ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК 519.833.2

КУРСОВАЯ РАБОТА

Командный исследовательский проект на тему:

Позиционные игры: равновесия Нэша; Комбинаторные игры: функции Гранди и Смита

Выполнили студенты:

группы #БПМИ223, 2 курса	Бутырин Богдан Георгиевич
группы #БПМИ221, 2 курса	Луценко Антон Игоревич
группы #БПМИ223, 2 курса	Пескин Максим Павлович

Проверен руководителем проекта:

Гурвич Владимир Александрович, PhD
Ведущий научный сотрудник
Международная лаборатория теоретической информатики ФКН ВШЭ

Москва 2024

Содержание

1 Аннотация

Рассмотрим конечные детерминированные игры с n игроками. Данные игры могут быть смоделированы конечными ориентированными графами, обладающими терминальными вершинами (то есть стоками), которые соответствуют концу игры. Вершины данного графа, не являющиеся терминалами, обозначают позиции игры (будем считать, что в каждой вершине вписано число от 1 до n – номер игрока, который обязан сделать ход в данной позиции). При этом граф игры может содержать циклы, то есть игра может длиться бесконечно. Рассмотрим конденсацию графа игры. Будем считать, что каждая из двусвязных компонент, содержащая хотя бы две вершины, образует свой исход (в таком случае этот исход соответствует некоторой бесконечной партии). Далее будем называть исход, соответствующий одной из ранее описанных двусвязных компонент, *бесконечным*.

Обозначим все бесконечные исходы за $V_C = \{c_1, \ldots, c_r\}$, а все терминалы – за $V_T = \{a_1, \ldots, a_p\}$. Наша цель – исследовать существования равновесия по Нэшу при условии, что каждый игрок выбирает для себя фиксированную стратегию и не меняет ее во время игры. Оказывается, что при $n = 2$ равновесие по Нэшу обязательно существует, однако при $n > 2$ это уже может быть неверно. Сформулируем следующие условия:

(C) Для любого из n игроков каждый из бесконечных исходов хуже каждого терминального исхода.

(C22) Будем говорить, что цикл C является *сильным* для игрока k, если для игрока k цикл C лучше некоторых двух терминалов. Существует не более одной пары цикл-игрок (C, k) такой что цикл C сильный для игрока k.

В статье [2] была сформулирована следующая гипотеза:

Гипотеза. ***(Catch 22)*** *Из условия* ***(C22)*** *следует существование равновесия по Нэшу. Иными словами, в любой игре без равновесия по Нэшу найдется хотя бы две пары цикл-игрок (C, k) таких что цикл C сильный для игрока k.*

Данную гипотезу мы и хотим опровергнуть в нашей работе, используя программный код, который сгенерировал бы пример игры, в которой не существует равновесия по Нэшу, но при этом выполнено либо условие **(C)**, либо условие **(C22)**.

2 Ключевые слова

Теория игр, равновесие по Нэшу, контрпример, ориентированный граф, сильно связные компоненты

3 Введение

Детерминированная игра на графе для нескольких игроков задается конечным ориентированным графом $G = (V, E)$, вершины которого разбиты на $n + 1$ непересекающихся подмножеств

$$V = V_1 \sqcup V_2 \sqcup ... \sqcup V_n \sqcup V_T$$

где вершина $v \in V_i$ $(1 \leqslant i \leqslant n)$ считается *позицией*, контролируемой игроком с номером $i \in I = \{1, 2, ..., n\}$, а вершина $a \in V_T = \{a_1, a_2, ..., a_p\}$ называется *терминальной позицией* (терминалом), причем из такой вершины не выходит никаких ребер.

Ребро $(v, v') \in E$ называется *ходом* из позиции v в позицию v'. Мы также фиксируем стартовую позицию $v_0 \in V \setminus V_T$, в которую не входит ни одного ребра.

Стратегией игрока $i \in I$ называется функция s^i, которая сопоставляет ход (v, v') каждой вершине $v \in V_i$.

Зафиксировав для каждого из игроков стратегию, имеем набор стратегий (такой набор называется *профилем*) $s = (s^1, ..., s^n)$. В детерминированной игре на графе для нескольких игроков, чтобы задать профиль, мы должны для каждой вершины $v \in V \setminus V_T$, задать ход (v, v'). Также профиль можно рассматривать как подмножество ребер $s \subset E$. Очевидно, что в подграфе $G_s = (V, s)$, соответствующему профилю s, существует единственный путь, начинающийся в стартовой вершине v_0, который либо заканчивается в терминальной вершине, либо зацикливается. Иными словами, профиль стратегий s задает игру, которая начинается в v_0, заканчивается в терминальной вершине или образует «лассо», которое в качестве префикса содержит (возможно, пустой) путь, к которому присоединен некоторый цикл.

Для простоты мы предполагаем, что все циклы, лежащие в одной компоненте сильной связности, эквивалентны. Тогда всевозможные исходы образуют множество

$$A = \{a_1, ..., a_p, c_1, ..., c_q\}$$

где $a_1, ..., a_p$ – исходы, соответствующие терминалам, а $c_1, ..., c_q$ – исходы, соответствующие компонентам сильной связности.

Для каждого игрока мы задаем перестановку исходов, от худшего к лучшему для данного игрока. Действительно, данная перестановка будет задавать предпочтения каждого из игроков, но при этом нам не нужно знать конкретный численный выигрыш игроков при реализации того или иного исхода.

Пусть $s = (s^1, ..., s^n)$ – произвольный профиль, $i_0 \in I$ – некоторый игрок. Тогда можно

считать, что $s = (s^{i_0}, s^{-i_0})$, где s^{-i_0} есть набор стратегий оставшихся игроков. Будем говорить, что игрок i_0 может *улучшить свой исход* на профиле s, если существует такая стратегия $\hat{s}^{i_0}$, что для игрока i_0 исход профиля $(\hat{s}^{i_0}, s^{-i_0})$ лучше исхода профиля (s^{i_0}, s^{-i_0}).

Профиль s называется *равновесием Нэша*, если ни один из игроков не может улучшить свой исход на этом профиле.

Нас интересуют графы игр, для которых возможно задать такие предпочтения для каждого из игроков, что никакой из профилей стратегий не будет равновесием Нэша. В статье [1] был приведен пример подобной игры для трех игроков, не обладающей равновесием Нэша. Данный пример был взят нами за основу и является основным тестом при дальнейшей разработке компьютерной программы, которая бы по заданному графу игры (данная структура содержит вершины, ребра и информацию о том, в какой вершине какому игроку необходимо сделать ход) генерировала искомую перестановку предпочтений для всех игроков.

В статье [2] были сформулированы следующие условия на граф игры:

(C) Для любого из n игроков каждый из бесконечных исходов хуже каждого терминального исхода.

(C22) Будем говорить, что цикл C является *сильным* для игрока k, если для игрока k цикл C лучше некоторых двух терминалов. Существует не более одной пары цикл-игрок (C, k) такой что цикл C сильный для игрока k.

В статье [2] была сформулирована следующая гипотеза:

Гипотеза. ***(Catch 22)*** *Из условия* ***(C22)*** *следует существование равновесия по Нэшу. Другими словами, в любой игре без равновесия по Нэшу найдется хотя бы две пары цикл-игрок (C, k) таких что цикл C сильный для игрока k.*

Цель нашего проекта – опровергнуть написанную выше гипотезу, используя программный код, который сгенерировал бы пример игры, в которой не существует равновесия по Нэшу, но при этом выполнено либо условие **(C22)**, либо условие **(C)** – видно, что условие **(C)** сильнее условия **(C22)**.

Распределение ролей в проекте:

1 **Антон Луценко:** Архитектура компьютерной программы, реализация функционала компьютерной программы

2 **Максим Пескин:** Реализация функционала компьютерной программы

3 **Богдан Бутырин:** Реализация функционала компьютерной программы, написание независимой программы для проверки примера.

4 Обзор Литературы

Равновесие Нэша известно в теории игр уже довольно давно и применяется в различных аспектах нашей жизни. Поэтому некоторый интерес представляют игры, в которых равновесие Нэша отсутствует. Данная работа нацелена на нахождение небольших игр без равновесия Нэша, удовлетворяющих некоторым дополнительным условиям. Так, впервые такая игра без Равновесия для четырех игроков была описана в статье [3]. Улучшением данной статьи стала другая статья [1], содержащая пример игры для трех игроков, не обладающей равновесием Нэша. Данный пример был взят нами за основу и являлся основным тестом при разработке программы.

Однако наша работа заключается не просто в поиске детерминированных игр, не имеющих равновесия Нэша, а игр, удовлетворяющих некоторому свойству, а именно игр, которые опровергают гипотезу **Catch 22**, которая была сформулирована в статье [2]. Гипотеза гласит, что в любой игре, в которой нет равновесия Нэша найдется хотя бы два игрока, для которых циклическое повторение ходов будет лучше хотя бы двух терминальных исходов. Также в статье описана более слабая гипотеза, опровергнув которую можно опровергнуть и **Catch 22**, а именно, что в любой игре, в которой нет равновесия Нэша найдется игрок, для которого найдется терминал, в который он не захочет попадать, даже если это приведет к тому, что игра никогда не закончится. Наша работа опроверает одну из гипотез.

5 Главы

Со всей компьютерной программой, о которой пойдет речь в данной части курсовой работы, Вы можете ознакомиться в нашем репозитории на GitHub.

5.1 Общая идея

Для поиска контрпримера была реализована компьютерная программа, которая на вход получает только сам граф игры (только структуру графа без предпочтений и расстановок игроков). Далее программа перебирает все возможные расстановки P игроков в позиции (вершины) графа, а также терминалы, которые из граф а имеет смысл удалить (подробности см. в главе **5.2**).

Для каждой расстановки игроков формируется КНФ (см. главу **5.3**) и решается с помощью `CP-SAT Solver` из библиотеки `Google OR Tools`. Любое решение сформированной КНФ позволяет восстановить предпочтения игроков для текущей расстановки P, при которых **гипотеза C22** была бы неверна.

Также в самом начале граф игры обрабатывается алгоритмом поиска компонент сильной двусвязности для выявления всех нетривиальных (содержащих хотя бы 2 вершины) компонент сильной двусвязности. Данные компоненты соответствуют бесконечным исходам.

5.2 Перебор

Чтобы упростить поиск контрпримера, перебираются только скелеты графов, в которых из каждой вершины, кроме стартовой, выходит терминал.

Несмотря на то, что дополнительные терминалы не могут привести к появлению равновесия Нэша, лишние терминалы могут быть хуже циклов для некоторых игроков, что может привести к потере контрпримера. Поэтому было принято решение дополнительно перебирать все возможные комбинации включения/исключения терминалов.

После этого, на каждом из $2^n - 1$ (n – число нетерминальных вершин) графов перебираются все возможные соответствия вершинам игроков, которые задают какой игрок какую вершину контролирует. Затем на полученных графах запускается проверка на отсутствие равновесия Нэша и **гипотезу C22**. Такие переборы занимают довольно много времени даже на небольших графах. Поэтому все необходимые вычисления нужно выполнять параллельно, что позволяет уменьшить время работы в несколько раз.

5.3 Формирование КНФ

В данной главе будет рассмотрено, как устроена КНФ, которая подается на вход SAT Solver.

5.3.1 Переменные в КНФ

Данная КНФ содержит в себе $p \cdot \frac{n(n-1)}{2}$ (n — количество исходов, p — количество игроков) переменных X_{ij}^k. Равенство данной переменной 1 означает, что игрок k предпочитает исходу i исход j.

5.3.2 Дизъюнкты в КНФ

Теперь зададим условия на предложенные переменные. Во-первых, если зафиксировать игрока k, то X_{ij}^k должны задавать отношение строго порядка, так как задают предпочтения игрока. Для этого мы добавляем следующие условия:

1 Если мы захотим обратиться к переменой X_{ii}^k, то будет возвращаться тождественно ложная переменная (антирефлексивность)

2 Как можно заметить, переменных всего $p \cdot \frac{n(n-1)}{2}$, так как мы будем хранить лишь те X_{ij}^k, у которых $i < j$, что позволит нам при попытке обращения к X_{ji}^k возвращать $\overline{X_{ij}^k}$ (антисимметричность)

3 Последнее свойство, которое нам нужно наложить на X_{ij}^k, это транзитивность. Для этого для каждого игрока (пусть он имеет номер k) мы переберем все тройки (i, j, l) и добавим дизъюнкт $\left(\overline{X_{ji}^k} \vee \overline{X_{lj}^k} \vee \overline{X_{il}^k}\right)$ (с учетом правила 2), который запрещает ситуацию, при которой i лучше чем j , j лучше чем l, а l лучше чем i, так как данный дизъюнкт должен равняться 1, а это происходит тогда и только тогда, когда хотя бы одна переменная равна 1, а значит какое-то из перечисленных условий не верно. Также, нам надо запретить обратный цикл, то есть мы добавляем еще и дизъюнкт $\left(\overline{X_{ij}^k} \vee \overline{X_{jl}^k} \vee \overline{X_{li}^k}\right)$

Первые два условия достигаются за счет функции `get_val`, с помощью которой мы обращаемся к объектам, обозначающие наши переменные.

Далее, нам надо связать наши переменные и игру, в которой мы пытаемся найти набор предпочтений. А именно, равновесия Нэша в игре нет тогда и только тогда, когда для любой стратегии найдется игрок, который может ее поменять так, чтобы улучшить исход в свою пользу.

Тогда, переберем все стратегии. Зафиксируем одну и назовем ее исход t. Тогда для данной стратегии переберем все стратегии, которые могут быть получены из нее ровно одним игроком (переберем игрока и все стратегии которые он может получить, поменяв исходную). Пусть исходы данных стратегий — $t_1, t_2, \ldots, t_m$, а игроки, поменявшие исходную стратегию так, чтобы получились соответствующие исходы — $k_1, k_2, \ldots, k_m$. Тогда добавим в КНФ дизъюнкт $\left(X_{t_1 t}^{k_1} \vee X_{t_2 t}^{k_2} \vee \ldots \vee X_{t_m t}^{k_m}\right)$, который выполнен тогда и только тогда, когда данная стратегия не является равновесием Нэша.

5.4 Контрпример к гипотезе C22

5.4.1 Описание контрпримера

С помощью программного перебора расположения терминальных вершин, распределения игроков между вершинами графа и последующего составления КНФ был найден контрпример. В вершинах графа расставлены числа от 1 до 3 – номер игрока, который ходит в данной вершине. Вершина, выделенная синим, является стартовой позицией игры, а вершины, выделенные красным, являются терминалами. Помимо двух терминальных исходов, есть еще три бесконечных исхода – циклы длины 2. В результате перебора и решения КНФ также были найдены предпочтения для каждого игрока – число X_N^p обозначает выигрыш игрока номер p в случае исхода **TN**:

$$X_1^1 < X_5^1 < X_2^1 < X_3^1 < X_4^1$$

$$X_4^2 < X_1^2 < X_2^2 < X_5^2 < X_3^2$$

$$X_3^3 < X_1^3 < X_4^3 < X_2^3 < X_5^3$$

Из неравенств на предпочтения легко видеть, что только цикл **T3** является лучше двух терминалов для какого-то из игроков. Таким образом, условие **C22** выполнено. Однако **гипотеза C22** неверна – у приведенного примера нет равновесия по Нэшу.

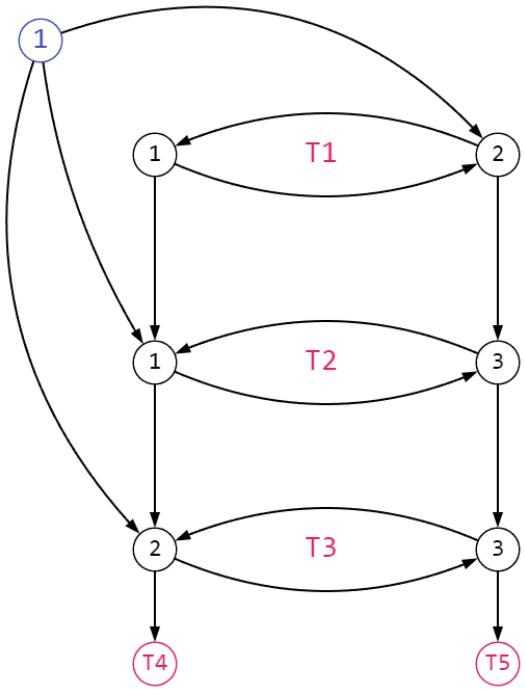


Рис. 5.1: Контрпример к C22

5.4.2 Независимая проверка контрпримера

Корректность примера была проверена следующей компьютерной программой:

```
#include <fstream>
#include <vector>
#include <iostream>

using namespace std;

class Checker {
private:
    int vertices_count;          // Number of vertices in the given graph
    int players_count;           // Number of players, currently only 3
    int terminals_count;         // Number of terminal components
    int starting_vertex;         // The vertex from which the game starts
    vector<vector<int>> graph;   // Given graph
    vector<int> vertex_component; // vertex_component[v] is number of component
        of vertex v
    vector<vector<vector<bool>>>
        player_preference;    // player_preference[p][x][y] = 1 if and only if p
```

```
            prefers terminal x over terminal y
    vector<int> vertex_player; // vertex_player[v] is number of player that
        controls this vertex
    vector<vector<int>> vertices_by_players;            // Vertices grouped by
        players
    vector<vector<vector<pair<int, int>>>> possible_changes; // Array of all
        possible ways a strategy can change
    bool is_correct = true; // Controls whether has Nash equililbrium been found
        or not

public:
    // Function that initializes Checker from graphs/c22_contrexample
    void init() {
        std::ifstream in("graphs/c22_contrexample.txt");
        players_count = 3;
        ////////////////////////// Graph input //////////////////////////
        in >> vertices_count;
        in >> starting_vertex;
        --starting_vertex;
        int edges_count; // Number of edges in the graph
        in >> edges_count;
        graph.assign(vertices_count, {});
        for (int i = 0; i < edges_count; ++i) {
            int from, to;
            in >> from >> to;
            --from;
            --to;
            graph[from].push_back(to);
        }
        // If a vertex is a leaf we make it a loop
        for (int i = 0; i < vertices_count; ++i) {
            if (graph[i].empty()) {
                graph[i].push_back(i);
```

```
45              }
46          }
47          ////////////////////// Terminals info input ///////////////////////
48          in >> terminals_count;
49          vertex_component.assign(vertices_count, -1);
50          for (int i = 0; i < terminals_count; ++i) {
51              int vertices_in_terminal_count;
52              in >> vertices_in_terminal_count;
53              for (int j = 0; j < vertices_in_terminal_count; ++j) {
54                  int vertex;
55                  in >> vertex;
56                  --vertex;
57                  vertex_component[vertex] = i;
58              }
59          }
60          ////////////////// Setting up player preference //////////////////
61          player_preference.assign(players_count,
               vector<vector<bool>>(terminals_count, vector<bool>(terminals_count)));
62          for (int p = 0; p < players_count; ++p) {
63              vector<int> pr;
64              for (int i = 0; i < terminals_count; ++i) {
65                  int t;
66                  in >> t;
67                  --t;
68                  pr.push_back(t);
69              }
70              for (int i = 0; i + 1 < pr.size(); ++i) {
71                  for (int j = i + 1; j < pr.size(); ++j) {
72                      player_preference[p][pr[i]][pr[j]] = true;
73                      player_preference[p][pr[j]][pr[i]] = false;
74                  }
75              }
76          }
```

```
77          ////////////////////////// Players input //////////////////////////
78          vertex_player.assign(vertices_count, 0);
79          vertices_by_players.assign(players_count, vector<int>());
80          for (int i = 0; i < vertices_count; ++i) {
81              in >> vertex_player[i];
82              --vertex_player[i];
83              vertices_by_players[vertex_player[i]].push_back(i);
84          }
85          possible_changes.resize(players_count);
86          in.close();
87      }
88
89      // Plays strategy represeneted by the given vector from starting_vertex to
        the end
90      int find_strategy_outcome(const vector<int>& strategy) const {
91          int current = 0;
92          for (int i = 0; i < strategy.size() + 10; ++i) {
93              current = strategy[current];
94          }
95          return vertex_component[current];
96      }
97
98      // Checks whether the player can get a better outcome for himself by
        changing the given strategy
99      bool is_improvable_by_player(int player, const vector<int>& strategy) const {
100         int outcome = find_strategy_outcome(strategy);
101         for (const auto& change : possible_changes[player]) {
102             vector<int> new_strategy = strategy;
103             for (const auto &[vertex, neighbour] : change) {
104                 new_strategy[vertex] = neighbour;
105             }
106             int new_outcome = find_strategy_outcome(new_strategy);
107             if (player_preference[player][outcome][new_outcome]) {
```

```
108                 return true;
109             }
110         }
111         return false;
112     }
113 
114     // Checks that the strategy is not a nash equililbrium
115     void check_not_equilibrium(const vector<int>& strategy) {
116         for (int player = 0; player < players_count; ++player) {
117             if (is_improvable_by_player(player, strategy))
118                 return;
119         }
120         is_correct = false;
121         cout << endl;
122         for (const auto& element : strategy) {
123             cout << element + 1 << " ";
124         }
125         cout << endl;
126     }
127 
128     // Recursively generates all strategies
129     void strategies_generate(vector<int>& strategy, int current_vertex) {
130         if (current_vertex == strategy.size()) {
131             check_not_equilibrium(strategy);
132             return;
133         }
134         for (const auto& neighbour : graph[current_vertex]) {
135             strategy[current_vertex] = neighbour;
136             strategies_generate(strategy, current_vertex + 1);
137         }
138     }
139 
140     // Recursively generates all changes that the player can apply to a strategy
```

```
    and puts them in possible_changes vector
void generate_change(int player, vector<pair<int, int>>& change_prefix) {
    int current_vertex_number = change_prefix.size();
    if (current_vertex_number == vertices_by_players[player].size()) {
        possible_changes[player].push_back(change_prefix);
        return;
    }
    for (const auto& neighbour :
        graph[vertices_by_players[player][current_vertex_number]]) {
        change_prefix.emplace_back(vertices_by_players[player][current_vertex_numbe
            neighbour);
        generate_change(player, change_prefix);
        change_prefix.pop_back();
    }
}

// Fills possible_changes vector
void make_changes() {
    for (int player = 0; player < players_count; ++player) {
        vector<pair<int, int>> empty_prefix;
        generate_change(player, empty_prefix);
    }
}

// Checks that the example given in graphs/c22_contrexample does not have a
    Nash equililbrium
bool check() {
    init();
    make_changes();
    vector<int> empty_strategy(vertices_count);
    strategies_generate(empty_strategy, 0);
    return is_correct;
}
```

```
};

int main() {
    Checker checker;
    if (checker.check()) {
        cout << "OK.\n";
    } else {
        cout << "Something went wrong.\n";
    }
}
```

5.4.3 Описание входных данных для независимой проверки

Структура входных данных для проверки корректности графа для контпримера:

```
// data from the file "graphs/c22_contrexample.txt"
9   // the number of vertexes (including terminals)
1   // the start vertex
15  // the number of edges
1 2 // "x y" for the edge x --> y
1 4
1 7
2 3
2 5
3 2
3 4
4 5
4 7
5 4
5 6
6 7
6 8
7 6
7 9
```

```
// outcomes (terminals + cycles)
5 // number of outcomes
2 // number of vertexes in the outcome
2 3 // vertexes in the current outcome
2
4 5
2
6 7
1
8
1
9
// preferences
1 5 2 3 4 // the first player
4 1 2 5 3 // the second player
3 1 4 2 5 // the third player
1 2 1 1 3 3 2 1 1 // the i-th number is the player which controls the i-th
   vertex
```

5.4.4 Корректность независимой проверки

Предложенный код не содержит алгоритмов сложнее рекурсии, что позволяет легко доказать его корректность. Вспомогательные комментарии подробно объясняют, что делает каждый метод класса `Checker`, который и осуществляет проверку.

Поясним некоторые моменты в данном коде, которые могут все же показаться нетривиальными. Стратегию мы описываем вектором длины `vertices_count`, который для каждой вершины показывает, куда из этой вершины надо пойти. Если из какой-то вершины нет исходящего ребра (то есть эта вершина является терминалом), то для удобства добавляется петля в эту вершину. Это нужно для того, чтобы работала функция `find_strategy_outcome`, которая просто симулирует `vertices_count` $+ 10$ ходов игры. Очевидно, что после такого количества ходов игра окажется в конечном исходе, так как по принципу Дирихле мы попадем в какое-то состояние дважды, а, значит, мы попадем в цикл, содержащий это состояние (а при попадении в терминал мы в нем просто остаемся, так как добавили петлю).

Также стоит отметить несколько применений рекурсии в данном коде для реализации

перебора. То, что эти рекурсии действительно перебирают все что от них требуется несложно показать с помощью метода математической индукции. Рассмотрим, например функцию, `strategies_generate`. Покажем по индукцией по `current_vertex` (`current_vertex` $\leqslant$ `vertices_count`, что для каждого корректного набора $s_0, s_1, \ldots, s_{\texttt{current\_vertex}-1}$ (под корректностью s понимается, что для всех элементов s_v в графе существует ребро из v в s_v) `strategies_generate` будет вызвана с параметрами (`strategy`, `current_vertex`), причем первые `current_vertex` элементов `strategy` будут совпадать с набором s.

База: В первый раз функция вызывается с параметрами $(\{\underbrace{0, 0, 0, 0, \ldots, 0}_{\texttt{vertices\_count раз}}\}, 0)$. Тогда утверждение для `current_vertex` $= 0$ выполнено.

Переход: Пусть выполнено утверждение для `current_vertex`, тогда покажем его для `current_vertex` $+ 1$. Если `current_vertex` $=$ `vertices_count`, то программа выйдет из рекурсии в первой условный оператор `if`, поэтому случая, когда `current_vertex` $\geqslant$ `vertices_count` не бывает, и осталось рассмотреть случай `current_vertex` $<$ `vertices_count`.

Зафиксируем набор $s_0, s_1, \ldots s_{\texttt{current\_vertex}}$. Тогда, по предположению индукции, в какой-то момент функция была вызвана с `current_vertex` равным данному и первые элементы `strategy` совпадают с $s_0, s_1, \ldots s_{\texttt{current\_vertex}-1}$. Внутри данного вызова происходит перебор всевозможных соседей вершины `current_vertex`. Тогда, так как набор s корректен, то один из `neighbour` будет $s_{\texttt{current\_vertex}}$ и внутри данной итерации цикла `for` произойдет нужный рекурсивный вызов для набора s. В силу произвольности s переход верен.

Значит, по индукции показали, что в итоге все возможные стратегии будут перебраны, так как утверждение верно и для `current_vertex` $=$ `vertices_count`, а как уже было показано в данной ситуации выполнится условный оператор `if` и произойдет проверка данной стартегии.

Корректность доказана.

Для другой рекурсивной функции (`generate_change`) доказательство полностью аналогично (индукция по `change_prefix.size()`).

Программа сделала полный перебор и вывела `OK`, что значит, что наш пример корректен.

6 Заключение

В ходе нашего курсового проекта была спроектирована перспективная программа, которая, принимая на вход только лишь конкретный граф игры, делает

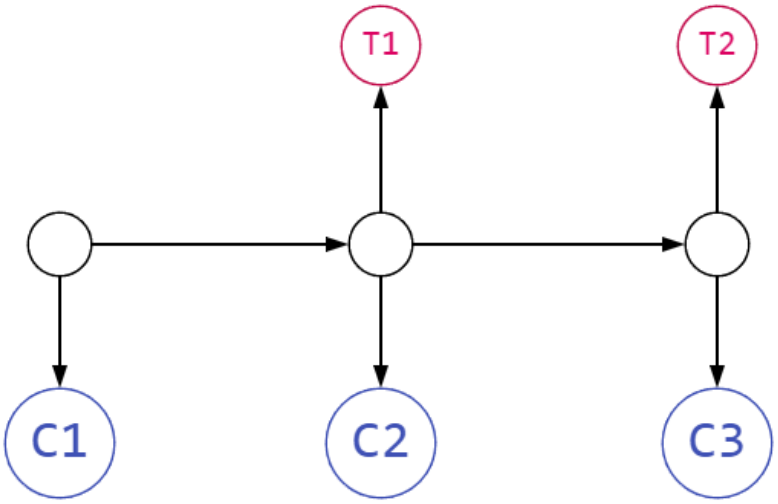
1 Эффективный перебор всевозможнных расстановок игроков в позиции (вершины графа)

2 Эффективный перебор возможного корректирования введенного графа: умное удаление терминалов, которые теоретически могут помешать построению необходимого примера

3 Для каждой расстановки эффективно решает задачу поиска примера предпочтений для построения полного примера игры без равновесия по Нэшу, который удовлетворял бы заранее определенным условиям, которые можно переформулировать на язык выполнимости некоторой КНФ

В результате проведенного исследования была опровергнута одна из известных в самом начале гипотез – **гипотеза C22**. Можно также пытаться сформулировать гипотезы, похожие на **C22** (например, **C21** – нет цикла, который хуже двух терминальных компонент, а также есть цикл, который хуже чем одна терминальная компонента, то тогда равновесие Нэша есть), которые, вероятно, также опровергаются нашими методами.

На наш взгляд, дальнейшие продвижения в данной области можно получить за счет анализа графов, созданных на основании следующей общей схемы:



Однако, естественно, количество циклов может больше трех. Как показала практика, возможно, существует пример где компоненты сильной связности **C1**, **C2**, ... являются циклами длины 2.

Список литературы

[1] Endre Boros, Vladimir Gurvich, Martin Milanič, Vladimir Oudalov и Jernej Vičič. “A three-person deterministic graphical game without Nash equilibria”. B: (2017). arXiv: 1610.07701 [cs.GT].

[2] Vladimir Gurvich. “On Nash-solvability of finite n-person deterministic graphical games; Catch 22”. B: (2021). arXiv: 2111.06278 [cs.GT].

[3] Vladimir Gurvich и Vladimir Oudalov. “A four-person chess-like game without Nash equilibria in pure stationary strategies”. B: (2014). arXiv: 1411.0349 [math.CO].