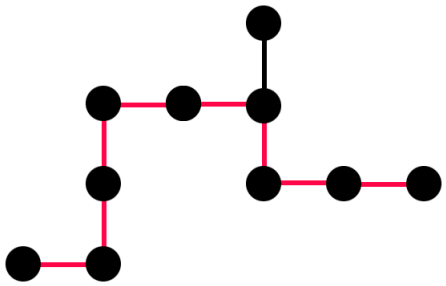

Рис. 2: Диаметр подграфа

Так как каждый подграф R_i связный, то его диаметр может быть ограничен следующим образом:

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

2.2 Подвижный агент: дрон

В распоряжении каждого игрока имеется один подвижный агент. Подвижным агентом $q^i, i \in N$, будем называть такого агента, положение которого на целочисленной решетке может изменяться в процессе игры, в то время как другие агенты имеют фиксированные положения (см. рис. 3). Связь подвижного агента с другим агентом определяется аналогично связи между неподвижными агентами. Подвижный агент может устанавливать

связи с любым подвижным агентом $q^i, i \in N$, и с любыми неподвижными агентами $v_p^i, p \in M_i, i \in N$.

Будем считать, что в начальный момент времени все дроны расположены в такой начальной позиции q^o, что длина пути от них до любого подвижного агента больше единицы:

$$d(q^o, v_p^i) > 1, p \in M_i, i \in N.$$

Это условие гарантирует, что ни один дрон не может установить связь с ни с одним неподвижным агентом в начальной стадии игры.

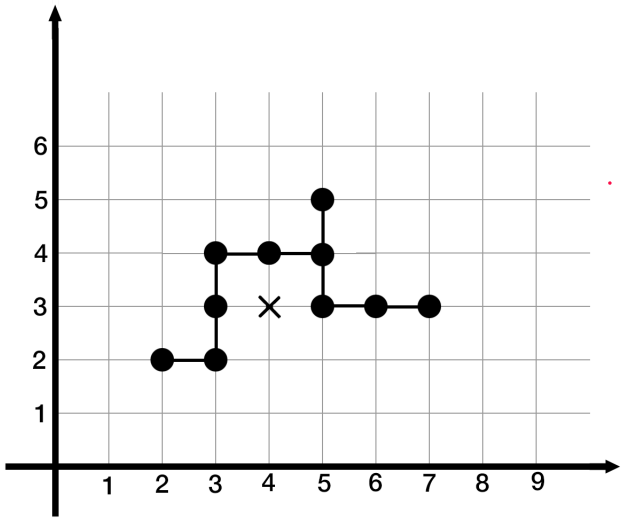

Рис. 3: Подвижный агент

Подграф R_i^* будем называть расширенными подграфом, где множество вершин:

$$V(R_i^*) = M_i \cup \bigcup_{i=1}^{n} q^i, i = 1, \ldots, n,$$

а множество ребер определяется связями на множестве $V(R_i^*)$ (см. рис. 4). Заметим возможность установления связи между расширенными подграфами, которая зависит от расположения подвижных агентов.

2.3 Альтернативы и выигрыши

Каждый из игроков стремится так расположить своего подвижного агента, чтобы минимизировать диаметр на своем результирующем расширенном подграфе.

Игроки ходят по очереди. На каждом шаге игрок i выбирает одни элемент из своего множества альтернатив W_i^*. Под множеством альтернатив W_i^* игрока i будем понимать все те возможные положения на целочисленной решетке для установления в них подвижного агента, которые дадут уменьшение диаметра расширенного подграфа игрока i по сравнению с текущим диаметром (или начальным диаметром, если это первый шаг). Это

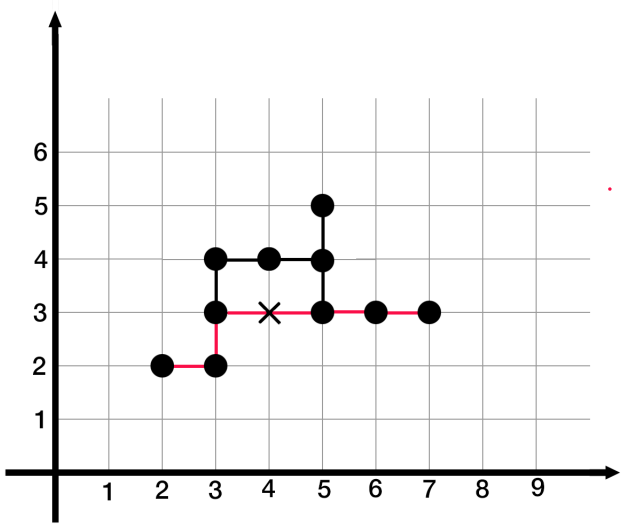

Рис. 4: Расширенный подграф

означает что на каждом шаге игрок рассматривает только те позиции для дрона, которые дадут ему уменьшение диаметра в сравнении с текущим его значением. Заметим, что множество альтернатив игрока i на каждом шаге зависит от его предыдущих ходов, и от предыдущих ходов других игроков.

Таким образом, каждый игрок решает задачу следующего вида:

$$\bar{q}_i = \underset{q_i \in W_i^*}{argmin} D(R_i)$$

Будем считать, что игрок «помнит» свои предыдущие ходы и «знает» предыдущие ходы других игроков. Следовательно, данный процесс может быть сформулирован как многошаговая игра с полной информацией.

Игра заканчивается когда ни один из игроков не может далее уменьшить диаметра своего расширенного подграфа. Функция выигрыша i-го игрока задается следующим образом:

$$H_i = d(R_i) - d(R_i^*) \geq 0.$$

2.4 Пример игры

Проиллюстрируем данную игру на примере.

Игра Г происходит на сетевой структуре, изображенной на рис. 5. Множество N состоит из двух игроков: $N = \{1, 2\}$. Агентов первого игрока будем изображать в виде заполненных кружков, а второго игрока — в виде квадратиков. Подвижного агента первого игрока будем изображать в виде крестика, а второго игрока — в виде кружка. Положение каждого агента задается двумя координатами в декартовой прямоугольной системе координат (ДПСК). Связи между агентами на рис. 5 изображены в виде линий, которые соединяют соответствующих агентов. Диаметр подграфа первого игрока $d(R_1) = 16$, а второго — $d(R_2) = 16$.

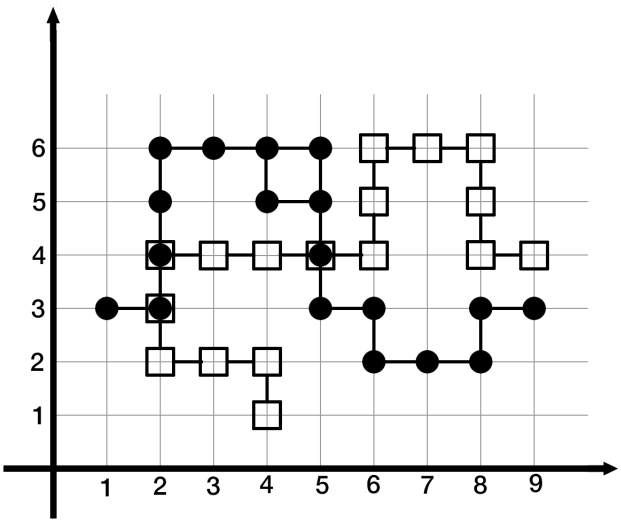

Рис. 5: Сетевая структура игры

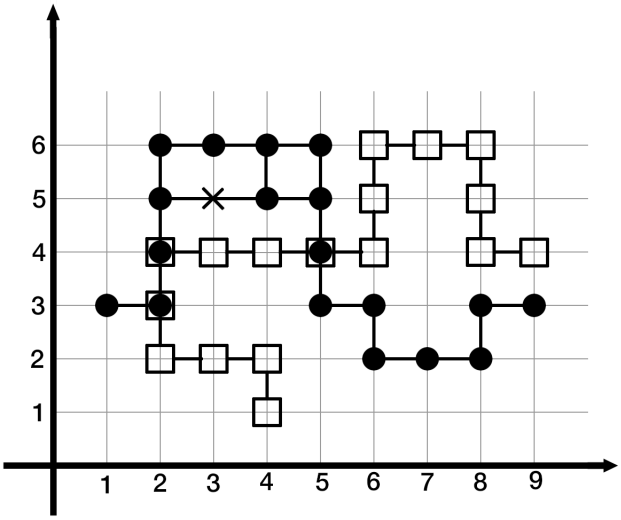

Рис. 6: Сетевая структура игры после хода первого игрока

Первым «ходит» игрок 1. Ему выгодно перемещение подвижного агента в одну из позиций: $(3,5),(7,3)$, так как в каждой из этих позиций он достигнет уменьшения диаметра своего подграфа. Пусть он перемещает своего подвижного агента в позицию $(3,5)$ (см. рис. 6). В результате диаметр образовавшегося расширенного подграфа $d(R_1^*) = 14$, что дает ему уменьшение диаметра на две единицы.

Далее «ходит» игрок 2. Одними из возможжных позиций для перемещения подвижного агента являются $(4,3),(7,4),(7,5)$. Пусть он перемещает своего подвижного агента в позицию $(4,3)$ (см. рис. 7). Тогда диаметр образовавшегося расширенного подграфа $d(R_2^*) = 12$.

Далее снова делает ход первый игрок. С учетом позиции подвижного агента второго игрока, он может расположить своего агента в одной из позиций: $(3,3),(7,3)$. Пусть он перемещает своего агента в позицию $(3,3)$, в результате чего диаметр нового результирующего подграфа $d(R_1^*) = 10$

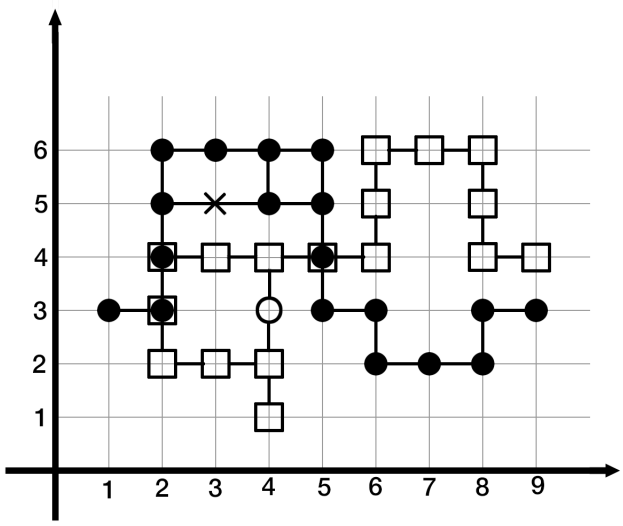

Рис. 7: Сетевая структура игры после хода второго игрока

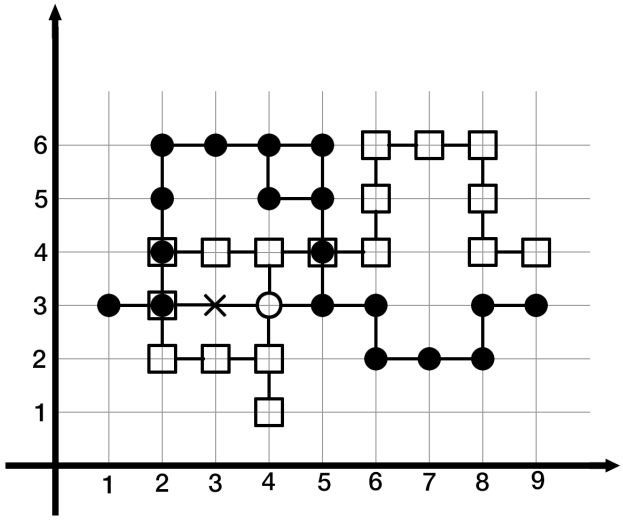

Рис. 8: Конечная сетевая структура игры

(см. рис. 8). Далее ни один из игроков не может уменьшить диаметр своего расширенного подграфа, и игра заканчивается с выигрышами $H_1 = 6$ и $H_2 = 4$.

Данная проблема может быть сформулирована как многошаговая игра с полной информацией [4].

Граф G, частично иллюстрирующий дерево решений многошаговой игры, изображен на рис. 9, где элементы множества очередности первого игрока X_1 изображены в виде кружков, а элементы множества очередности второго игрока X_2 — в виде квадратиков. Позиции, входящие в множества X_1, X_2, пронумерованы двойными индексами, а дуги — одним индексом. Выигрыши игроков записаны в окончательных позициях.

Первый игрок на первом шаге выбирает из двух альтернатив. Двойными индексами обозначены позиции дрона в ДПСК после хода игрока. В вершине $(3, 5)$ второй игрок выбирает из трех альтернатив и т. д.

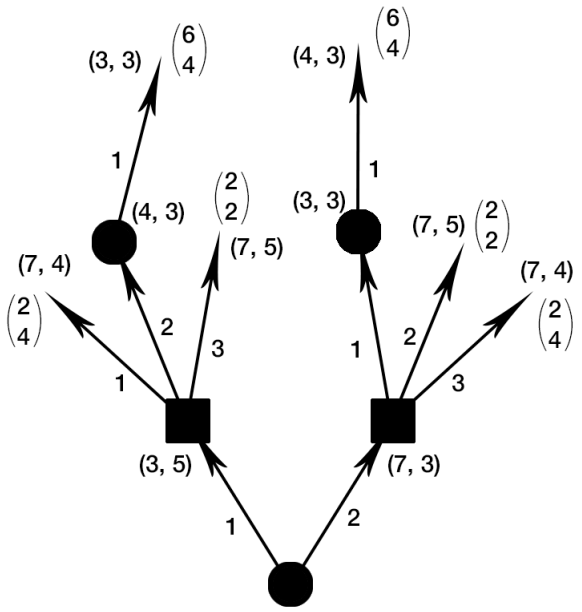

Рис. 9: Игра с полной информацией на древовидном графе G

На данном дереве можем видеть две ситуации абсолютного равновесия по Нэшу [4] (u_1^1, u_2^1) и (u_1^2, u_2^2) при условии, что второй игрок выберет вершину $(4, 3)$ вместо $(7, 4)$ для левой ветки дерева решений ($(3, 3)$ вместо $(7, 4)$ для правой), не смотря на то, что в обеих вершинах он получает одинаковый выигрыш. Стратегии $u_1^1 = (1, 1)$, $u_2^1 = (2)$, $u_1^2 = (2, 1)$, $u_2^2 = (1)$ указывают в каждой вершине номер дуги, по которой следует двигаться дальше.

2.5 Условие существования равновесия по Нэшу

Ситуация абсолютного равновесия по Нэшу существует в любой многошаговой игре с полной информацией на конечном древовидном графе $G = (X, F)$ [4]. Подчеркнем различие графов G и $R = (V, E)$: граф R описывает сетевую структуру игры, а граф G - дерево дерево игры.

Покажем, что в данной игре ситуация равновесия по Нэшу существует не всегда. Конечность древовидного графа означает, что каждый из игроков сделает конечное число шагов в игре. Таким образом, для гарантирования существования абсолютного равновесия по Нэшу в игре с полной информацией, мы должны гарантировать её конечность.

Рассмотрим пример игры с отсутствием равновесия по Нэшу. Сетевая структура изображена на рис. 10. Множество игроков $N = \{1, 2\}$. Неподвижные агенты первого игрока изображены в виде виде заполненных кружков, а второго игрока — в виде квадратиков. Диаметр подграфа первого игрока $d(R_1) = 10$ с началом в точке $(1, 4)$ и концом в $(7, 4)$, а второго — $d(R_2) = 15$ с началом в (5,1) и концом в (7,4). Связи между агентами, находящимися в одинаковых координатах, отсутствуют.

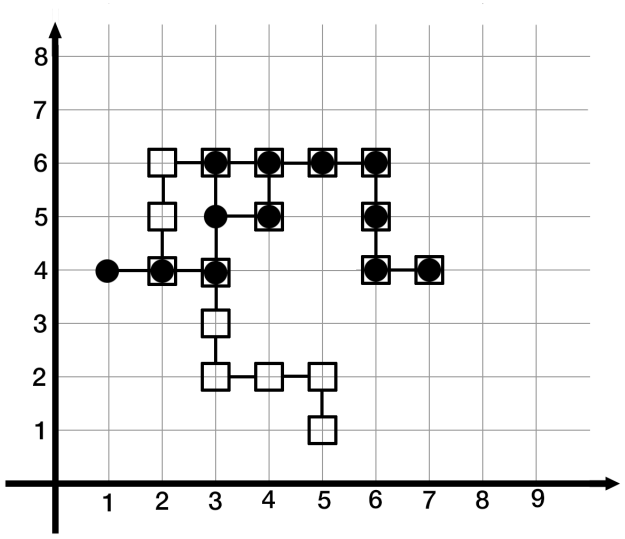

Рис. 10: Сетевая структура игры с отсутствием равновесия по Нэшу

1. Шаг № 1. Делает ход первый игрок. Единственная позиция для подвижного агента доступная ему - $(5, 5)$. Устанавливая дрон в эту позицию, он добивается уменьшения диаметра своего расширенного подграфа на две единицы. Таким образом $d(R_1^*) = 8$. Его ход никак не повлиял на диаметр другого игрока: $d(R_2^*) = 15$.

2. Шаг № 2. Далее ходит второй игрок. Он может установить своего активного агента в позицию $(4, 4)$. В результате этого хода $d(R_2^*) = 11$ и $d(R_1^*) = 8$. Таким образом, он уменьшил свой диаметр на 4 единицы.

3. Шаг № 3. Снова ход первого игрока. Он перемещает своего агента в позицию $(5, 4)$. Этот подвижный агент устанавливает связь с активным агентом другого игрока, который на данном шаге в позиции $(4, 4)$, и с неподвижным агентом в позиции $(6, 4)$. Таким образом, используя в своих целях агента второго игрока, первый игрок получает уменьшение диаметра еще на две единицы: $d(R_1^*) = 6$, $d(R_2^*) = 9$. Заметим также, что результатом его хода стало также уменьшение диаметра второго игрока на две единицы.

4. Шаг № 4. Желая использовать дрон первого игрока, второй игрок перемещает своего агента в позицию $(5, 3)$ и получает $d(R_2^*) = 5$. Но в результате его хода, перестает существовать построенный путь первого игрока, и его диаметр устанавливается в первоначальное значение: $d(R_1^*) = 10$.

5. Шаг № 5. Первый игрок снова поместит дрон в позицию $(5, 5)$, разрушив построенный путь второго игрока и получив $d(R_1^*) = 8$. Для второго игрока $d(R_2) = 15$.

6. Шаг № 6. Второй игрок переместит агента в позицию $(4, 4)$. И так далее.

Таким образом дрон первого игрока будет постоянно перемещаться между позициями $(5,5)$ и $(5,4)$, а второго - $(4,4)$ и $(5,3)$. Следовательно, данная игра не конечношаговая и в ней отсутствует абсолютное равновесие по Нэшу.

Сформулируем условие существования абсолютного равновесия по Нэшу: каждый игрок перемещает своего дрона так, чтобы диаметр остальных игроков не увеличился. Будем называть это условие условием «благожелательности» игроков. Очевидно, что после введения данного условия количества альтернатив для каждого игрока может уменьшиться.

Рассмотрим теперь эту же игру, но теперь на каждого игрока наложим условие «благожелательности». Пусть, первый шаг делает первый игрок. Он помещает своего агента в позицию $(5,5)$. На втором шаге второй игрок устанавливает агента в позицию $(4,4)$. На третьем шаге первый игрок, желая использовать дрон второго игрока, перемещает активного агента в позицию $(5,4)$ (см. рис. 11). На этом игра заканчивается: второй игрок не может переместить беспилотник в позицию $(5,3)$, т. к. это приведет к увеличению текущего диаметра первого игрока. Очевидно, что в данном случае единственное решение и будет равновесным по Нэшу.

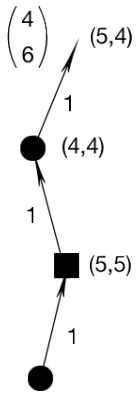

Рис. 11: Дерево игры при условии, что первым «ходит» первый игрок

Рассмотрим теперь случай, когда первый шаг делает второй игрок. Он устанавливает подвижного агента в позицию $(4,4)$. После его хода, первый игрок имеет две альтернативы: позиция $(5,4)$ и $(5,5)$. Помещая дрон в любую из этих позиций, первый игрок заканчивает игру (см. рис. 12). В данном случае стратегии $u_1 = (1)$, $u_2 = (1)$ приведут к абсолютному равновесию по Нэшу с выигрышам для первого игрока 4, для второго - 6.

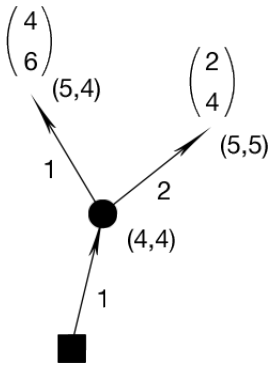

Рис. 12: Дерево игры при условии, что первым «ходит» второй игрок

Глава 3. Описание программного обеспечения

В рамках данной работы автором было реализовано программное обеспечение, позволяющее исследовать игру, описанную в главе 2.

3.1 Постановка задачи для программного обеспечения

Программный продукт должен удовлетворять следующим требованиям:

1. Должен быть реализован алгоритм, позволяющий по имеющимся входным данным о расположении агентов построить все сценарии развития игры.
2. Архитектура программы должна позволять относительно легкую масштабируемость, изменение деталей постановки игры и количества игроков.

3.2 Обоснование подхода к решению задачи

Наглядно продемонстрировать все сценарии развития игры можно с помощью дерева решений. Данное дерево состоит из единственного узла - корня и присоединенных к нему поддеревьев, которые также являются деревьями. Таким образом это дерево может быть определено через само себя. Поэтому было принято решение использовать рекурсию для построения дерева решений.

Рекурсивная процедура работает следующим образом: «рисует» одну линию (до разветвления), а затем вызывает сама себя для «рисования» поддеревьев. Эти поддеревья отличаются своими корнями и количеством содержащихся в них разветвлений. В данном случае корень соответствует одной вершине из множества очередности игрока, а разветвления - множеству альтернатив этого же игрока.

Параметр	Значение
Step	Номер шага
P	Номер игрока
Pos	Новая позиция подвижного агента
Dpr	Диаметр на предыдущем шаге
CurrPayoff	Выигрыш игрока от этого шага
P1 GlobPayoff	Текущий общий выигрыш первого игрока
P2 GlobPayoff	Текущий общий выигрыш второго игрока

Таблица 1: Параметры

Листьями мы будем называть узлы, не содержащие поддеревьев. Множество листьев соответствует множеству окончательных позиций, т. е. таких позиций, в которых у игроков нет альтернатив для выбора положения дрона. Глубина рекурсии соответствует общему количеству шагов, сделанных игроками.

Вывод программы состоит из параметров, приведенных в таблице 1.

3.3 Описание архитектуры программы

Данный программный продукт написан на объектно ориентированном языке высокого уровня Objective-C с использованием фреймворка Foundation. В качестве среды для разработки использовался X-Code.

Фреймворк Foundation определяет базовый слой Objective-C классов, представляет множество парадигм, которые определяют функциональность не охватываемую языком Objective-C. Фреймворк включает в себя корневой класс (NSObject), классы представляющие базовые типы данных, такие как строки (NSString) и числа (NSNumber), классы коллекции для хранения других объектов (NSArray, NSDictionary, NSSet) и т. д.

В качестве библиотеки для описания графов использовалась библиотека с открытым исходным кодом: PESGraph. Она содержит классы для описания графа (PESGraph), ребер (PESGraphEdge), вершин (PESGraphNode), маршрута (PESGraphRoute) и шага в маршруте между двумя вершинами (PESGraphRouteStep).

Были написаны следующие основные классы: NTPosition, NTSpace, NTPlayer. Класс NTPosition описывает положение в ДПСК, а также дает возможность проверки узлов на соседство. Класс NTPlayer описывает игрока. NTSpace описывает подпространство, в котором происходит игра. Этот класс является синглтоном.

Для добавления методов к уже существующим классам библиотеки PESGraph использовались категории. Использование данных технологий позволяет не вносить изменения в исходные файлы классов, что облегчает разработку и возможность обновления версии библиотеки без влияния на добавленные методы. Добавленные в категорию методы доступны всем классам, унаследованным от существующего. В программе была добавлена категория PESGraph+Diameter с методом diameter, который возвращает объект типа PESGraphRoute, описывающий диаметр для текущего графа.

Для добавления свойств к уже существующим классам использовались расширения классов. В программе создано расширение класса PESGraphNode, добавляющее свойство типа NTPosition. Название расширения PESGraphNode_Position.

Глава 4. Тестирование программы на примере игры двух лиц

4.1 Постановка игры

Игра Г происходит на сетевой структуре, изображенной на рис. 13. Имеется два игрока: $N = \{1, 2\}$. Вершины подграфа первого игрока будем изображать в виде заполненных кружков, а второго игрока — в виде квадратиков. Заметим, что в позиции $(3, 5)$ находятся одновременно два неподвижных агента разных игроков. Однако связь между ними отсутствует.

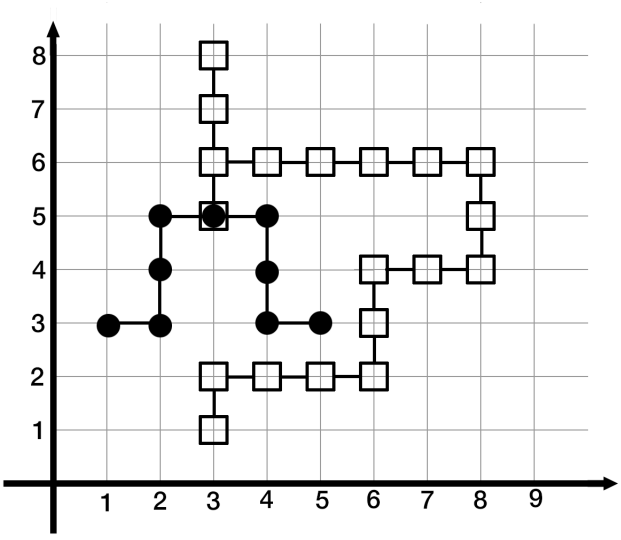

Рис. 13: Сетевая структура игры

Диаметр подграфа первого игрока $d(R_1) = 8$ с началом в точке $(2, 3)$ и концом в $(5, 3)$. Диаметр второго игрока $d(R_2) = 17$ с началом в точке $(3, 8)$ и концом в $(3, 1)$. Целью каждого из игроков на протяжении всей игры является минимизация своего диаметра.

4.2 Результат при условии, что первым «ходит» первый игрок

Результат работы программы представлен на рис. 14. Данный результат означает, что на первом шаге первый игрок может установить своего агента в позицию $(3, 3)$, что дает ему уменьшение его диаметра на 4 единицы. Видим, что данный ход никак не влияет на диаметр второго игрока (P2 GlobalPayoff: 0). Далее «ходит» второй игрок. После хода первого игрока, для него имеются три альтернативы для установления подвижного агента: $(3, 4), (6, 5), (7, 5)$. При установлении подвижного агента в одну из этих позиций он получит выигрыш 10, 4, и 2 соответственно. После любого

```
P1 startDiameterRoute: 8
P2 startDiameterRoute: 17
Step:1  P1  Pos:(3,3)  Dpr: 8  Dn: 4  CurrPayoff: 4    P1 GlobPayoff: 4    P2 GlobPayoff: 0
Step:2  P2  Pos:(3,4)  Dpr:17  Dn: 7  CurrPayoff:10    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:2  P2  Pos:(6,5)  Dpr:17  Dn:13  CurrPayoff: 4    P1 GlobPayoff: 4    P2 GlobPayoff: 4
Step:2  P2  Pos:(7,5)  Dpr:17  Dn:15  CurrPayoff: 2    P1 GlobPayoff: 4    P2 GlobPayoff: 2
Step:1  P1  Pos:(3,4)  Dpr: 8  Dn: 6  CurrPayoff: 2    P1 GlobPayoff: 2    P2 GlobPayoff: 0
Step:2  P2  Pos:(3,3)  Dpr:17  Dn: 7  CurrPayoff:10    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:2  P2  Pos:(6,5)  Dpr:17  Dn:13  CurrPayoff: 4    P1 GlobPayoff: 2    P2 GlobPayoff: 4
Step:3  P1  Pos:(3,3)  Dpr: 6  Dn: 4  CurrPayoff: 2    P1 GlobPayoff: 4    P2 GlobPayoff: 4
Step:4  P2  Pos:(3,4)  Dpr:13  Dn: 7  CurrPayoff: 6    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:2  P2  Pos:(7,5)  Dpr:17  Dn:15  CurrPayoff: 2    P1 GlobPayoff: 2    P2 GlobPayoff: 2
Step:3  P1  Pos:(3,3)  Dpr: 6  Dn: 4  CurrPayoff: 2    P1 GlobPayoff: 4    P2 GlobPayoff: 2
Step:4  P2  Pos:(3,4)  Dpr:15  Dn: 7  CurrPayoff: 8    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:4  P2  Pos:(6,5)  Dpr:15  Dn:13  CurrPayoff: 2    P1 GlobPayoff: 4    P2 GlobPayoff: 4
```

Рис. 14: Вывод программы при условии, что первым «ходит» первый игрок

из этих ходов игра заканчивается, так как нет больше альтернатив ни для одного игрока.

Рассмотрим другую ветку дерева игры. Первый игрок на первом шаге может установить подвижного агента позицию $(3, 4)$. На втором шаге второй игрок имеет в качестве альтернатив позиции $(3, 3), (6, 5), (7, 5)$. Например, если второй игрок установит агента в позицию $(6, 5)$, то на третьем ходе первый игрок сможет установить дрон в позицию $(3, 3)$ и т. д.

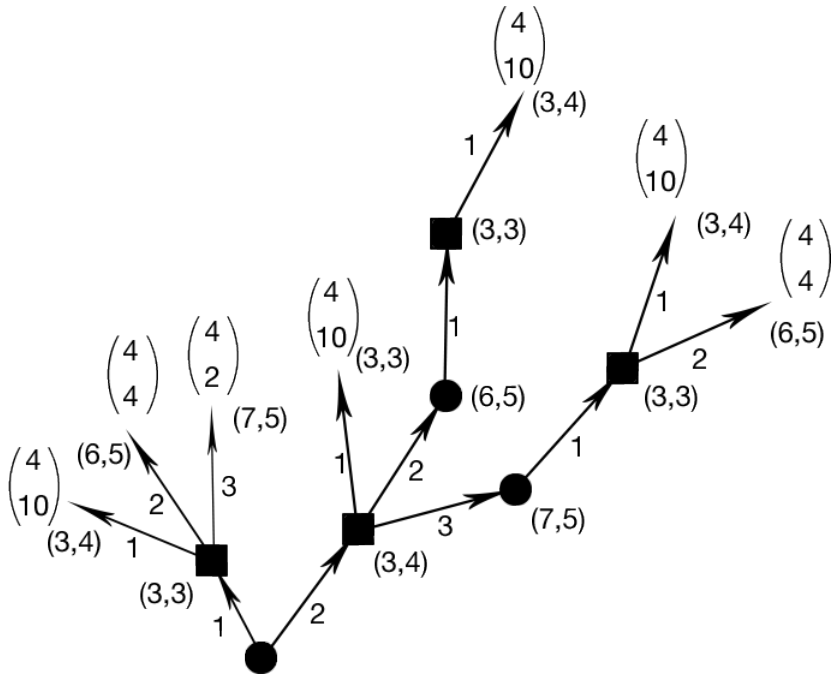

Рис. 15: Дерево игры при условии, что первым «ходит» первый игрок

Дерево решений данной игры, полученное на основании результатов программы, изображено на рис. 15.

На данном дереве множество очередности первого игрока изображено в виде кружков, а второго - квадратиков. Каждая вершина из множеств очередности помечена двойным индексом, означающим новое положение дрона на ДПСК. Одиночный индекс означает номер альтернативы для игрока. В окончательных позициях записаны выигрыши игроков.

4.3 Результат при условии, что первым «ходит» второй игрок

Рассмотри теперь ситуацию, когда первый ход делает второй игрок. Результат программы в этом случае приведен на рис. 16.

```
P1 startDiameterRoute: 8
P2 startDiameterRoute: 17
Step:1  P2  Pos:(6,5)  Dpr:17  Dn:13  CurrPayoff: 4    P1 GlobPayoff: 0    P2 GlobPayoff: 4
Step:2  P1  Pos:(3,3)  Dpr: 8  Dn: 4  CurrPayoff: 4    P1 GlobPayoff: 4    P2 GlobPayoff: 4
Step:3  P2  Pos:(3,4)  Dpr:13  Dn: 7  CurrPayoff: 6    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:2  P1  Pos:(3,4)  Dpr: 8  Dn: 6  CurrPayoff: 2    P1 GlobPayoff: 2    P2 GlobPayoff: 4
Step:3  P2  Pos:(3,3)  Dpr:13  Dn: 7  CurrPayoff: 6    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:1  P2  Pos:(7,5)  Dpr:17  Dn:15  CurrPayoff: 2    P1 GlobPayoff: 0    P2 GlobPayoff: 2
Step:2  P1  Pos:(3,3)  Dpr: 8  Dn: 4  CurrPayoff: 4    P1 GlobPayoff: 4    P2 GlobPayoff: 2
Step:3  P2  Pos:(3,4)  Dpr:15  Dn: 7  CurrPayoff: 8    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:3  P2  Pos:(6,5)  Dpr:15  Dn:13  CurrPayoff: 2    P1 GlobPayoff: 4    P2 GlobPayoff: 4
Step:2  P1  Pos:(3,4)  Dpr: 8  Dn: 6  CurrPayoff: 2    P1 GlobPayoff: 2    P2 GlobPayoff: 2
Step:3  P2  Pos:(3,3)  Dpr:15  Dn: 7  CurrPayoff: 8    P1 GlobPayoff: 4    P2 GlobPayoff:10
Step:3  P2  Pos:(6,5)  Dpr:15  Dn:13  CurrPayoff: 2    P1 GlobPayoff: 2    P2 GlobPayoff: 4
Step:4  P1  Pos:(3,3)  Dpr: 6  Dn: 4  CurrPayoff: 2    P1 GlobPayoff: 4    P2 GlobPayoff: 4
Step:5  P2  Pos:(3,4)  Dpr:13  Dn: 7  CurrPayoff: 6    P1 GlobPayoff: 4    P2 GlobPayoff:10
```

Рис. 16: Вывод программы при условии, что первым «ходит» второй игрок

Данный результат интерпретируется аналогично предыдущему результату и соответствует дереву решений, изображенному на рис. 17.

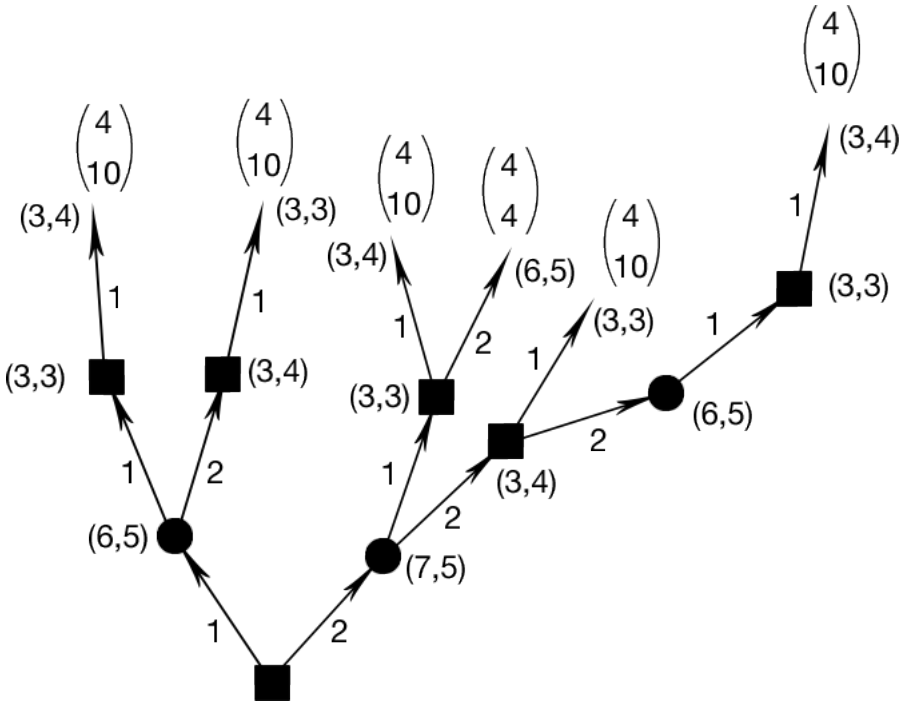

Рис. 17: Дерево игры при условии, что первым «ходит» второй игрок

4.4 Вывод

Заметим принципиальное различие между деревьями игры в двух случаях. Также минимальный выигрыш, который может гарантировать себе второй игрок когда «ходит» первым 4, а когда вторым - 2.

Таким образом, на основании полученных результатов можно сказать, что процесс развития игры зависит от того, который из игроков будет делать первый шаг.

Заключение

Анализ проблемы оперативного восстановления связи в местах природных катастроф показал, что ключевую роль здесь играет разумное использование ограниченных ресурсов, в частности, беспилотников (дронов). С учетом динамики развития процесса, было принято решение в качестве сетевой структуры использовать MANET сеть. В данном исследовании использовался подход, позволяющий рассмотреть совместный сценарий поведения узлов сети для достижения максимальной выгоды.

Рассмотрена новая постановка динамической игры управления агентами в сети. Данная игра была исследована с применением методов теории многошаговых игр. Найден алгоритм, позволяющий построить дерево решений для конечношаговой игры. Сформулировано условие существования абсолютного равновесия по Нэшу. Приведен пример игры двух лиц с отсутствием такого равновесия.

Написано программное обеспечение, позволяющее построить дерево решений для конечношаговой игры. Программный продукт протестирован на примере игры двух лиц.

Данное исследование может быть расширено использованием других типов решеток, в частности треугольной и шестиугольной. Данные типы решеток более точно соответствуют архитектуре MANET сетей. Также может быть введен кооперативный поход, позволяющий более точно описать координирование действий спасательных отрядов.

Некоторые результаты данного исследования нашли отражение в следующих публикациях [8], [9].

Список литературы

[1] Новиков Д. А. Игры и сети. // Математическая теория игр и её приложения, 2010. Т. 2. Вып. 1. С. 107–124.

[2] Yu F. R. Cognitive Radio Mobile Ad Hoc Networks. Springer, 2011. 507 с.

[3] Zhu Han, Dusit Niyato, Walid Saad, Tamer Başar, Are Hjørungnes. Game Theory in Wireless and Communication Networks. Theory, Models, and Applications. Cambridge University Press, 2012. 554 с.

[4] Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр. СПб.: БХВ-Петербург, 2012. 424 с.

[5] Петросян Л. А., Седаков. А. А. Многошаговые игры с полной информацией. // Математическая теория игр и её приложения, 2009. Вып. 26.1. С. 121–138.

[6] Dijkstra E. W. A note on two problems in connexion with graphs. // Numerische Matematik, 1959. С. 269–271.

[7] Wilson R. J. Introduction to Graph Theory. Edinburgh: Oliver and Boyd, 1972. 209 с.

[8] Тимонин Н. О. Одна динамическая игра управления агентами в сети. // Процессы управления и устойчивость: Труды 47-й международной научной конференции аспирантов и студентов / под ред. Н. В. Смирнова, Т. Е. Смирновой. СПб.: Издат. Дом С.-Петерб. гос. ун-та, 2016. Принято к публикации.

[9] Gromova E., Gromov D., Timonin N., Kirpichnikova A., Blakeway S. A dynamic game of mobile agents placement on MANET. Submitted to IEEE conference SIMS 2016.

Приложение

main.m

```
#import <Foundation/Foundation.h>
#import "NTPlayer.h"
#import "NTSpace.h"


/*
 *  Game tree creation.
 *
 *  @param i                    index of player
 *  @param players              array of players
 *  @param step                 step
 *  @param benevolenceCondition condition for benevolence
 */
void moveForPlayer(int i, NSArray *players, int step, BOOL
    benevolenceCondition) {
    NTPlayer *player = players[i];
    step++;

    for (NTPosition *position in [NTSpace sharedInstance].freePositions) {
        NTPosition *previousActiveAgentPosition = player.activeAgent.
            position;
        NSUInteger previousDiameterLength = player.currentDiameterRoute.
            length;

        for (NTPlayer *player in players) {
            player.previousDiameterLength = player.currentDiameterRoute.
                length;
        }
        [player moveActiveAgentToPosition:position];

        // Check benevolence for players
        BOOL benevolence = YES;
        if (benevolenceCondition) {
            for (NTPlayer *otherPlayer in players) {
                if (![otherPlayer isEqual:player]) {
                    benevolence = (otherPlayer.currentDiameterRoute.length >
                        otherPlayer.previousDiameterLength) ? NO : YES;
                }
            }
        }

        if ((player.currentDiameterRoute.length < player.
            previousDiameterLength) && benevolence) {
```

```
38              NSMutableString *logStirng = [NSMutableString stringWithFormat:@"
                    Step:%i P%lu Pos:(%lu,%lu) Dpr:%2lu Dn:%2.f CurrPayoff:%2.f",
                    step,(unsigned long)player.playerId + 1, position.x, position.
                    y, (unsigned long)previousDiameterLength, player.
                    currentDiameterRoute.length, previousDiameterLength - player.
                    currentDiameterRoute.length];
39
40              for (NTPlayer *player in players) {
41                  [logStirng appendString:[NSString stringWithFormat:@" P%lu
                        GlobPayoff:%2.f",(unsigned long)player.playerId+1, player.
                        startDiameterRoute.length - player.currentDiameterRoute.
                        length]];
42              }
43              NSLog(@"%@",logStirng);
44
45              for (NTPlayer *nextPlayer in players) {
46                  if (![nextPlayer isEqual:player]) {
47                      moveForPlayer((int)nextPlayer.playerId, players, step,
                            benevolenceCondition);
48                  }
49              }
50
51          }
52          [player moveActiveAgentToPosition:previousActiveAgentPosition];
53      }
54  }
55
56
57  int main(int argc, const char * argv[]) {
58
59      @autoreleasepool {
60
61          NTPlayer *p1 = [[NTPlayer alloc] initWithDefaultConfig1];
62          NTPlayer *p2 = [[NTPlayer alloc] initWithDefaultConfig4];
63
64          [p2.agents addObject:p1.activeAgent];
65          [p1.agents addObject:p2.activeAgent];
66
67          NSArray *players = @[p1, p2];
68
69          [NTSpace sharedInstance].players = players;
70
71          NSLog(@"P1 startDiameterRoute: %lu", (unsigned long)p1.
                    startDiameterRoute.length);
72          NSLog(@"P2 startDiameterRoute: %lu", (unsigned long)p2.
                    startDiameterRoute.length);
73
74          moveForPlayer(0, players, 0, YES); // Player 1 makes first step
75          moveForPlayer(1, players, 0, YES); // Player 2 makes first step
```

```
76
77      }
78      return 0;
79  }
```

NTSpace.h

```
1  #import <Foundation/Foundation.h>
2  #import "NTPosition.h"
3  #import "NTPlayer.h"
4
5
6  @interface NTSpace : NSObject
7
8  /**
9   *  Players
10  */
11 @property (nonatomic, strong, readwrite) NSArray<NTPlayer *> *players;
12
13 /**
14  *  Free positions for drons
15  */
16 @property (nonatomic, strong, readonly) NSArray<NTPosition *> *freePositions
       ;
17
18 /**
19  *  This class is a singleton. Convenience method to get NTSpace instance
20  *
21  *  @return NTSpace instance
22  */
23 + (instancetype)sharedInstance;
24
25 @end
```

NTSpace.m

```
1  #import "NTSpace.h"
2
3
4  @interface NTSpace ()
5
6  /**
7   *  Free positions for drons
8   */
9  @property (nonatomic, strong, readwrite) NSArray<NTPosition *> *
       freePositions;
10
11 @end
12
```

```
13 @implementation NTSpace
14
15
16 #pragma mark - Initializators
17
18 + (instancetype)sharedInstance {
19     static NTSpace *space = nil;
20     static dispatch_once_t onceToken;
21     dispatch_once(&onceToken, ^{
22         space = [[self alloc] init];
23         space.freePositions = [NSMutableArray new];
24     });
25
26     return space;
27 }
28
29 #pragma mark - Setter/Getter
30
31 - (NSString *)description {
32     NSMutableString *descr = [NSMutableString new];
33
34     for (NTPosition *position in self.freePositions) {
35         [descr appendString:[NSString stringWithFormat:@" (%lu, %lu) ",
               position.x, position.y]];
36     }
37
38     return descr;
39 }
40
41
42 - (NSArray<NTPosition *> *)freePositions {
43     NSMutableArray *resultPositions = [NSMutableArray new];
44
45     NTPosition *leftBottom = [NTPosition positionWithX:0 Y:0];
46     NTPosition *rightTop = [NTPosition positionWithX:0 Y:0];
47
48     NSMutableArray *takenPositions = [NSMutableArray new];
49     for (NTPlayer *p in self.players) {
50         for (PESGraphNode *node in p.agents) {
51
52             leftBottom.x = (node.position.x < leftBottom.x) ? node.position.x
                    : leftBottom.x;
53             leftBottom.y = (node.position.y < leftBottom.y) ? node.position.y
                    : leftBottom.y;
54
55             rightTop.x = (node.position.x > rightTop.x) ? node.position.x :
                   rightTop.x;
56             rightTop.y = (node.position.y > rightTop.y) ? node.position.y :
                   rightTop.y;
```

```
57
58              [takenPositions addObject:node.position];
59          }
60      }
61
62      for (int j = (int)leftBottom.x; j <= (int)rightTop.x; j++) {
63          for (int i = (int)leftBottom.y; i <= (int)rightTop.y; i++) {
64
65              BOOL flag = YES;
66              for (NTPosition *position in takenPositions) {
67                  if ((int)position.x == j && (int)position.y == i) {
68                      flag = NO;
69                  }
70              }
71
72              if (flag) {
73                  [resultPositions addObject:[NTPosition positionWithX:j Y:i]];
74              }
75
76          }
77      }
78
79      return [NSArray arrayWithArray:resultPositions];
80  }
81
82
83  @end
```

NTPosition.h

```
1  #import <Foundation/Foundation.h>
2
3  @interface NTPosition : NSObject
4
5  /**
6   *  X coordinate in Cartesian coordinate system
7   */
8  @property (nonatomic, readwrite) NSUInteger x;
9
10 /**
11  *  Y coordinate in Cartesian coordinate system
12  */
13 @property (nonatomic, readwrite) NSUInteger y;
14
15 /**
16  *  Convenience initializer that allows create NTPositon with two Cartesian
        coordinates
17  *
18  *  @param x X coordinate
```

```
 *  @param y Y coordinate
 *
 *  @return NTPosition instance
 */
+ (instancetype)positionWithX:(NSUInteger)x Y:(NSUInteger)y;

/**
 *  Check, is the position nearby another position
 *
 *  @param position NTPosition to check
 *
 *  @return YES/NO
 */
- (BOOL)isNearWithPosition:(NTPosition *)position;

@end
```

NTPosition.m

```
#import "NTPosition.h"

@interface NTPosition ()

@end

@implementation NTPosition

+ (instancetype)positionWithX:(NSUInteger)x Y:(NSUInteger)y {
    NTPosition *p = [[self alloc] init];
    p.x = x;
    p.y = y;
    return p;
}

- (BOOL)isNearWithPosition:(NTPosition *)position {
    BOOL result = (sqrt( pow((double)self.x - (double)position.x, 2) + pow((
        double)self.y - (double)position.y, 2)) == 1) ? YES : NO;
    return result;
}

@end
```

NTPlayer.h

```
#import <Foundation/Foundation.h>
#include "NTPosition.h"
#import "PESGraph.h"
#import "PESGraph+Diameter.h"
#import "PESGraphNode_Position.h"
```

```
#import "PESGraphEdge.h"
#import "PESGraphRoute.h"

@interface NTPlayer : NSObject

/**
 *  Player identifier
 */
@property (nonatomic, readonly) NSUInteger playerId;

/**
 *  Players previous diameter length
 */
@property (nonatomic, readwrite) NSUInteger previousDiameterLength;

/**
 *  All agents
 */
@property (nonatomic, strong, readonly) NSMutableArray<PESGraphNode *> *
    agents;

/**
 *  Drone
 */
@property (nonatomic, strong, readonly) PESGraphNode *activeAgent;

/**
 *  Diameter at first game stage
 */
@property (nonatomic, strong, readonly) PESGraphRoute *startDiameterRoute;

/**
 *  Diameter at current game stage
 */
@property (nonatomic, strong, readonly) PESGraphRoute *currentDiameterRoute;

/**
 *  Move active agent to new position
 *
 *  @param position New active agent position
 */
- (void)moveActiveAgentToPosition:(NTPosition *)position;

/**
 *  Initializators with different agents positions
 *
 *  @return instance of NTPlayer
 */
- (instancetype)initWithDefaultConfig1;
```

```
54 - (instancetype)initWithDefaultConfig2;
55 - (instancetype)initWithDefaultConfig3;
56 - (instancetype)initWithDefaultConfig4;
57 - (instancetype)initWithDefaultConfig5;
58 - (instancetype)initWithDefaultConfig6;
59 - (instancetype)initWithDefaultConfig7;
60 - (instancetype)initWithDefaultConfig8;
61
62 @end
```

NTPlayer.m

```
1 #import "NTPlayer.h"
2 #import "NTSpace.h"
3
4 @interface NTPlayer ()
5
6 /**
7  *  Players graph
8  */
9 @property (nonatomic, strong, readwrite) PESGraph *graph;
10
11 /**
12  *  Source node in diameter
13  */
14 @property (nonatomic, strong, readwrite) PESGraphNode *sourse;
15
16 /**
17  *  Destignation node in diameter
18  */
19 @property (nonatomic, strong, readwrite) PESGraphNode *destignation;
20
21 /**
22  *  All agents
23  */
24 @property (nonatomic, strong, readwrite) NSMutableArray<PESGraphNode *> *
     agents;
25
26 /**
27  *  Diameter at first game stage
28  */
29 @property (nonatomic, strong, readwrite) PESGraphRoute *startDiameterRoute;
30
31 /**
32  *  Diameter at current game stage
33  */
34 @property (nonatomic, strong, readwrite) PESGraphRoute *currentDiameterRoute
     ;
35
```

```
36
37 @end
38
39
40 @implementation NTPlayer
41
42 #pragma mark - Setter/Getter
43
44 - (PESGraph *)graph {
45     PESGraph *graph = [PESGraph new];
46
47     for (int i = 0; i < (int)self.agents.count; i++) {
48         for (int j = 0; j < (int)self.agents.count; j++) {
49             if (i > j) {
50                 PESGraphNode *fromNode = self.agents[i];
51                 PESGraphNode *toNode = self.agents[j];
52
53                 if ([fromNode.position isNearWithPosition:toNode.position]) {
54                     PESGraphEdge *newEdge = [PESGraphEdge edgeWithName:[
                            NSString stringWithFormat:@"%@ <-> %@", fromNode.
                            identifier, toNode.identifier] andWeight:@1];
55                     [graph addBiDirectionalEdge:newEdge fromNode:fromNode
                            toNode:toNode];
56                 }
57
58             }
59         }
60     }
61
62     return graph;
63 }
64
65
66 - (PESGraphRoute *)startDiameterRoute {
67     PESGraphRoute *route;
68     if (!_startDiameterRoute) {
69         route = [self.graph diameter];
70
71         self.sourse = route.startingNode;
72         self.destignation = route.endingNode;
73         _startDiameterRoute = route;
74
75     } else {
76         route = _startDiameterRoute;
77     }
78
79     return route;
80 }
81
```

```
82
83 - (PESGraphRoute *)currentDiameterRoute {
84     if (!self.startDiameterRoute) {
85         [self startDiameterRoute];
86     }
87
88     return [self.graph shortestRouteFromNode:self.sourse toNode:self.
           destignation];
89 }
90
91
92 #pragma mark - Initializators
93
94 - (instancetype)initWithDefaultConfig1 {
95     self = [super init];
96     if (self) {
97
98         _playerId = 0;
99
100        PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
               :[NTPosition positionWithX:1 Y:3]];
101        PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
               :[NTPosition positionWithX:2 Y:3]];
102        PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
               :[NTPosition positionWithX:2 Y:4]];
103        PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
               :[NTPosition positionWithX:2 Y:5]];
104        PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
               :[NTPosition positionWithX:3 Y:5]];
105        PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
               :[NTPosition positionWithX:4 Y:5]];
106        PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
               :[NTPosition positionWithX:4 Y:4]];
107        PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
               :[NTPosition positionWithX:4 Y:3]];
108        PESGraphNode *node9 = [PESGraphNode nodeWithIdentifier:@"9" position
               :[NTPosition positionWithX:5 Y:3]];
109
110        PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"100"
               position:[NTPosition positionWithX:0 Y:0]];
111
112        _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
               node4, node5, node6, node7, node8, node9, activeNode]];
113        _activeAgent = activeNode;
114    }
115
116    return self;
117 }
118
```

```
- (instancetype)initWithDefaultConfig2 {
    self = [super init];
    if (self) {
        _playerId = 1;

        PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
            :[NTPosition positionWithX:1 Y:2]];
        PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
            :[NTPosition positionWithX:2 Y:2]];
        PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
            :[NTPosition positionWithX:2 Y:1]];
        PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
            :[NTPosition positionWithX:3 Y:1]];
        PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
            :[NTPosition positionWithX:4 Y:1]];
        PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
            :[NTPosition positionWithX:4 Y:2]];
        PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
            :[NTPosition positionWithX:5 Y:2]];

        PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"200"
            position:[NTPosition positionWithX:0 Y:0]];

        _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
            node4, node5, node6, node7, activeNode]];
        _activeAgent = activeNode;
    }

    return self;
}

- (instancetype)initWithDefaultConfig3 {
    self = [super init];
    if (self) {
        _playerId = 1;

        PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
            :[NTPosition positionWithX:1 Y:2]];
        PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
            :[NTPosition positionWithX:2 Y:2]];
        PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
            :[NTPosition positionWithX:2 Y:1]];
        PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
            :[NTPosition positionWithX:3 Y:1]];
        PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
            :[NTPosition positionWithX:4 Y:1]];
        PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
            :[NTPosition positionWithX:5 Y:1]];
        PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
```

```
            :[NTPosition positionWithX:5 Y:2]];
153         PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
            :[NTPosition positionWithX:6 Y:2]];
154
155         PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"200"
            position:[NTPosition positionWithX:0 Y:0]];
156
157         _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
            node4, node5, node6, node7, node8, activeNode]];
158         _activeAgent = activeNode;
159     }
160
161     return self;
162 }
163
164 - (instancetype)initWithDefaultConfig4 {
165     self = [super init];
166     if (self) {
167         _playerId = 1;
168
169         PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
            :[NTPosition positionWithX:3 Y:7]];
170         PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
            :[NTPosition positionWithX:3 Y:6]];
171         PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
            :[NTPosition positionWithX:4 Y:6]];
172         PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
            :[NTPosition positionWithX:5 Y:6]];
173         PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
            :[NTPosition positionWithX:6 Y:6]];
174         PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
            :[NTPosition positionWithX:7 Y:6]];
175         PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
            :[NTPosition positionWithX:8 Y:6]];
176         PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
            :[NTPosition positionWithX:8 Y:5]];
177         PESGraphNode *node9 = [PESGraphNode nodeWithIdentifier:@"9" position
            :[NTPosition positionWithX:8 Y:4]];
178         PESGraphNode *node10 = [PESGraphNode nodeWithIdentifier:@"10"
            position:[NTPosition positionWithX:7 Y:4]];
179         PESGraphNode *node11 = [PESGraphNode nodeWithIdentifier:@"11"
            position:[NTPosition positionWithX:6 Y:4]];
180         PESGraphNode *node12 = [PESGraphNode nodeWithIdentifier:@"12"
            position:[NTPosition positionWithX:6 Y:3]];
181         PESGraphNode *node13 = [PESGraphNode nodeWithIdentifier:@"13"
            position:[NTPosition positionWithX:6 Y:2]];
182         PESGraphNode *node14 = [PESGraphNode nodeWithIdentifier:@"14"
            position:[NTPosition positionWithX:5 Y:2]];
183         PESGraphNode *node15 = [PESGraphNode nodeWithIdentifier:@"15"
```

```
            position:[NTPosition positionWithX:4 Y:2]];
184         PESGraphNode *node16 = [PESGraphNode nodeWithIdentifier:@"16"
            position:[NTPosition positionWithX:3 Y:2]];
185         PESGraphNode *node17 = [PESGraphNode nodeWithIdentifier:@"17"
            position:[NTPosition positionWithX:3 Y:1]];
186         PESGraphNode *node18 = [PESGraphNode nodeWithIdentifier:@"18"
            position:[NTPosition positionWithX:3 Y:5]];
187         PESGraphNode *node19 = [PESGraphNode nodeWithIdentifier:@"19"
            position:[NTPosition positionWithX:3 Y:8]];
188
189         PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"200"
            position:[NTPosition positionWithX:0 Y:0]];
190
191         _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
            node4, node5, node6, node7, node8, node9, node10, node11, node12,
             node13, node14, node15, node16, node17, node18, node19,
            activeNode]];
192         _activeAgent = activeNode;
193     }
194
195     return self;
196 }
197
198 - (instancetype)initWithDefaultConfig5 {
199     self = [super init];
200     if (self) {
201         _playerId = 0;
202
203         PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
            :[NTPosition positionWithX:1 Y:3]];
204         PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
            :[NTPosition positionWithX:2 Y:3]];
205         PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
            :[NTPosition positionWithX:2 Y:4]];
206         PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
            :[NTPosition positionWithX:2 Y:5]];
207         PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
            :[NTPosition positionWithX:2 Y:6]];
208         PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
            :[NTPosition positionWithX:3 Y:6]];
209         PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
            :[NTPosition positionWithX:4 Y:6]];
210         PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
            :[NTPosition positionWithX:5 Y:6]];
211         PESGraphNode *node9 = [PESGraphNode nodeWithIdentifier:@"9" position
            :[NTPosition positionWithX:5 Y:5]];
212         PESGraphNode *node10 = [PESGraphNode nodeWithIdentifier:@"10"
            position:[NTPosition positionWithX:5 Y:4]];
213         PESGraphNode *node11 = [PESGraphNode nodeWithIdentifier:@"11"
```

```
            position:[NTPosition positionWithX:5 Y:3]];
214         PESGraphNode *node12 = [PESGraphNode nodeWithIdentifier:@"12"
            position:[NTPosition positionWithX:6 Y:3]];
215         PESGraphNode *node13 = [PESGraphNode nodeWithIdentifier:@"13"
            position:[NTPosition positionWithX:6 Y:2]];
216         PESGraphNode *node14 = [PESGraphNode nodeWithIdentifier:@"14"
            position:[NTPosition positionWithX:7 Y:2]];
217         PESGraphNode *node15 = [PESGraphNode nodeWithIdentifier:@"15"
            position:[NTPosition positionWithX:8 Y:2]];
218         PESGraphNode *node16 = [PESGraphNode nodeWithIdentifier:@"16"
            position:[NTPosition positionWithX:8 Y:3]];
219         PESGraphNode *node17 = [PESGraphNode nodeWithIdentifier:@"17"
            position:[NTPosition positionWithX:9 Y:3]];
220         PESGraphNode *node18 = [PESGraphNode nodeWithIdentifier:@"18"
            position:[NTPosition positionWithX:4 Y:5]];
221
222         PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"100"
            position:[NTPosition positionWithX:0 Y:0]];
223
224         _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
            node4, node5, node6, node7, node8, node9, node10, node11, node12,
             node13, node14, node15, node16, node17, node18, activeNode]];
225         _activeAgent = activeNode;
226     }
227
228     return self;
229 }
230
231 - (instancetype)initWithDefaultConfig6 {
232     self = [super init];
233     if (self) {
234         _playerId = 1;
235
236         PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
            :[NTPosition positionWithX:4 Y:1]];
237         PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
            :[NTPosition positionWithX:4 Y:2]];
238         PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
            :[NTPosition positionWithX:3 Y:2]];
239         PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
            :[NTPosition positionWithX:2 Y:2]];
240         PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
            :[NTPosition positionWithX:2 Y:3]];
241         PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
            :[NTPosition positionWithX:2 Y:4]];
242         PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
            :[NTPosition positionWithX:3 Y:4]];
243         PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
            :[NTPosition positionWithX:4 Y:4]];
```

```
244         PESGraphNode *node9 = [PESGraphNode nodeWithIdentifier:@"9" position
                :[NTPosition positionWithX:5 Y:4]];
245         PESGraphNode *node10 = [PESGraphNode nodeWithIdentifier:@"10"
                position:[NTPosition positionWithX:6 Y:4]];
246         PESGraphNode *node11 = [PESGraphNode nodeWithIdentifier:@"11"
                position:[NTPosition positionWithX:6 Y:5]];
247         PESGraphNode *node12 = [PESGraphNode nodeWithIdentifier:@"12"
                position:[NTPosition positionWithX:6 Y:6]];
248         PESGraphNode *node13 = [PESGraphNode nodeWithIdentifier:@"13"
                position:[NTPosition positionWithX:7 Y:6]];
249         PESGraphNode *node14 = [PESGraphNode nodeWithIdentifier:@"14"
                position:[NTPosition positionWithX:8 Y:6]];
250         PESGraphNode *node15 = [PESGraphNode nodeWithIdentifier:@"15"
                position:[NTPosition positionWithX:8 Y:5]];
251         PESGraphNode *node16 = [PESGraphNode nodeWithIdentifier:@"16"
                position:[NTPosition positionWithX:8 Y:4]];
252         PESGraphNode *node17 = [PESGraphNode nodeWithIdentifier:@"17"
                position:[NTPosition positionWithX:9 Y:4]];
253
254         PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"200"
                position:[NTPosition positionWithX:0 Y:0]];
255
256         _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
                node4, node5, node6, node7, node8, node9, node10, node11, node12,
                 node13, node14, node15, node16, node17, activeNode]];
257         _activeAgent = activeNode;
258     }
259
260     return self;
261 }
262
263 - (instancetype)initWithDefaultConfig7 {
264     self = [super init];
265     if (self) {
266         _playerId = 0;
267
268         PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
                :[NTPosition positionWithX:1 Y:4]];
269         PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
                :[NTPosition positionWithX:2 Y:4]];
270         PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
                :[NTPosition positionWithX:3 Y:4]];
271         PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
                :[NTPosition positionWithX:3 Y:5]];
272         PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
                :[NTPosition positionWithX:3 Y:6]];
273         PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
                :[NTPosition positionWithX:4 Y:6]];
274         PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
```

```
            :[NTPosition positionWithX:5 Y:6]];
275         PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
            :[NTPosition positionWithX:6 Y:6]];
276         PESGraphNode *node9 = [PESGraphNode nodeWithIdentifier:@"9" position
            :[NTPosition positionWithX:6 Y:5]];
277         PESGraphNode *node10 = [PESGraphNode nodeWithIdentifier:@"10"
            position:[NTPosition positionWithX:6 Y:4]];
278         PESGraphNode *node11 = [PESGraphNode nodeWithIdentifier:@"11"
            position:[NTPosition positionWithX:7 Y:4]];
279         PESGraphNode *node12 = [PESGraphNode nodeWithIdentifier:@"12"
            position:[NTPosition positionWithX:4 Y:5]];
280
281         PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"100"
            position:[NTPosition positionWithX:0 Y:0]];
282
283         _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
            node4, node5, node6, node7, node8, node9, node10, node11, node12,
             activeNode]];
284         _activeAgent = activeNode;
285     }
286
287     return self;
288 }
289
290 - (instancetype)initWithDefaultConfig8 {
291     self = [super init];
292     if (self) {
293         _playerId = 1;
294
295         PESGraphNode *node1 = [PESGraphNode nodeWithIdentifier:@"1" position
            :[NTPosition positionWithX:5 Y:1]];
296         PESGraphNode *node2 = [PESGraphNode nodeWithIdentifier:@"2" position
            :[NTPosition positionWithX:5 Y:2]];
297         PESGraphNode *node3 = [PESGraphNode nodeWithIdentifier:@"3" position
            :[NTPosition positionWithX:3 Y:2]];
298         PESGraphNode *node4 = [PESGraphNode nodeWithIdentifier:@"4" position
            :[NTPosition positionWithX:3 Y:3]];
299         PESGraphNode *node5 = [PESGraphNode nodeWithIdentifier:@"5" position
            :[NTPosition positionWithX:3 Y:4]];
300         PESGraphNode *node6 = [PESGraphNode nodeWithIdentifier:@"6" position
            :[NTPosition positionWithX:2 Y:4]];
301         PESGraphNode *node7 = [PESGraphNode nodeWithIdentifier:@"7" position
            :[NTPosition positionWithX:2 Y:5]];
302         PESGraphNode *node8 = [PESGraphNode nodeWithIdentifier:@"8" position
            :[NTPosition positionWithX:2 Y:6]];
303         PESGraphNode *node9 = [PESGraphNode nodeWithIdentifier:@"9" position
            :[NTPosition positionWithX:3 Y:6]];
304         PESGraphNode *node10 = [PESGraphNode nodeWithIdentifier:@"10"
            position:[NTPosition positionWithX:4 Y:6]];
```

```
305         PESGraphNode *node11 = [PESGraphNode nodeWithIdentifier:@"11"
                position:[NTPosition positionWithX:5 Y:6]];
306         PESGraphNode *node12 = [PESGraphNode nodeWithIdentifier:@"12"
                position:[NTPosition positionWithX:6 Y:6]];
307         PESGraphNode *node13 = [PESGraphNode nodeWithIdentifier:@"13"
                position:[NTPosition positionWithX:6 Y:5]];
308         PESGraphNode *node14 = [PESGraphNode nodeWithIdentifier:@"14"
                position:[NTPosition positionWithX:6 Y:4]];
309         PESGraphNode *node15 = [PESGraphNode nodeWithIdentifier:@"15"
                position:[NTPosition positionWithX:7 Y:4]];
310         PESGraphNode *node16 = [PESGraphNode nodeWithIdentifier:@"16"
                position:[NTPosition positionWithX:4 Y:5]];
311
312         PESGraphNode *node17 = [PESGraphNode nodeWithIdentifier:@"17"
                position:[NTPosition positionWithX:4 Y:2]];
313
314         PESGraphNode *activeNode = [PESGraphNode nodeWithIdentifier:@"200"
                position:[NTPosition positionWithX:0 Y:0]];
315
316         _agents = [NSMutableArray arrayWithArray:@[node1, node2, node3,
                node4, node5, node6, node7, node8, node9, node10, node11, node12,
                 node13, node14, node15, node16, node17, activeNode]];
317         _activeAgent = activeNode;
318     }
319
320     return self;
321 }
322
323
324 #pragma mark - Methods
325
326 - (void)moveActiveAgentToPosition:(NTPosition *)position {
327     self.activeAgent.position = position;
328 }
329
330
331 @end
```

PESGraph+Diameter.h

```
1 #import "PESGraph.h"
2
3 @interface PESGraph (Diameter)
4
5 /**
6  *  Returns the route object that describes the diameter of graph.
7  *
8  *  @return either a PESGraphRoute object or nil, if no diameter is possible
9  */
```

```
- (PESGraphRoute *)diameter;

@end
```

PESGraph+Diameter.m

```
#import "PESGraph+Diameter.h"
#import "PESGraphNode.h"
#import "PESGraphRoute.h"
#import "PESGraphEdge.h"

@implementation PESGraph (Diameter)

- (PESGraphRoute *)diameter {

    PESGraphRoute *result = nil;
    for (NSString *sourseKey in self.nodes) {
        PESGraphNode *sourse = [self.nodes objectForKey:sourseKey];

        for (NSString *destignationKey in self.nodes) {
            PESGraphNode *destignation = [self.nodes objectForKey:
            destignationKey];

            if (![sourse isEqual:destignation]) {
                PESGraphRoute *route = [self shortestRouteFromNode:sourse
                toNode:destignation];
                result = (route.length > result.length) ? route : result;
            }
        }
    }
    return result;
}

@end
```

PESGraphNode_Position.h

```
#import "PESGraphNode.h"
#import "NTPosition.h"

@interface PESGraphNode ()

/**
 *  Node position in Cartesian system
 */
@property (nonatomic, strong) NTPosition *position;

@end
```