Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

КУРСОВАЯ РАБОТА

Исследовательский проект на тему

"Равновесие Нэша в позиционных играх и комбинаторные игры"

Выполнил студент группы 182, 3 курса,
Рахматуллин Рамазан Зофарович

Руководитель КР:
К.ф.-м.н., Доцент, Гурвич Владимир Александрович

Москва 2021

Содержание

1 Аннотация

1.1 Аннотация

В данной работе рассматривается разрешимость по Нэшу позиционных игр в чистых стационарных стратегиях на графах с терминальной и аддитивной функциями стоимости. Целью работы являлась разработка эффективных алгоритмов проверки разрешимости по Нэшу игр на графах из упомянутых классов. С помощью этих алгоритмов предлагалось найти примеры, доказывающие гипотезы о неразрешимости по Нэшу игр в следующих классах: игры на графах двух игроков с аддитивной функцией стоимости с положительными стоимостями, и игры на графах трех и более игроков с терминальной функцией стоимости, где цикл является наихудшим исходом для всех игроков. Результатом работы является реализация полученных алгоритмов, доказательство отсутствия таких примеров для графов маленького размера путем программного перебора, используя полученные программы. На основании полученных данных была выдвинута гипотеза о разрешимости класса, превосходящего класс игр с терминальной функцией стоимости с циклом как наихудшим исходом.

1.2 Abstract

This work studies Nash solvability in pure stationary strategies of positional graphical games with additive and terminal cost functions. The goal of this work was to develop efficient algorithms to check Nash solvability of games on graphs from mentioned classes. Those algorithms were expected to be used to find examples that prove hypotheses about Nash unsolvability of games from the following classes: graphical games with additive and positive cost function for two players, and graphical games with terminal cost function for three and more players, with cycle being the worst option for all players. Our main result is an implementation of such algorithms, proof of Nash solvability of aforementioned classes for small

size graphs by bruteforce on computer using obtained programs. Based on our findings, we made a conjecture about solvability of superclass of games with terminal cost function, with cycle being the worst option.

1.3 Список ключевых слов

Равновесие Нэша, игры с полной информацией, чистые стационарные стратегии, игры на графах с аддитивной функцией стоимости, игры на графах с терминальной функцией стоимости.

2 Введение

2.1 Описание предметной области

Джон Нэш ввел понятие равновесия (РН) [1] для игр с n игроками. С момента его появления, данный термин стал одним из ключевых в теории игр. Интерес вызван в первую очередь тем, что теорию игр в целом и РН в частности можно применить в самых различных сферах, например: в социологии, политологии, экономике и других. Для некоторых игр существование и поиск РН являются хорошо изученными задачами, однако во множестве направлений критерий существования и поиск РН остаются нерешенными проблемами.

2.2 Постановка задачи

В данной работе изучаются игры на графах в чистых стационарных стратегиях.

Будем рассматривать игру n игроков, пронумерованных от 1 до n. Пусть задан ориентированный граф $G = (V, E)$, а также начальная вершина s_0. Обозначим за $V_{\mathbf{T}}$ множество вершин с исходящей степенью, равной нулю. Остальные вершины разделены на n классов, таким образом $V = V_1 \cup V_2 \ldots \cup V_n \cup V_{\mathbf{T}}$, причем вершины класса V_i контролируются игроком i. Вершины множества $V_{\mathbf{T}}$ называются терминальными или терминалами.

Определение 1 (Стратегия, множество стратегий)**.** *Стратегией* x_i игрока i будем называть выбор одного исходящего ребра для каждой вершины $v \in V_i$. Выбор ребра e из вершины v означает, что если в процессе игры мы попадем в вершину v, то из нее мы обязательно выйдем по ребру e. Множество всех x_i задает *множество стратегий* X_i игрока i.

Определение 2 (Профиль стратегий, исход игры, игровая форма)**.** Набор стратегий игроков $x = (x_1, x_2, \ldots, x_n)$ называется *профилем стратегий*,

$x \in X = (X_1 \times X_2 \times \ldots \times X_n)$. Используя определение стратегии, профиль задает ровно одно исходящее ребро из каждой нетерминальной вершины. Если начать в вершине s_0 и ходить по выбранным ребрам, то данный процесс либо зацикливается, либо заканчивается в терминальной вершине. Полученный путь o (который может зациклиться) будем называть *исходом игры*. Множество всевозможных исходов обозначим за O. Введем вспомогательную функцию $f : X \to O$, переводящую профили в исходы. Эта функция называется *игровой формой*.

Для сравнения исходов необходимо определить функцию стоимости, переводящую исход в число для каждого игрока. Введем функцию $H_i : O \to \mathbb{R}$. Обозначим за $H_i(o)$ стоимость исхода o для игрока i. Дополнительно обозначим $H(i, o) = H_i(o)$. Игроки стараются минимизировать стоимость исхода, изменяя свою стратегию (на стратегию остальных игроков они напрямую не влияют).

Определение 3 (Равновесие Нэша)**.** Профиль x_0 называется *Равновесием Нэша(РН)*, если для любого игрока i верно, что этот игрок не может поменять свою стратегию так, чтобы новый профиль x_i^* был лучше для этого игрока (при этом стратегии остальных игроков не меняются). Формально, $\forall i, x_i^* : H_i(f(x_i^*)) \geq H_i(f(x_0))$. Здесь x_i^* обозначает профиль x_0, в котором стратегия игрока i заменена на любую другую.

Определение 4 (Игра, игровая структура, класс игр, разрешенные функции стоимости)**.** Кортеж (G, D, H, s_0) называется *игрой*. Здесь G — ориентированный граф, D — разбиение вершин графа $V = V_1 \cup V_2 \ldots \cup V_n \cup V_{\mathbf{T}}$, H — функция стоимости исходов, s_0 — начальная вершина.

Кортеж (G, D, s_0) называется *игровой структурой*.

Класс игр — множество всевозможных игр, в которых функция H удовлетворяет некоторым условиям, задающим этот класс. Такие функции будем называть *разрешенными* в этом классе.

Определение 5 (Разрешимость по Нэшу)**.** Игра называется *разрешимой по Нэшу*, если в ней существует равновесие Нэша. Аналогично игровая структура разрешима по Нэшу в каком-то классе игр, если для любой разрешенной функции из рассматриваемого класса игр, полученная игра разрешима. Аналогично класс игр разрешим по Нэшу, если любая игра класса разрешима.

Для простоты далее будем говорить, что игра/игровая структура/класс игр разрешим(-а)/неразрешим(-а). Рассмотрим два класса игр A и B, причем $A \subset B$. Тогда разрешимость B влечет разрешимость A. Из этого следует, что при поиске игры без РН стоит искать пример, который принадлежит как можно меньшему классу. Тогда этот пример будет доказывать неразрешимость не только этого класса, но и всех надклассов.

2.2.1 Аддитивная функция стоимости

Определим для каждого игрока функцию $h_i : E \to \mathbb{R}$. $h_i(e) = w$ означает, что для игрока i ребро e стоит w. Тогда если исход задается путем $o = (e_1, e_2, \ldots, e_p)$, положим $H_i(o) = \sum_{j=1}^{p} h_i(e_j)$. Если же исход зацикливается, его стоимость будем считать равной бесконечности со знаком, равным знаку у суммы стоимостей ребер на цикле (поскольку предполагается, что мы будем бесконечно ходить по циклу). Формально, пусть o — исход, который зацикливается, и C_o — множество ребер, лежащих на цикле. Тогда $H_i(o) = \infty \cdot (\sum_{e \in C_o} h_i(e)) = \infty \cdot sgn(\sum_{e \in C_o} h_i(e))$. Для удобства будем рассматривать такие h_i, что сумма на любом достижимом цикле не равна нулю, чтобы значение H_i всегда было определено.

Определение 6 (Аддитивная функция стоимости)**.** Функция H, определенная выше, называется *аддитивной функцией стоимости* (также кумулятивной функцией стоимости). Игры с такой функцией стоимости задают класс игр с аддитивной (кумулятивной) функцией стоимости.

Заметим, что стоимость незацикливающихся исходов не зависит от номера вершины, в которой кончается игра. Поэтому, если $|V_{\mathbf{T}}| > 1$, то можно постро-

ить эквивалентную игровую структуру, заменив все терминальные вершины на одну, сохранив при этом исходные ребра.

В данной работе нас будут интересовать игры двух игроков, в которых $\forall i, e : h_i(e) > 0$, из чего следует, что $\forall i, o : H_i(o) > 0$ и что любой зацикливающийся исход будет стоить $+\infty$. Раз все зацикливающиеся исходы эквивалентны, обозначим их всех как $c \in O$. Поскольку игроки минимизируют стоимость исхода, c всегда будет наихудшим исходом для любого игрока.

2.2.2 Терминальная функция стоимости

Скажем, что от исхода o нас интересует лишь конечная вершина пути, либо тот факт, что исход зацикливается (но неважно как именно). Тогда множество исходов задается как $O = V_{\mathbf{T}} \cup \{c\}$, где c будет обозначать зацикливающиеся исходы. Тогда скажем, что $H_i : O \to \mathbb{R}$ может быть любой функцией. Заметим, что при поиске примеров без РН можно предположить, что H_i инъективно. Если это не так, то можно задать любой порядок на равных значениях, и если РН не существовало, то оно не появится. Так как нас интересует только порядок, можно считать, что H_i задает перестановку, таким образом $H_i : O \to \{0, 1, \ldots, |V_{\mathbf{T}}|\}$ (для удобства перестановка начинается с нуля). Для удобства также скажем, что в этом классе игр игроки максимизируют свой результат, превращая H_i в функцию платежа от полученного исхода. Таким образом, платеж, равный 0, считается наихудшим для игрока, в то время как платеж $|V_{\mathbf{T}}|$ считается наилучшим.

Определение 7 (Терминальная функция стоимости)**.** Функция H, определенная выше, называется *терминальной функцией стоимости*. Игры с такой функцией стоимости задают класс игр с терминальной функцией стоимости.

Так как функция H_i задает порядок на исходах, можно ввести понятие лучшего исхода: исход a лучше исхода b для игрока i, если $H_i(a) > H_i(b)$. Аналогично определяется наихудший ($H_i(o) = 0$) и наилучший ($H_i(o) = |V_{\mathbf{T}}|$) исход.

В этой работе будут рассмотрены игры трех и более игроков. Нас интересует разрешимость класса игр, в которых цикл является наихудшим исходом для всех игроков. То есть функции H_i удовлетворяют свойству $\forall i : H_i(c) = 0$.

2.3 Структура работы

Идея данной работы заключалась в разработке и реализации эффективного алгоритма, который для заданной игровой структуры (тройки (G, D, s_0)) проверяет ее разрешимость в рассматриваемом классе. Необходимо было придумать такой алгоритм для обоих рассматриваемых классов: игры с аддитивной и терминальной функциями стоимости. Программы писались и разрабатывались на языке `C++`, поскольку он является одним из самых быстрых и эффективных высокоуровневых языков программирования, а для нашей программы в первую очередь важна скорость работы. Так как классы значительно отличались, разработка алгоритма для каждого класса являлась отдельной задачей. Эту часть работы покрывает раздел 5. После получения такого алгоритма, его необходимо применить на перспективных игровых структурах в поисках примера без PH. Изначально предполагалось найти этот пример на графах маленького размера путем ручного перебора. Однако было решено разработать методы генерации графов, среди которых должен существовать искомый пример, эта часть работы покрывается в разделе 4. Наконец, результатом работы является программное доказательство отсутствия примеров без PH в данных классах для графов маленького размера. Для терминального класса также была выдвинута гипотеза о разрешимости надкласса игр, где цикл является наихудшим исходом. Описание гипотезы доступно в разделе 6, более подробные результаты и методы доказательства разрешимости доступны в разделе 5.

3 Обзор литературы

Рассмотрим релевантные классы игр, в которых известен результат о разрешимости.

Для начала отметим, что рассматриваемый нами класс игр затрагивает только чистые стратегии, то есть те стратегии, в которых исход игры детерминирован в зависимости от профиля стратегий. Нэш в своей работе [1] также рассматривал смешанные стратегии: распределение вероятностей над чистыми стратегиями. В своей работе он доказал, что в любой игре, в которой разрешены смешанные стратегии (то есть всевозможные распределения над чистыми стратегиями), существует РН. Нас интересует разрешимость в чистых стратегиях.

Для игр с аддитивной и терминальной функциями стоимости на ациклических графах гарантируется разрешимость [2]. Этот результат легко получить, применив метод обратной индукции.

3.1 Аддитивный случай

Рассмотрим аддитивный случай. Пример без РН с $n = 3$ игроками с положительными стоимостями был получен в [7]. Стоит отметить, что неразрешимость класса с $n = 3$ игроками означает неразрешимость классов с $n > 3$. В упомянутой работе также представлены известные науке результаты о смежных классах, таких как усредненная функция стоимости, когда стоимость пути равна не сумме ребер, а среднему. В другой работе [9] представлен пример с двумя игроками без РН для случая, когда все циклы стоят $+\infty$. Для $n = 2$ примеры без РН с произвольными стоимостями также известны. Открытым остается вопрос о разрешимости классов игр двух игроков с положительными стоимостями ребер. В данной работе были разобраны графы маленьких размеров, на них пример найден не был.

3.2 Терминальный случай

В терминальном случае рассматриваются различные формулировки, например равномерное РН, а также улучшающие циклы [8]. Нас будет интересовать стандартная формулировка РН. Было доказано [9], что случай $n = 2$ разрешим по Нэшу в классе игр с терминальной функцией стоимости. С другой стороны, в [3] был приведен минимальный пример для $n = 3$ игроков без РН, в котором однако цикл не является худшим исходом для всех игроков. В работе была выдвинута гипотеза, что существует пример без РН для $n \geq 3$ игроков, в котором цикл был бы наихудшим исходом для всех игроков. Данная гипотеза была основана на таблице из [8]. А именно, условие о том, что цикл является наихудшим исходом для всех игроков, не влияло на результат разрешимости класса ни в одном из 15 рассмотренных случаев, вследствие чего было выдвинуто предположение, что в данном случае это условие также не влияет на результат. Результаты нашей работы говорят об обратном.

4 Генерация графов

Опишем подходы для поиска игровых структур, которые были разработаны и использованы в данной работе. Напомним, что игровая структура задается тройкой (G, D, s_0), где G — ориентированный граф, D — разбиение вершин на терминальные и подконтрольные игрокам, s_0 — номер начальной вершины. Также напомним, что любая вершина с исходящей степенью ноль по определению является терминальной, а остальные вершины терминальными не являются и принадлежат каким-то игрокам. Базовая версия программы подразумевала подачу на вход одной игровой структуры и ее проверки на разрешимость. Данная версия была неэффективна, ведь она покрывала лишь очень малое количество игровых структур, а именно лишь те, которые были поданы ей на вход.

4.1 Классы PC и $\mathbf{C_n}$

Пусть $\mathbf{P}^+$ — множество ациклических графов, $\mathbf{C}$ —- множество неациклических графов, $\mathbf{P}$ — множество ациклических графов с одним истоком (вершина с входящей степенью 0). Можно заметить, что $\mathbf{P} \subset \mathbf{P}^+$, а также что в любом графе $P \in \mathbf{P}$ из истока достижимы все вершины.

Перспективным классом графов является следующий класс. Рассмотрим некоторый ациклический граф $P \in \mathbf{P}$, из него будем проводить ребра в неациклический граф $C \in \mathbf{C}$. Таким образом, наш граф будет состоять из двух подграфов P и C, дополнительно соединенных ребрами, направленными из P в C. Обозначим множество таких графов за $\mathbf{PC}$. Таким образом мы задали граф G в игровой структуре. Будем считать, что начальная позиция s_0 равна истоку в подграфе P. Осталось задать разбиение вершин D. Для этого скажем, что каждая нетерминальная вершина может быть назначена любому игроку среди n для какого-то параметра n. Осталось заметить, что данный класс задает всевозможные игровые структуры. Давайте для любой пары (G, s_0) построим эквивалентный граф, принадлежащий классу $\mathbf{PC}$. Для

этого выберем $C = G$, в качестве P можно взять граф из одной вершины, между P и C провести ровно одно ребро в вершину $s_0 \in C$.

Определим класс $\mathbf{C_n}$. Возьмем ориентированный цикл длины n, также добавим n вершин, обозначим вершины на цикле за $c_1, c_2, \ldots, c_n$, остальные вершины обозначим за $a_1, a_2, \ldots, a_n$. Дополнительно к ребрам на цикле добавим следующие ребра: $\forall i : e_i = (c_i, a_i)$. Вершины a_i будут терминалами, поэтому необходимо назначить вершины на цикле игрокам. Для этого выбиралось некоторое $x|n$, после чего вершина c_i назначалась игроку $(i \mod x) + 1$. Так получается граф $\mathbf{C_n}(x)$.

В качестве базовых кандидатов подграфа C выбирались графы из множества $\mathbf{C_n}$ для любого $n \leq 6$ ($n \leq 2$ для аддитивного класса). Такой выбор обосновывался тем, что предыдущие примеры для смежных классов игр были найдены на графах из класса $\mathbf{PC}$, где $C \in \mathbf{C_n}$. Для получения других кандидатов на роль C в циклы из $\mathbf{C_n}$ добавлялись хорды, также были попытки объединить несколько циклов в кактус, наложить циклы друг на друга, соединить циклы последовательно и параллельно, и так далее.

Определение 8 (Перспективные кандидаты)**.** Обозначим множество кандидатов на роль подграфа C, полученное перечисленными выше методами, как множество *перспективных* кандидатов.

Перспективные кандидаты были использованы в переборе в тех случаях, когда подграф C задавался вручную (классы $\mathbf{PC}$, $\mathbf{GPC}$, $\mathbf{RPC}$).

Для подграфа P нет каких-то специальных конструкций, в целом перебирались любые префиксы, добавлялись любые разумные ребра между P и C.

Мы могли бы перебрать перспективных кандидатов на роль C, после чего выбрать некоторое P и проверить полученный граф на разрешимость. Однако, большое количество различных P не позволяло это сделать. Также, в теории могло оказаться, что пример без PH не содержит перспективный подграф C. Поэтому необходимо было решить обе эти проблемы.

4.2 Класс GPC

Возьмем некоторый граф $G_0 = (V, E_0)$ из класса $\mathbf{P}^+\mathbf{C}$ — данный класс получен из классов $\mathbf{P}^+$ и $\mathbf{C}$ тем же способом, что и $\mathbf{PC}$. То есть отличие заключается в том, что теперь у графа G_0 не обязательно единственный исток. Построим некоторое разбиение D, учитывая уже известное разбиение для C, ведь C получено из множества перспективных кандидатов. Рассмотрим некоторое множество E_1 ребер на вершинах V. Построим граф $G = (V, E)$, где $E = E_0 \cup E_2$, $E_2 \subset E_1$. То есть в граф G_0 добавим некоторые ребра из E_1. Перебрав всевозможные подмножества $E_2 \subset E_1$ размеров меньше некоторого параметра, мы получили класс $\mathbf{GPC}(G_0, D, E_1)$. У данного класса существовала значительная проблема, заключащаяся в том, что на практике начальное приближение G_0 имело очень мало ребер, а множество E_1 имело слишком много ребер. Вследствие чего был предложен более эффективный способ перебора графов.

4.3 Класс RPC

Зафиксируем C из множества перспективных кандидатов, пусть y — число нетерминалов в C. Пусть x — желаемое число вершин в P. Будем генерировать подграф $P \in \mathbf{P}$ случайным образом, также будем генерировать ребра между P и C. Пронумеруем вершины от 1 до $x + y$, вершины P будут иметь номера $1, 2, \ldots, x$, нетерминальные вершины C будут иметь номера $x + 1, x + 2, \ldots, x + y$, нумерация терминалов в C нас не интересует.

Генерация P происходит следующим образом: каждой вершине $i \in [2; x]$ выберем независимо равновероятно случайного предка $j < i$, после чего добавим ребро (j, i). Далее, переберем все пары $(v, u) : v < u \leq x$ вершин на префиксе, между которыми не было добавлено ребро, и добавим его с некоторой вероятностью. Заметим, что граф все еще останется ацикличным, ведь ребра добавлялись только от меньших индексов к большим. На практике хорошо работала вероятность $\frac{1}{x+2}$.

Теперь необходимо добавить ребра из P в C, для этого также перебирались всевозможные пары v, u, где $v \in P$, $u \in C, u \notin V_{\mathbf{T}}$, после чего с вероятностью $\frac{1}{x+2}$ в граф добавлялось ребро (v, u). Тогда в среднем добавлялось порядка $\frac{xy}{x+2} \approx y$ ребер, причем в каждый нетерминал подграфа C входило примерно одно ребро.

Также в подграфе P случайным образом выбиралось не более двух вершин, для каждой из которых создавался свой терминал t, в который добавлялось ребро из выбранной вершины. Стоит отметить, что некоторые вершины в P могут не иметь исходящих ребер. Также как и раньше, будем считать их терминалами. Назначение нетерминальных вершин игрокам в подграфе P делалось случайным, число игроков являлось параметром.

Таким образом получается класс графов $\mathbf{RPC}(C, D_C, x, n)$. Данный класс был использован для получения результатов в классе терминальной функции стоимости, с помощью него были получены игры, в которых цикл является почти наихудшим исходом. Также были замечены закономерности в разрешимости терминального класса. Эти результаты описаны в соответствующем разделе.

4.4 Класс RPRC

Зафиксируем количество вершин в подграфах P и C, а также количество терминалов, эти числа обозначим за x, y, z соответственно. Будем генерировать подграф $P \in \mathbf{P}$ и ребра между P и C тем же методом, что и в классе **RPC**. Отличие состоит в том, что подграф C мы также будем генерировать.

Переберем всевозможные пары $v, u \in C : v \neq u$, добавим ребро (v, u) с вероятностью $\frac{1}{y+2}$. Тогда в среднем число ребер будет равно $\frac{y(y-1)}{y+2} \approx y$. В данный момент в подграфе C может не существовать терминалов. Чтобы это исправить, добавим z новых терминалов, обозначим это множество за T. Для каждой вершины из C с вероятностью $\frac{1}{y+1}$ решим, добавлять ли ребро из этой вершины в T. В случае положительного ответа, добавим ребро в случайную

вершину $u \in T$. Получается, что из каждой вершины подграфа C ведет не более одного ребра в T, причем это ребро присутствует с вероятностью $\frac{1}{y+1}$ и его конец выбирался случайно равновероятно среди всех вершин в T. Вершины графа будем назначать случайно равновероятно между игроками.

Данный класс обозначим за $\mathbf{RPRC}(x, y, z, n)$. Можно заметить, что на самом деле этот метод генерации эквивалентен $\mathbf{RPC}(C \cup T, D_C, x, n)$ для подграфа $C \cup T$, сгенерированного методом выше. Преимущество **RPRC** относительно **RPC** заключается, во-первых, в том, что теперь программе не нужны графы на вход, она их генерирует сама. А, во-вторых, в том, что теперь перебираются графы, в которых в один терминал может вести сразу несколько ребер из подграфа C. Данный класс был активно использован для перебора игр в классе аддитивной функции стоимости, поскольку в этом классе получение перспективных игровых структур для перебора являлось трудоемкой задачей.

4.5 Программное ускорение

Стоит отметить, что во всех рассмотренных методах генерации программа запускает проверку на независимых примерах. По этой причине было решено использовать параллельные вычисления, на практике программа запускала проверку примеров в семь потоков, что на практике давало ускорение не менее чем в пять раз. В данном случае число потоков было выбрано равно семи из-за вычислительной мощности используемого компьютера.

4.6 Результат

Данная часть программы заняла 24 килобайта.

5 Алгоритм поиска РН

Далее описаны алгоритмы, которые проверяют на разрешимость поданную на вход игровую структуру, в случае неразрешимости также приводят неразрешимую игру.

5.1 Алгоритм для аддитивной функции стоимости

Напомним, что в случае аддитивного класса нас интересует разрешимость игр двух игроков при условии $\forall i, e : h_i(e) > 0$. Множеством исходов являются всевозможные вершинно простые пути, начинающиеся в s_0 и заканчивающиеся в единственном терминале (обозначим его за t_0), а также зацикливающийся исход c. Каждому ребру соответствует пара чисел $e : (h_1(e), h_2(e))$ — стоимость ребра для первого и второго игрока соответственно, стоимость исхода $H_i(o)$ равна сумме стоимостей ребер на пути, который этот исход образует в случае, если $o \neq c$, иначе $H_i(c) = +\infty$. Так как игроки минимизируют стоимость исхода, c является наихудшим исходом для обоих игроков.

Пусть $|X_1| = n$, $|X_2| = m$ — количество стратегий первого и второго игрока, пронумеруем стратегии первого игрока числами на отрезке $[1; n]$, также пронумеруем стратегии второго игрока. Теперь любой профиль будет иметь вид $x = (a, b), a \in [1; n], b \in [1; m]$. Множество профилей $X = X_1 \times X_2$ можно представить как двумерную матрицу размера $n \times m$, где строке соответствует стратегия первого игрока, а столбцу соответствует стратегия второго игрока. Пусть $o_{i,j}$ — исход игры при профиле $x = (i, j)$. Пусть $A : a_{i,j} = H_1(f(o_{i,j})), B : b_{i,j} = H_2(f(o_{i,j}))$. Напомним, что f — игровая форма заданной игровой структуры. Матрицы A и B задают матрицу стоимостей для первого и второго игрока. В этой формулировке получаем, что профиль $x = (i, j)$ является РН тогда и только тогда, когда a_x является локальным минимумом по строке, а b_x является локальным минимумом по столбцу.

5.1.1 Метод случайного сэмплирования

Самым простым на первый взгляд алгоритмом является назначение случайной пары чисел каждому ребру, после чего задача сводится к поиску элемента, являющегося минимумом по строке в матрице A и минимумом по столбцу в матрице B. Построение матриц можно осуществить за $O(nm \cdot g)$, где g — количество ребер в графе. Тогда матожидание времени работы алгоритма сводится к подсчету вероятности, что случайная игра в этой игровой структуре неразрешима. На практике вероятность очень мала, поэтому данный метод неприменим на практике.

5.1.2 Метод линейного программирования

Подойдем к задаче с другой стороны. Пусть мы не хотим явно считать матрицы A и B, но вместе с тем хотим узнать критерий разрешимости игровой структуры. Как было сказано раннее, критерий состоит в том, что не существует строки i и столбца j, что элемент на их пересечении является минимумом по строке в A и минимумом по столбцу в B. Это эквивалентно тому, что если каждой строке i из A сопоставить клетку с минимумом в этой строке m_i^a, и каждому столбцу j из B сопоставить клетку с минимумом в этом столбце m_j^b, то множества m^a и m^b не должны пересекаться.

Осталось понять, как выразить критерий существования назначения стоимостей ребрам для заданных множеств m^a и m^b. В терминах линейного программирования это делается легко: так как каждый элемент матрицы соответствует исходу, то факт того, что m_i^a является минимумом в строке означает, что соответствующий исход стоит не больше чем любой другой исход в строке. Сопоставим каждому ребру переменную, получится g переменных. Отбросив исход c, каждый исход o задается суммой переменных $p(o)$, в сумму входят переменные тех ребер, которые есть на пути, который соответствует этому исходу. Тогда условие, что исход o_1 меньше исхода o_2 задается как $p(o_1) - p(o_2) < 0$. Из упомянутых фактов становится понятно, что в строках

матрицы A достаточно оставить уникальные по строкам исходы, аналогично для столбцов матрицы B. К сожалению, минимумов в строке может быть несколько, тогда перебор замедляется в экспоненциальное число раз. Чтобы это исправить, выдвенем гипотезу, что существует неразрешимая игра, в которой все исходы имеют различную стоимость для обоих игроков. Тогда в любой строке минимум единственен и условие переписывается в виде $p(x) - p(y) \leq -1$.

Итоговый алгоритм работает следующим образом: мы храним две системы линейных неравенств L_a и L_b, изначально пустые. Мы параллельно перебираем строки матрицы A и столбцы матрицы B, для удобства скажем, что сейчас мы рассматриваем строку i матрицы A. Мы перебираем элемент m_i^a, который будет минимумом в этой строке. Чтобы элемент мог стать минимумом, нужно проверить, что он не лежит в множестве m^b, а также что после добавления неравенств на минимум в строке i система L_a остается разрешимой. Если проверка удалась, мы обновляем множество m^a, добавляем неравенства на минимум в систему L_a и переходим на следующую глубину перебора. Данный алгоритм на практике отсекает много ветвей перебора, поскольку система часто становится неразрешимой.

5.1.3 Результат

Данная часть программы заняла 19 килобайт. Так как нам неизвестен пример без PH с положительными стоимостями, для сравнения скорости поиска примера мы использовали разрешимость систем не только для положительных чисел, но и для отрицательных. Как говорилось ранее, в этом классе игр мы посчитали класс **RPRC** самым эффективным. Так как в этом классе игр достаточно перебирать игровые структуры с единственным терминалом, а число игроков равно двум, генератор принимал только два параметра, то есть выглядел как $\mathbf{RPRC}(x, y, 1, 2)$. Для увеличения скорости поиска в нашей программе было ограничение, что проверялись только те игровые структуры, в которых количество профилей не превышает 150. С такими па-

раметрами программа находила пример без РН в произвольных стоимостях в среднем за 200 примеров. Всего за одну секунду программа проверяла порядка 700 примеров. Перебрав параметры x и y до шести каждый и включив программу на разумное время, мы пришли к выводу, что примера в данном классе нет. Использовать параметры больше шести не имело смысл из-за ограничения на количество профилей.

5.2 Алгоритм для терминальной функции стоимости

Перейдем к анализу игр с терминальной функцией стоимости. Напомним, что в этом случае нас интересует разрешимость игр трех и более игроков, где множество исходов задается как $O = V_{\mathbf{T}} \cup \{c\}$, где c означает зацикливающийся исход. Так как нас интересует только порядок на исходах, каждая функция $H_i : O \to [0; |V_{\mathbf{T}}|]$ задает перестановку чисел. Поскольку в этой формулировке H_i принято считать функцией платежа, то игроки максимизируют платеж исхода. Таким образом, исход с платежом 0 считается наихудшим, а исход с платежом $|V_{\mathbf{T}}|$ считается наилучшим. Мы хотим проверить разрешимость игр, в которых цикл является наихудшим исходом для всех игроков, то есть $\forall i : H_i(c) = 0$. Для этого мы пытаемся построить игру, в которой достигаются такие функции платежей. Так как найти такой пример довольно сложно (либо невозможно), для заданной игровой структуры мы ищем такие перестановки H_i, что платеж цикла для игроков $H_i(c)$ был бы как можно хуже (меньше) по некоторому критерию.

Профиль x является РН, если $\forall i, x_i^* : H_i(f(x)) \geq H_i(f(x_i^*))$, где x_i^* равно любому профилю, который отличается от x только стратегией игрока i, f — игровая форма. Соответственно, критерий неразрешимости получается следующий: $\forall x\ \exists i, x_i^* : H_i(f(x)) < H_i(f(x_i^*))$. Иными словами, для любого профиля x существует игрок i и профиль x_i^*, отличающийся от x в координате i такой, что этот профиль лучше для игрока i чем x. Запишем этот критерий в других терминах. Зафиксируем профиль x, выпишем все пары

$(i, f(x_i^*))$ в список l_x. Тогда в этом списке должна существовать пара (i, y) такая, что $H_i(f(x)) < H_i(y)$. Это условие зависит от $f(x)$, поэтому для каждого исхода o создадим список L_o, в который поместим все списки l_x такие, что $f(x) = o$. Теперь заметим, что если в L_o если два списка $l_1 \subset l_2$, то выполнение условия для l_1 влечет выполнение условия для l_2, значит список l_2 не участвует в критерии, удалим его из множества L_o. Теперь мы для каждого исхода o получили множество ограничений L_o, для каждого списка $l \in L_o$ должна быть хотя бы одна пара, которая лучше чем этот исход. Формально, критерий неразрешимости игровой структуры звучит так: $\forall o \in O : \forall l \in L_o : \exists (i, o_1) \in l : H_i(o) < H_i(o_1)$.

5.2.1 Метод полного перебора

Самым простым методом будет перебрать для каждого игрока его перестановку исходов H_i. Если обозначить число исходов за $t = |O|$, а время одной проверки за T, время работы этого алгоритма составит $O((t!)^n \cdot T)$. Этот алгоритм работает слишком долго для эффективного перебора, однако у него есть другое преимущество. Напомним, что $H(i, o) = H_i(o)$. Введем дополнительное понятие.

Определение 9 (Равномерное равновесие Нэша)**.** Профиль x является *равномерным равновесием Нэша(РРН)* для тройки $(G = (V, E), D, H)$, если для любой начальной позиции $s_0 \in V$ профиль x является РН в полученной игре.

Очевидно, РРН является более сильным условием, чем РН. Заметим следующее: так или иначе мы перебираем графы из класса **PC**. Пусть мы зафиксировали тройку (G, D, H), причем $G \in \mathbf{PC}$, поэтому $G = P + C$. Если в подграфе C есть РРН, то в G есть РН. Этот факт можно вывести методом обратной индукции по вершинам подграфа P. Таким образом, раз мы ищем функцию платежа H такую, что не существует РН в исходном графе, то нам подойдут лишь такие функции H, для которых нет РРН в подграфе C. Приведенный алгоритм сначала перебирает функцию платежа H, а

потом проверяет существование РН. Этот алгоритм легко модифицировать так, чтобы он проверял существование РРН.

Напомним, что в данном классе игр мы ищем пример без РН, где цикл является наихудшим исходом для всех игроков. Таким образом, нам подойдут лишь такие подграфы C, что существуют функция H такая, что $\forall i : H(i, c) = 0$, причем для данной функций H не существует РРН в C. Этот факт можно использовать для ручного поиска перспективных циклических подграфов. В данном случае нас интересует тот факт, что для $\mathbf{C_n}(n)$ при $n > 2$ существуют функции $H : H(i, c) = 0$, что полученная игра не содержит РРН. Отсюда был сделан вывод, что перебор стоит проводить в первую очередь на графах, которые содержат $\mathbf{C_n}(n)$.

5.2.2 Метод перебора с отсечениями

Далее будем считать, что H_i задает перестановку исходов $h^i_0, h^i_1, \ldots, h^i_{|V_\mathbf{T}|}$, причем $H_i(h^i_j) = j$.

Вместо того, чтобы перебирать перестановку H_i, будем строить ее инкрементально. А именно, в каждой вершине перебора у нас будут построены перестановки H_j для $j < i$, а также будет зафиксировано несколько максимальных элементов перестановки H_i, то есть будут известны элементы $h^i_{|V_\mathbf{T}|}, h^i_{|V_\mathbf{T}|-1}, \ldots$. Теперь необходимо перебрать очередной элемент перестановки.

Рассмотрим критерий отсутствия РН: $\forall a \in O : \forall l \in L_a : \exists (i, o_1) \in l : H_i(a) < H_i(o_1)$. Если для какого-то исхода a существует $l \in L_a$, что все пары в l хуже чем a, то a является РН. Будем делать следующее, для каждой пары $(a, l) : l \in L_a$, будем хранить счетчик $cnt(a, l)$ количества пар из списка l, которые гарантированно хуже исхода a. Если в какой-то момент это количество стало равно размеру списка $cnt(a, l) = |l|$, то эта ветка перебора не подходит, ведь критерий нарушился. Допустим сейчас мы ставим исход o, причем множество непоставленных исходов в H_i, исключаяя o, равно R. Необходимо выполнить $\forall r \in R, \forall (i, r) \in l \in L_o : cnt(o, l)$ += 1, параллельно

проверяя не нарушился ли критерий. Это решение отсекает некоторые ветки перебора, однако мы используем не всю известную информацию.

Пусть $r \in R$. Тогда для списков $l \in L_r$, таких что $(i, o) \in l$, пара (i, o) станет лучше исхода r, то есть этот список уже не нарушит критерий, назовем его неактивным. Тогда заметим, что возможно существует исход, который можно сейчас поставить, не увеличив счетчик на еще активных списках. Если такой исход существует, то его всегда выгодно поставить кроме случая, когда этот исход равен c (ведь мы хотим, чтобы c было поставлено в самом конце, то есть было наихудшим). Если такой исход существует, то эта эвристика отсекает все ветки перебора в данной вершине, кроме одной. Также стоит заметить замечательный факт: если мы добавим фиктивного игрока, которому не назначим никаких вершин, то скорость работы перебора почти не поменяется из-за этой эвристики.

5.2.3 Результат

Данная часть программы заняла 28 килобайт. Представленный выше метод перебора с отсечениями можно улучшить, чтобы он обязывал цикл быть наихудшим исходом, или, например, чтобы цикл был k-ым наихудшим с конца исходом для не более чем p игроков, а для остальных игроков цикл был бы худшим исходом. Формально, если обозначить множество игроков за I, то алгоритм можно модифицировать так, чтобы он искал функцию платежа со следующим условием: $I = A \sqcup B, |A| \leq p, \forall i \in A : H_i(c) \leq k, \forall i \in B : H_i(c) = 0$. С помощью этого метода были рассмотрены графы класса **RPC** для различных циклических подграфов размеров до шести, если не включать в подсчет терминальные вершины. Для графов с не более чем 8 вершинами, не включая терминальные вершины, перебор перебирает порядка 3000 примеров в секунду, чего было достаточно для эффективного перебора всевозможных графов. Так как нам не удалось найти примера, где цикл является наихудшим исходом, было решено найти наилучшие приближения такого примера, то есть такую функцию H, что цикл является как можно

более плохим исходом для каждого игрока.

6 Примеры без РН для терминальной функции стоимости

Определение 10 (Цикличность игры)**.** Рассмотрим некоторую игру (G, D, H, s_0) без РН. Выпишем последовательность длины n: $(H_1(c), H_2(c), \ldots, H_n(c))$, упорядочим ее по возрастанию, удалив при этом нули. Полученную последовательность назовем *цикличностью игры.*

В идеале мы бы хотели найти игру с пустой цикличностью, однако ее может не существовать, поэтому давайте найдем некоторую наименьшую цикличность по всем играм класса. Заметим, что для одной игровой структуры может существовать несколько функций H без РН, поэтому игровой структуре в соответствие будем ставить множество из всех ее цикличностей. Обозначим это множество за $Seq(G, D, s_0)$.

Необходимо придумать какой-то способ сравнения цикличностей чтобы найти пример, максимально приближающийся к пустой цикличности.

Определение 11 (Доминация последовательности)**.** Пусть даны две упорядоченные по возрастанию последовательности (цикличности) a и b длины n и m. Последовательность a *доминируется* последовательностью b ($a \leq b$), если $n \leq m$ и $\forall i \in [1; n] : a_{n+1-i} \leq b_{m+1-i}$.

Обозначим за $S(G, D, s_0)$ множество минимальных цикличностей в $Seq(G, D, s_0)$ с точки зрения доминации (то есть таких, меньше которых нет). Тогда если структура (G, D, s_0) неразрешима для случая, когда цикл является наихудшим исходом, то $S(G, D, s_0)$ содержит ровно один элемент — пустую цикличность.

Гипотеза. Пусть (G, D, s_0) — неразрешимая игровая структура, $A = [2, 2]$ — последовательность. Для любой цикличности $B \in S(G, D, s_0)$ выполнено следующее: B доминирует A.

Пример, в котором получена оценка $[2, 2]$ был построен в [3], поэтому нижняя оценка на цикличность достигается. Данная гипотеза была проверена для всевозможных графов без петель размера до 10. Стоит также отметить, что эта гипотеза является более сильным утверждением, чем неразрешимость класса игр с терминальной функцией стоимости.

Теперь проанализируем свойства $S(G, D, s_0)$ в зависимости от игровой структуры. Пусть W_n — множество всевозможных игровых структур (G, D, s_0), в которых $G \in \mathbf{PC}, G = P + \mathbf{C_n}(n)$. Пусть X_n — множество минимальных цикличностей в объединении $\bigcup\limits_{Q \in W_n} S(Q)$. По результатам работы программы было выяснено, что при росте n множество X_n возрастает по определенным критериям, таким как длина минимальной последовательности, минимальный максимум среди последовательностей, и т.д.

Стоит отметить, что в игровых структурах, описанных выше, ребра из P проводились только в вершины цикла $\mathbf{C_n}(n)$, но не проводились в терминалы этого подграфа. Если разрешить проводить ребра в терминалы циклического подграфа, то существует пример с $S(P + \mathbf{C_3}(3), D, s_0) = \{[2, 2]\}$, который мы, однако, приводить не будем. Отметим лишь, что структура этого примера аналогична структуре примера для $\mathbf{C_2}(2)$ с тем отличием, что вместо дополнительного терминала для подграфа P используется лишний терминал в подграфе $\mathbf{C_3}(3)$.

Приведем полученные множества X_n для $n \leq 4$ в формате $n : X_n$.

- $2 : \{[2, 2]\}$
- $3 : \{[2, 3], [1, 2, 2]\}$
- $4 : \{[2, 4], [1, 2, 3], [1, 1, 2, 2]\}$

Для наглядности на рисунке 6.1 приведена игра, в которой достигается цикличность $[2, 3] \in X_3$.

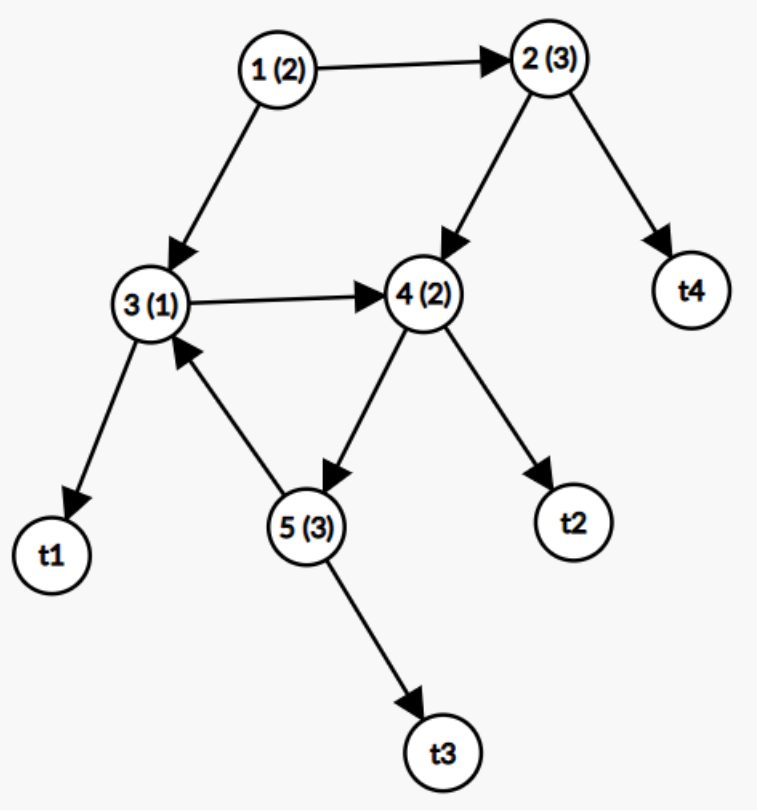

Рис. 6.1: Игра с цикличностью $[2, 3]$

Вершины $\{\texttt{t1}, \texttt{t2}, \texttt{t3}, \texttt{t4}\}$ являются терминальными, остальные вершины задаются как `i (p)`, где `i` равно номеру вершины, `(p)` равно номеру игрока, которому назначена вершина `i`. Вершина 1 является стартовой, перестановки H_i представлены ниже.

- $H_1 = [\texttt{t3}, \texttt{t1}, \texttt{c}, \texttt{t4}, \texttt{t2}]$
- $H_2 = [\texttt{t2}, \texttt{t1}, \texttt{t4}, \texttt{c}, \texttt{t3}]$
- $H_3 = [\texttt{c}, \texttt{t3}, \texttt{t4}, \texttt{t1}, \texttt{t2}]$

Отметим, что время работы программы, потраченное на поиск примера без PH зависело от того, с какими ограничениями искалась цикличность (речь про количество ненулевых значений в последовательности и максимум в последовательности). Причем было получено, что если искать примеры в правильном классе игр (W_n), то для любых достижимых ограничений эти примеры находились довольно быстро, в течение 5 секунд.

7 Заключение

Основным результатом работы является полученная программа проверки разрешимости игровых структур с терминальной функцией стоимости, а также доказательство разрешимости по Нэшу игр этого класса с циклом как худшим исходом для всех игроков, в которых количество нетерминальных вершин не превосходит десяти. Было получено более сильное утверждение, что для любой неразрешимой игры, ее цикличность доминирует последовательность $[2, 2]$. Полученная закономерность на последовательность X_n также может свидетельствовать о том, что упомянутый выше класс разрешим по Нэшу.

Также была получена программа для проверки разрешимости в игровых структурах с аддитивной функцией стоимости, и аналогичное доказательство разрешимости по Нэшу игр двух игроков с положительными стоимостями, в которых количество вершин не превосходит семи.

Исходные коды программ доступны в репозитории Github.

Список литературы

1. J. Nash, Equilibrium points in n-person games, Proc. Natl. Acad. Sci. 36 (1) (1950) 48–49.

2. H. Kuhn, Extensive games, Proc. Natl. Acad. Sci. 36 (1950) 286–295.

3. E. Boros, V. Gurvich, M. Milanic, V. Oudalov, J. Vicic, A three-person deterministic graphical game without Nash equilibria (2018)

4. V. Gurvich, The solvability of positional games in pure strategies, USSR Comput. Math. Math. Phys. 15 (2) (1975) 74–87.

5. V. Gurvich, Equilibrium in pure strategies, Sov. Math. Doklady 38 (3) (1989) 597–602.

6. V. Gurvich, A four-person chess-like game without Nash equilibria in pure stationary strategies, Bus. Inform. 1 (31) (2015) 68–76

7. V. Gurvich, V. Oudalov, On Nash-solvability in pure stationary strategies of the deterministic n-person games with perfect information and mean or total effective cost, Discrete Appl. Math. 167 (2014) 131–143.

8. E. Boros, K. Elbassioni, V. Gurvich, K. Makino, On Nash equilibria and improvement cycles in pure positional strategies for Chess-like and Backgammon-like n-person games, Discrete Math. 312 (4) (2012) 772–788.

9. E. Boros, V. Gurvich, On Nash-solvability in pure strategies of finite games with perfect information which may have cycles, RUTCOR Research Report, RRR-60-2001, Rutgers University; Math. Soc. Sciences 46 (2003) 207–241