|  |  |  |
| --- | --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** | |
| ФАКУЛЬТЕТ | «Информатика и системы управления» (ИУ) | |
| КАФЕДРА | «Системы обработки информации и управления» (ИУ5) | |
| **РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  ***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:***  **«Исследование методов сжатия графов**  **на основе векторного представления»** | | |
| Студент группы ИУ5-44М | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. О. Лещев |
| Руководитель ВКР | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. Е. Гапанюк |
| Нормоконтролер | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. Н. Кротов |
|  | | |
| 2020 г. | | |

РЕФЕРАТ

Расчетно-пояснительная записка в размере 106 страниц, 63 рисунков, 1 таблицы, 76 использованных источников, 3 приложений.

СЖАТИЕ БЕЗ ПОТЕРЬ, СЖАТИЕ ГРАФОВ, МАШИННОЕ ОБУЧЕНИЕ, ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ.

Объектом исследования является метод сжатия графов на основе векторного представления.

Целью работы является исследование возможности применения векторного представления графов для сжатия их информационного представления и разработка соответствующего метода сжатия графа на основе векторного представления.

В работе предложен метод сжатия графов на основе векторного представления связей. Найдены подходящие структуры искусственных нейронных сетей для решения задач генерации векторного представления и классификации наличия связей, а также был создан подходящий формат данных для хранения графа.

Было проведено сравнение разработанного метода сжатия графов на основе векторного представления с традиционными способами хранения графов, такими как список связей и матрица смежности. Было выявлено, что предлагаемый метод сжатия графов применим для достаточно плотных и больших графов.

Разработка метода сжатия графов производилась на основании задания на выполнение выпускной квалификационной работы магистра, подписанного руководителем квалификационной работы и утверждённого заведующим кафедрой ИУ5 «Системы обработки информации и управления» Московского государственного технического университета имени Н.Э. Баумана.

СОДЕРЖАНИЕ

[РЕФЕРАТ 2](#_Toc44676749)

[ВВЕДЕНИЕ 6](#_Toc44676750)

[1 Критический анализ существующих подходов к решению задачи исследования 8](#_Toc44676751)

[1.1 Обзор и анализ публикаций по теме исследования 8](#_Toc44676752)

[1.1.1 Описание предметной области 8](#_Toc44676753)

[1.1.1.1 Графы 8](#_Toc44676754)

[1.1.1.2 Векторные представления 9](#_Toc44676755)

[1.1.2 Критический анализ публикаций по теме исследования 10](#_Toc44676756)

[1.1.2.1 Общие техники, применяемые в алгоритмах сжатия графов 10](#_Toc44676757)

[1.1.2.2 Публикации, посвященные сжатию графов без потерь 12](#_Toc44676758)

[1.1.2.3 Публикации, посвященные генерации векторных представлений для графов 13](#_Toc44676759)

[1.1.2.4 Нейронные сети, генерирующие векторные представления для узлов графов 14](#_Toc44676760)

[1.1.2.4.1 DeepWalk 14](#_Toc44676761)

[1.1.2.4.2 node2vec 15](#_Toc44676762)

[1.1.2.4.3 Structural Deep Network Embedding 16](#_Toc44676763)

[1.2 Определение направления исследования 17](#_Toc44676764)

[1.3 Формулировка цели и задачи 18](#_Toc44676765)

[2 Разработка метода сжатия графов 19](#_Toc44676766)

[2.1 Общие теоретические положения 19](#_Toc44676767)

[2.1.1 Графы 19](#_Toc44676768)

[2.1.2 Векторные представления 21](#_Toc44676769)

[2.2 Описание метода сжатия графов 21](#_Toc44676770)

[2.2.1 Общая структура метода сжатия графов 21](#_Toc44676771)

[2.2.2 Модель генерации векторных представлений 24](#_Toc44676772)

[2.2.3 Модель классификации наличия связи 27](#_Toc44676773)

[2.2.4 Процесс обучения моделей 29](#_Toc44676774)

[2.2.5 Процедура квантования модели классификации наличия связи 32](#_Toc44676775)

[2.2.6 Очистка бинарного представления модели сжатия графов 37](#_Toc44676776)

[2.2.7 Обработка потери точности 38](#_Toc44676777)

[2.3 Программная реализация метода сжатия графов 39](#_Toc44676778)

[2.3.1 Формат данных 39](#_Toc44676779)

[2.3.2 Построение матрицы смежности графа 43](#_Toc44676780)

[2.3.3 Модель генерации векторных представлений 45](#_Toc44676781)

[2.3.4 Модель классификации наличия связи 46](#_Toc44676782)

[2.3.5 Процесс обучения моделей 47](#_Toc44676783)

[2.3.6 Очистка бинарного представления модели 50](#_Toc44676784)

[2.3.7 Обработка потери точности 51](#_Toc44676785)

[2.4 Разработка модели для оценки границ применимости разработанного метода сжатия графов 53](#_Toc44676786)

[2.4.1 Составление формул для вычисления информационного объема графа в различных представлениях 54](#_Toc44676787)

[2.4.1.1 Параметры графа 54](#_Toc44676788)

[2.4.1.2 Информационное представление графа 56](#_Toc44676789)

[2.4.1.3 Представление графа в виде матрицы смежности 59](#_Toc44676790)

[2.4.1.4 Представление графа в виде списка рёбер 60](#_Toc44676791)

[2.4.1.5 Векторное представление связей графа 61](#_Toc44676792)

[2.4.2 Реализация формул для вычисления информационного объема графа в КЭСМИ 62](#_Toc44676793)

[2.5 Оценка границ применимости разработанного метода сжатия графов 87](#_Toc44676794)

[3 Анализ эффективности разработанного метода сжатия графа 92](#_Toc44676795)

[3.1 Исходный граф для сжатия 92](#_Toc44676796)

[3.2 Анализ результатов сжатия 93](#_Toc44676797)

[3.3 Апробация алгоритма на других графах 96](#_Toc44676798)

[ЗАКЛЮЧЕНИЕ 98](#_Toc44676799)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 99](#_Toc44676800)

[ПРИЛОЖЕНИЕ А Техническое задание 107](#_Toc44676801)

[ПРИЛОЖЕНИЕ Б Вывод модели оценки размеров графа в консоль 113](#_Toc44676802)

[ПРИЛОЖЕНИЕ В Графическая часть выпускной квалификационной работы 129](#_Toc44676803)

ВВЕДЕНИЕ

В настоящее время сжатию информации отводят довольно мало внимания. Объемы дискового пространства относительно его стоимости растут экспоненциально [1]. Исходя из этого разработчики многого прикладного программного обеспечения не задумываются о размере итоговой программы. В большинстве случаев они думают только об удобстве разработки, а потому подключают множество тяжелых библиотек ради каких-то незначительных функций [2] [3]. Нередко в больших библиотеках основной объем представляют собой данные, которые могут быть необходимы библиотеке. Похожая ситуация происходит на рынке компьютерных игр: за последние годы размер компьютерных игр вырос на порядок, и, судя по имеющемуся тренду, это не собирается прекращаться [4]. В последнем случае такая ситуация происходит преимущественно из-за размера текстур — то есть тоже данных.

Сжатие способно очень сильно сократить объем передаваемых данных [5]. Однако им зачастую не пользуются в связи с различными факторами:

* существуют ограничения на скорость извлечения данных;
* занимаемое дисковое пространство не критично;
* время разработки ограничено.

Одна из областей, где сжатие только начинают применять, это графы. Причем, насколько показывает исследование предметной области, используется сжатие с потерями, и эти алгоритмы применяются в основном для извлечения какой-то основополагающей структуры графа, оставляя лишь скелет этого графа, а не большую часть связей [6] [7]. В данной работе предлагается метод сжатия графов на основе векторного представления связей. Этот метод позволяет сохранить большую часть связей в графе и, возможно, получить лишние связи, которых не было изначально. В зависимости от области применения подобный результат может оказаться даже преимуществом, так как алгоритм открывает новые, ранее незамеченные связи, и появляется возможность для их исследования. С другой стороны, есть возможность предварительно проверить точность векторного представления связей в графе, и составить списки исключений в случае, когда необходимо сжатие без потерь.

Для создания предлагаемого метода сжатия графов была найдена подходящая структура нейронной сети, которая будет решать задачу генерации векторного представления и классифицировать наличие связей, а также был создан подходящий формат данных для хранения графа как в традиционном виде (в виде списка связей), так и в векторном представлении для удобства сравнения представлений графов.

Было проведено сравнение предлагаемого метода сжатия графов на основе векторного представления с традиционными способами хранения графов, такими как список связей и матрица смежности. Было выявлено, что предлагаемый метод сжатия графов применим для достаточно плотных и больших графов.

# Критический анализ существующих подходов к решению задачи исследования

## Обзор и анализ публикаций по теме исследования

### Описание предметной области

#### Графы

Граф  — это упорядоченная пара , где  — непустое множество вершин или узлов, а  — множество пар (в случае неориентированного графа — неупорядоченных) вершин, называемых ребрами или связями. Множества и  являются конечными множествами. Ребро и любая из его двух вершин называются инцидентными [8] [9].

В данной работе будут рассматриваться неориентированные графы, однако большинство представленных подходов можно легко адаптировать для ориентированных графов. Также не рассматриваются случаи, когда граф содержит циклы (также называемые петлями), то есть случаи, когда концы ребра являются одним и тем же узлом. Аналогично не рассматриваются случаи множественных связей между парой узлов и взвешенные графы (и любые другие графы, у которых с рёбрами связаны некоторые данные).

Планарный граф — граф, который можно изобразить на плоскости без пересечений рёбер не по вершинам [8].

В информатике используются следующие способы представления связей графа [8] [9]:

* матрица смежности — таблица, где столбцы и строки соответствуют вершинам графа, а в каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот);
* матрица инцидентности — таблица, где строки соответствуют вершинам графа, а столбцы соответствуют связям графа, в ячейку матрицы на пересечении строки  со столбцом  записывается в случае, если связь  выходит из вершины , в случае, если связь входит в вершину, и во всех остальных случаях;
* список смежности — список, где каждой вершине графа соответствует строка, в которой хранится список смежных вершин;
* список ребер — список, где каждому ребру графа соответствует строка, в которой хранятся две вершины, инцидентные ребру.

#### Векторные представления

Векторное представление — общее название для различных подходов к моделированию предметной области и обучению представлений, направленных на сопоставление объектам из некоторого словаря векторов из  для , значительно меньшего количества объектов в словаре. Векторное представление активно используется в обработке естественного языка. Для построения векторных представлений слов используют нейронные сети [10], методы снижения размерности в применении к матрицам совместных упоминаний слов [11] и явные представления, обучающиеся на контекстах упоминаний слов [12].

Существуют три основные идеи, как векторные представления используются в графах [13] [14] [15]:

* генерация векторных представлений для каждой связи графа;
* генерация векторных представлений для каждого узла графа;
* генерация векторных представлений для всего графа.

Чаще всего используется генерация векторных представлений для каждого узла графа с помощью специальных разновидностей генеративных нейронных сетей для сжатия данных о связях в графе в небольшие вектора и последующее извлечение этих данных с целью получения похожего, но не идентичного из-за специфики данного типа нейронных сетей представления связей в графе [16] [17] [18]. Работа этого вида алгоритмов зависит от следующих факторов:

* наличие данных, привязанных к узлу графа;
* зависимость связей от данных, привязанных к узлу графа;
* логичность топологии графа.

Таким образом векторное представление узлов графа может хранить информацию о связях этого узла.

### Критический анализ публикаций по теме исследования

В сжатии графов основным направлением является сжатие с потерями. Предполагается, что для изучения графов полезнее оставить какие-то наиболее важные узлы и связи, извлечь основную структуру графа, отбросить излишние вершины и рёбра [6] [7]. Данные алгоритмы выходят за рамки данной работы и рассматриваться не будут. Наиболее подробный обзор существующих публикаций на тему сжатия графов без потерь даёт статья «Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations» [19].

#### Общие техники, применяемые в алгоритмах сжатия графов

Многие работы используют одни и те же техники, которые применимы для любых графов и могут использоваться во многих алгоритмах. Так, для кодирования идентификаторов вершин (и любых других чисел) можно использовать формат величин переменной длины, также называемой VLQ или Varint. Это универсальный код, который использует произвольное число двоичных октетов (восьмибитных байтов) для представления произвольно большого целого числа. Формат величин переменной длины — это, по сути, представление беззнакового целого числа по основанию 128 с добавлением восьмого бита для обозначения продолжения байтов [20] [21]. В этом формате число занимает ровно столько байт, сколько необходимо для его представления по основанию 128, а длина последовательности байтов, обозначающих число, определяется по старшему биту этих байтов: если бит установлен, то байт не последний. Длину числа в байтах в данном формате можно определить следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Очевидна проблема такого формата величин переменной длины: он никак не рассматривает отрицательные числа. С точки зрения Protocol Buffers такие числа кодируются как очень большие беззнаковые числа, и таким образом отрицательные числа всегда занимают максимальное число байтов. В случае, когда отрицательные числа важны, можно использовать ZigZag-кодирование [21]:

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Таким образом, если известно, что отрицательных чисел много и среди них будут достаточно малые по модулю числа, можно использовать последовательно ZigZag-кодирование и формат величин переменной длины. Комбинация этих двух методов позволяет достаточно эффективно представлять числа любого знака при условии, что среди них встречается достаточно малых по модулю чисел.

Довольно популярный приём для алгоритмов, представляющих графы, — перенумерация вершин. Если идентификаторы вершин не являются началом натурального ряда [22], можно составить отображение из идентификаторов вершин в начало натурального ряда, и использовать получившиеся числа в качестве идентификаторов вершин при хранении графа. Данный приём позволяет максимально эффективно использовать формат величин переменной длины [19].

В случае, если для кодирования используются списки смежности, нередко можно применить кодирование с помощью ссылок, когда вместо повторения одних и тех же идентификаторов узлов в нескольких списках, эти идентификаторы можно привести лишь в первом списке, а в остальных сослаться на первый список [19] [23] [24]. Этот метод в случае огромных списков смежности может быть улучшен с помощью кодирования длин серий, когда вместо непосредственно нулей и единиц указаны длины последовательностей нулей и единиц [19] [25]. Другой способ упрощения кодирования подобных элементов — кодирование интервалов идентификаторов вершин [19]. В случае, если нужно кодировать не последовательные идентификаторы вершин, можно использовать кодирование пропусков: записывать не сами идентификаторы вершин, а разницу между идентификаторами вершин. В таком случае можно, опять же, получить меньший размер за счёт формата величин переменной длины, так как разницы между идентификаторами меньше, чем сами идентификаторы [19].

#### Публикации, посвященные сжатию графов без потерь

Множество статей предлагают приемы для кодирования графов из определенных предметных областей: графов отдельных участков Всемирной паутины, графов социальных сетей, графов биологических сетей (таких как сетей сборки генома, сетей регуляции генов, метаболических графов, протеиновых сетей и тому подобное), графов Resource Description Framework, графов компьютерных сетей, химических сетей, географических наборов данных и графов структуры сверхбольших интегральных микросхем [19]. Однако все они используют некоторые особенности кодируемых данных, и эти подходы не масштабируются на другие графы.

Также существуют алгоритмы для сжатия динамических графов, которые нужно редактировать со временем и выполнять к ним различные запросы. Данные алгоритмы применяются в графовых базах данных, таких как neo4j [26]. Примерами таких алгоритмов являются DEX [27] [28], -деревья [29] [30], ZipG [31], Succinct [32], GBASE [33] [34] и другие [19]. Проблема всех этих решений в том, что им приходится делать компромиссы для того, чтобы обеспечить возможность быстро изменять графы и выполнять запросы к ним, и им сложно достигнуть теоретических пределов сжатия графов.

Целый пласт статей посвящен теоретическим выкладкам, показывающим, какова нижняя граница информационного объема графа, на которые наложены определенные ограничения. Так, было показано, что связи планарного графа, состоящего из треугольников, можно представить с помощью битов [35]. Большинство статей рассматривают планарные графы, что довольно заметно ограничивает их применимость [19]. Если смотреть с точки оптимизации представления матрицы смежности, в худшем случае требуется битов [36].

Также есть работа, которая обсуждает возможность векторного представления связей, генерируемых для каждого узла. В данной работе показано, что для графов, представимых с помощью рассматриваемой техники, в худшем случае требуется битов [37].

#### Публикации, посвященные генерации векторных представлений для графов

Векторные представления для графов также являются довольно обширной темой. Существуют десятки различных алгоритмов генерации некоторых векторных представлений. Эти алгоритмы не заточены под конкретные применения, но предполагается, что после генерации векторные представления можно использовать для сжатия графов с потерями, визуализации, кластеризации, предсказания будущих связей, классификации. Так как рассматриваемые алгоритмы являются алгоритмами общего назначения, то нет возможности построить рейтинг сгенерированных векторных представлений, можно лишь проверить применимость этих представлений для каких-то определенных задач [38] [39] [40] [41].

Важно отметить, что задача предсказания будущих связей, упомянутая выше, отличается от задачи предсказания наличия связи по некоторым данным, рассматриваемой в данной выпускной квалификационной работе. В задаче предсказания будущих связей предполагается, что у каждого ребра графа есть момент времени , связанный с этим ребром, и тогда подграф содержит все рёбра, для которых . Предлагается по подграфу предсказать, какие связи будут содержаться в подграфе . В данной области также есть различные алгоритмы, которые с разных сторон подходят к этому вопросу [42]. Однако очевидно, что данная задача всё-таки заметно отличается от той, которую предлагается решать в данной работе, хотя бы потому что с рёбрами не должно быть связано никаких данных.

#### Нейронные сети, генерирующие векторные представления для узлов графов

Статья «Graph Embeddings — The Summary» предлагает хороший обзор алгоритмов генерации векторных представлений для каждого узла графа, основанных на нейронных сетях [15]. Данный подход кажется наиболее перспективным с точки зрения эффективности разрабатываемого метода сжатия графов, поскольку есть понятный механизм, как оптимизировать нейронные сети под решение заданной задачи. Далее будут рассмотрены основные методы генерации векторных представлений для каждого узла графа.

##### DeepWalk

Метод DeepWalk использует случайные пути из заданного узла графа для генерации некоторого векторного представления этого узла [43]. Этот алгоритм состоит из трёх шагов:

1. Генерация случайных путей.
2. Обучение модели skip-gram.
3. Вычисление векторных представлений.

В графическом виде эти шаги приведены на рисунке 1.

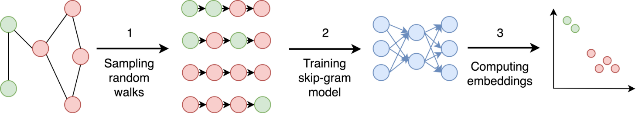


Рисунок 1 — Шаги работы метода DeepWalk

Проблема данного метода заключается в том, что используются случайные и довольно длинные пути, и связность с ближайшими узлами передается относительно плохо.

##### node2vec

Метод node2vec во многом похож на DeepWalk, но вводит два дополнительных параметра, которые регулируют пути, которые генерируются для заданного узла. Так, параметр  определяет штраф за переход в неисследованную часть графа, а параметр  определяет штраф за возврат в предыдущий узел [44]. Графическая иллюстрация этой системы приведена на рисунке 2.

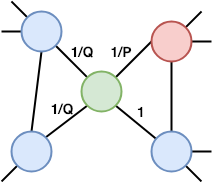


Рисунок 2 — Система весов метода node2vec при переходе из правого верхнего узла в центральный

Данный подход позволяет настраивать, в какой степени векторное представление передает микрокартину графа вокруг заданного узла, а в какой — макрокартину. Благодаря этому метод node2vec можно настраивать для различных задач.

##### Structural Deep Network Embedding

Метод Structural Deep Network Embedding предлагает принципиально иной подход к генерации векторных представлений. Его основной задачей как раз является передача непосредственных связей узла графа. Нейронная сеть представляет собой автокодировщик, который работает над одной строкой матрицы смежности, соответствующей определенному узлу. При этом в процессе обучения рассматриваются два узла одновременно, и для смежных узлов помимо точности представления ставится еще задача минимизации расстояния между генерируемыми векторными представлениями [45]. Графически эта идея представлена на рисунке 3.

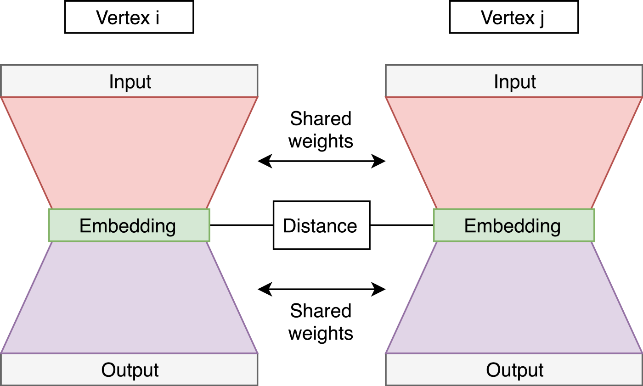


Рисунок 3 — Структура метода Structural Deep Network Embedding

Соответственно, после обучения нейронной сети достаточно взять сгенерированное внутреннее представление, подставить его в середину нейронной сети, и получить строку матрицы связности для заданного узла. Данный метод довольно неплохо подходит для поставленной задачи.

## Определение направления исследования

По результатам анализа публикаций были определены описанные ниже направления исследования.

Прежде всего необходимо определить алгоритм генерации векторных представлений и алгоритм предсказания наличия связи по сгенерированным векторным представлениям. Рассмотренные выше статьи предлагают как математические модели генерации векторных представлений, так и различные структуры нейронных сетей для решения данной задачи.

Необходимо определить входные данные, подаваемые на вход алгоритма генерации векторного представления. Анализ публикаций показывает, что распространены следующие варианты входных данных для генерации векторного представления:

* Идентификатор узла. В таком случае требуется подходящая система обучения модели, которая сможет подобрать подходящие коэффициенты модели генерации векторного представления.
* Различные пути из рассматриваемого узла. В таком случае модель учится учитывать не только соседние узлы, но и некоторую окрестность графа вокруг рассматриваемого узла.
* Строки матрицы смежности. Таким образом мы, фактически, предлагаем модели сжать эту строку в небольшой вектор. Данный подход наиболее удобен тем, что на его основе можно обучить автокодировщик и можно определить понятную функцию потерь.

Далее необходимо определить способ обучения выбранных алгоритмов генерации векторных представлений и предсказания наличия связи по сгенерированным векторным представлениям. Рассмотренные выше статьи практически не затрагивают вопрос обучения алгоритмов генерации векторного представления, просто предлагая модель, по которой алгоритм должен работать.

После разработки метода сжатия графов необходимо исследовать границы применимости разработанного метода, чтобы можно было по параметрам графа определить, имеет ли смысл применять разработанный метод.

## Формулировка цели и задачи

Целью данной выпускной квалификационной работы является разработка метода сжатия информационного представления графе на основе векторного представления связей.

В процессе выполнения данной выпускной квалификационной работы должны быть решены следующие задачи:

1. Проведен обзор и анализ публикаций по теме векторного представления вершин графа.
2. Проведен анализ существующих методов сжатия графов.
3. Описан разработанный метод сжатия графов.
4. Разработана программная реализация описанного метода сжатия графов.
5. Разработана модель для оценки эффективности разработанного метода.
6. Описан процесс моделирования.
7. Проанализированы результаты проведенного моделирования.

# Разработка метода сжатия графов

## Общие теоретические положения

### Графы

Граф  — это упорядоченная пара , где  — непустое множество вершин или узлов, а  — множество пар (в случае ориентированного графа — упорядоченных, в случае неориентированного графа — неупорядоченных) вершин, называемых ребрами или связями. Множества и  являются конечными множествами [8] [9].

Инцидентность — понятие, используемое в отношении ребра и вершины: если  — вершины, а  — соединяющее их ребро, тогда вершина  и ребро  инцидентны, вершина  и ребро  также являются инцидентными. Две вершины (или два ребра) инцидентными быть не могут. Для обозначения ближайших вершин (рёбер) используется понятие смежности.

Смежность — понятие, используемое в отношении двух рёбер либо двух вершин: два ребра, инцидентные одной вершине, называются смежными; две вершины, инцидентные одному ребру, также называются смежными.

Кратные рёбра — несколько рёбер, инцидентных одной и той же паре вершин.

Несвязный граф — граф, не содержащий рёбер.

Петля — ребро, начало и конец которого находятся в одной и той же вершине.

Простой граф — граф, в котором нет кратных рёбер и петель.

Полный граф — простой неориентированный граф, в котором каждая пара различных вершин смежна. Полный граф с  вершинами имеет рёбер.

Плотный граф — граф, в котором число рёбер близко к максимально возможному у полного графа с числом  вершин:

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

Разреженный граф — граф, имеющий малое число рёбер.

Плотность графа  для неориентированного простого графа с числом вершин  определяется как отношение числа его рёбер  к числу рёбер полного графа [46]:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

Максимальная плотность графа равна 1 (для полных графов), минимальная плотность графа равна 0 — для несвязанного графа [47].

В данной работе будут рассматриваться простые неориентированные графы, однако большинство представленных подходов можно легко адаптировать для простых ориентированных графов. Также в данной работе не рассматриваются взвешенные графы (и любые другие графы, у которых с рёбрами связаны некоторые данные).

Будут рассматриваться следующие способы представления связей графа [8] [9]:

* матрица смежности — таблица, где столбцы и строки соответствуют вершинам графа, а в каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот);
* список ребер — список, где каждому ребру графа соответствует строка, в которой хранятся две вершины, инцидентные ребру.

### Векторные представления

Векторное представление — общее название для различных подходов к моделированию предметной области и обучению представлений, направленных на сопоставление объектам из некоторого словаря векторов из  для , значительно меньшего количества объектов в словаре.

Существуют три основные идеи, как векторные представления используются в графах [13] [14] [15]:

* генерация векторных представлений для каждой связи графа;
* генерация векторных представлений для каждого узла графа;
* генерация векторных представлений для всего графа.

Чаще всего используется генерация векторных представлений для каждого узла графа с помощью специальных разновидностей генеративных нейронных сетей для сжатия данных о связях в графе в небольшие вектора и последующее извлечение этих данных с целью получения похожего, но не идентичного из-за специфики данного типа нейронных сетей представления связей в графе [16] [17] [18]. Работа этого вида алгоритмов зависит от следующих факторов:

* наличие данных, привязанных к узлу графа;
* зависимость связей от данных, привязанных к узлу графа;
* логичность топологии графа.

Таким образом векторное представление узлов графа может хранить информацию о связях этого узла.

## Описание метода сжатия графов

### Общая структура метода сжатия графов

Было решено выбрать следующую структуру метода сжатия графа. На вход предлагаемого метода сжатия графа подается граф в виде списка вершин и списка связей , где для каждой связи из списка связей выполняется следующее условие:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Из графа извлекается матрица смежности . Для этого создаётся пустой массив размерностью . Элементы главной диагонали приравниваются единице для того, чтобы на вход модели генерации векторных представлений подавались более консистентные данные:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

После этого для каждой связи из списка связей элементы матрицы смежности на соответствующих вершинам связи строке и столбце, а также симметричном элементе относительно главной диагонали приравниваются единице, чтобы отобразить наличие связи:

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

где — индекс элемента в списке вершин ,

— индекс элемента в списке вершин .

Таким образом получаем готовую матрицу смежности, строки которой можно использовать для подачи их в модель генерации векторных представлений.

Очевидно, что данную операцию может быть сложно провести в случае, если матрица смежности не помещается целиком в память. В таком случае можно генерировать строки матрицы смежности на лету, прямо в процессе обучения модели генерации векторного представления. Данный процесс, очевидно, займет больше времени, но в зависимости от данных может быть возможно кэширование данных и дедупликация строк матрицы смежности, то есть замена дублирующихся строк одной, таким образом возможно сэкономить память и при этом не тратить процессорное время на повторное вычисление одних и тех же данных.

После подготовки матрицы смежности её строки можно подавать в модель генерации векторных представлений. Структура данной модели и способ ее обучения будут обсуждаться в пунктах 2.2.2 и 2.2.4 соответственно.

После обучения модели генерации векторных представлений для каждого узла генерируются векторные представления, которые будут добавлены в поле node.embedding. Данные векторные представления парами можно подавать на модель классификации наличия связи чтобы проверить, есть ли связь между соответствующими вершинами графа. Структура данной модели и способ ее обучения будут обсуждаться в пунктах 2.2.3 и 2.2.4 соответственно.

После обучения модели классификации наличия связи её можно сохранить в поле tflite\_network. После этого получившийся файл готов к отправке и считыванию клиентом, не видевшим исходный граф.

Таким образом исходный граф проходит следующие этапы обработки:

* построение матрицы смежности;
* передача строк матрицы смежности в модель генерации векторного представления;
* передача сгенерированных векторных представлений парами в модель классификации наличия связи.

Изображение данных этапов приведено на рисунке 4.

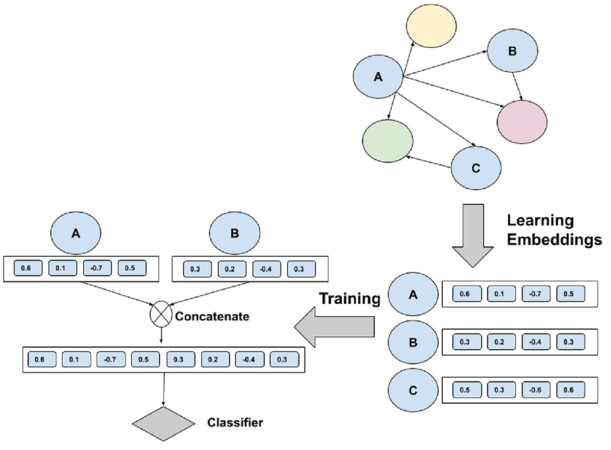


Рисунок 4 — Этапы обработки исходного графа

### Модель генерации векторных представлений

Для модели генерации векторных представлений было решено использовать перцептрон с активационной функцией ReLU. Перцептрон содержит входной слой, где каждый нейрон соответствует элементу строки матрицы смежности, может содержать один или несколько скрытых слоёв нейронов, где каждый нейрон очередного слоя соединён с каждым нейроном предыдущего слоя, и выходной слой, где каждый выходной нейрон соединён с нейроном предыдущего слоя. С каждой связью между нейронами ассоциирован вес, который домножается на вход по этой связи, результаты суммируются, к ним прибавляется некоторая константа и результат обрабатывается некоторой активационной функцией [48]. Функцию нейрона можно представить следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

где  — входной вектор нейрона,

 — активационная функция,

 — размерность входного вектора нейрона,

 — веса, на которые домножается входной вектор нейрона,

 — вес, который прибавляется к сумме.

Активационная функция — тот элемент, который позволяет нейронной сети представлять нелинейные функции. Классическая активационная функция, используемая в искусственных нейронных сетях, — сигмоидальная функция [49]:

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

Проблема данной активационной функции в том, что она относительна сложна для вычисления. При этом, как показала практика, для получения нейронной сетью возможности представления нелинейных функций, такая сложная активационная функция не нужна. В результате было выявлено, что в большинстве случаев хорошо работает функция, получившая название ReLU [50] [51]:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Очевидно, что функция ReLU вычисляется максимально быстро, и было показано, что её возможностей в большинстве случаев хватает.

В качестве активационной функции выходного слоя может использоваться функция бинаризации:

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

Преимущество данной функции в том, что её результат можно сохранить в один бит, что заметно уменьшает размер графа, сжатого разрабатываемым методом. Недостатком данной функции является то, что требуется больше выходных нейронов для передачи эквивалентного количества информации, а значит требуется больше времени для обучения модели. Также особенностью данной функции является то, что её производная во всех точках кроме нуля равна нулю, а в нуле производная данной функции не определена. Чтобы решить эту проблему, используется отмасштабированная производная сигмоидальной функции (9) — она даёт достаточно неплохое приближение поведения функции бинаризации, и при этом данная производная везде определена.

Путём экспериментов было определено, что количество скрытых слоёв, количество нейронов в них и количество выходов зависят от сложности кодируемого графа. Так, для простейшего для данного метода графа было достаточно одного полносвязного слоя (то есть имелось 0 скрытых слоёв) и 8 выходных нейронов с функцией бинаризации в качестве функции активации. Графическое изображение данной структуры приведено на рисунке 5.

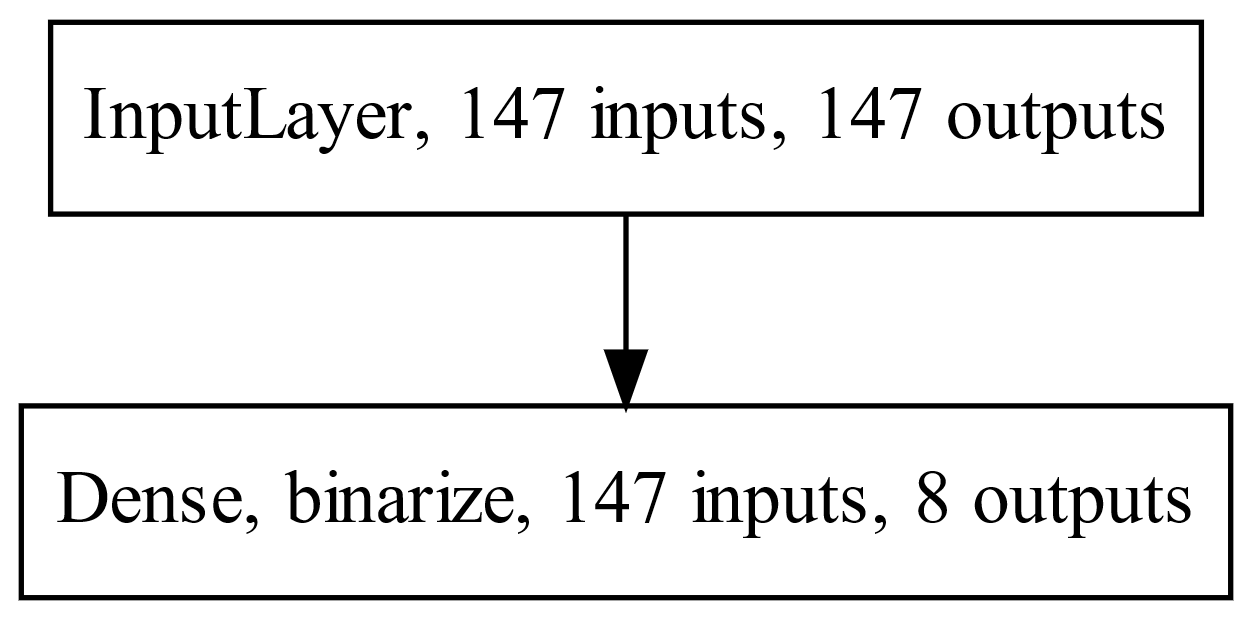


Рисунок 5 — Графическое изображение модели генерации  
векторных представлений для простейшего графа

Для графа составов команд Чемпионата мира по футболу 2014 [52] потребовалась сеть с двумя скрытыми слоями (со 128 и 64 нейронами соответственно) и выходной слой с 16 нейронами и функцией активации ReLU. Графическое изображение данной структуры приведено на рисунке 6.

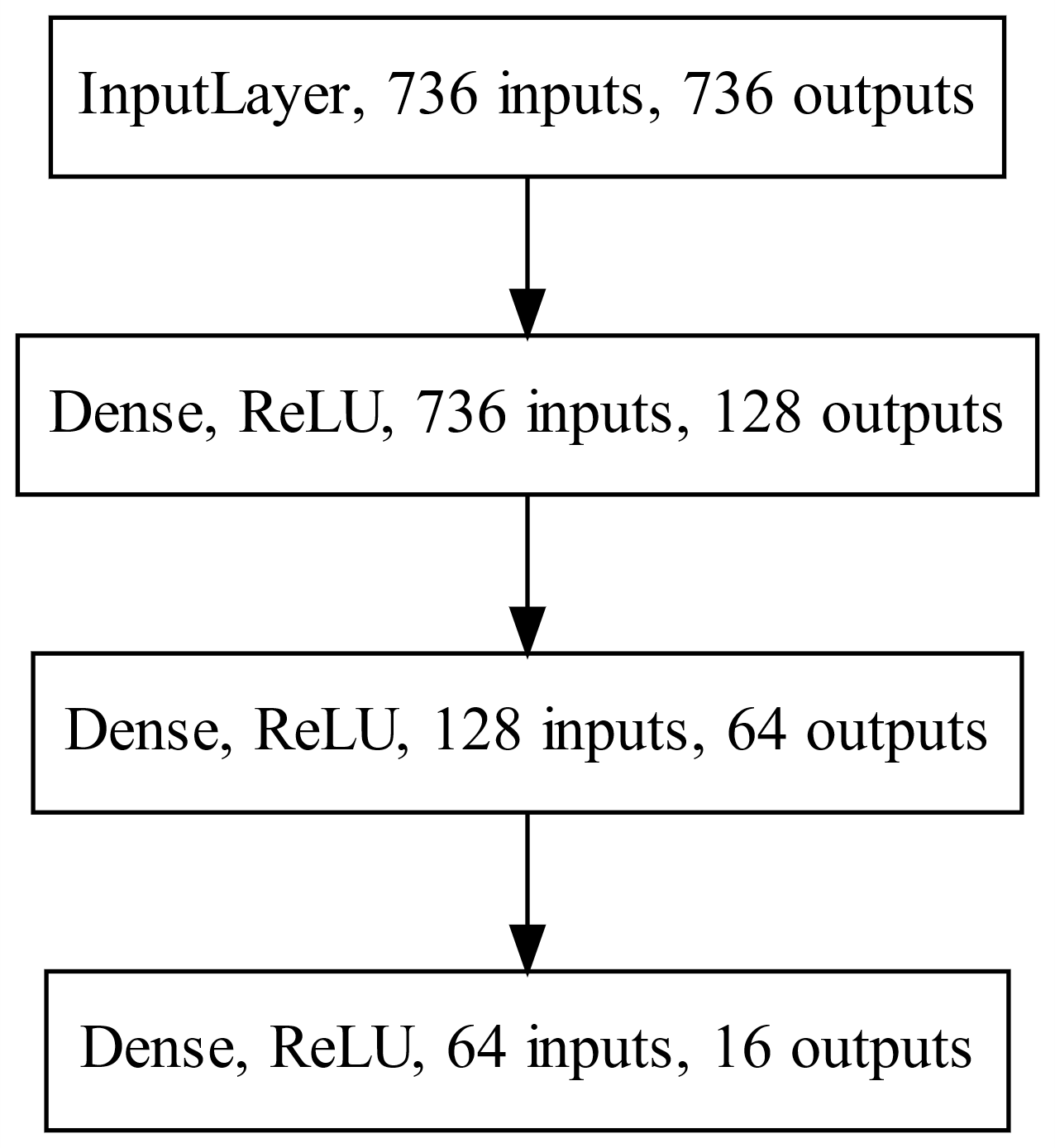


Рисунок 6 — Графическое изображение модели генерации векторных  
представлений

В принципе единственное, на что влияет количество и размер скрытых слоёв данной модели: время и сложность обучения. На выходной результат влияет только количество выходных нейронов модели и их функция активации (используется ли функция бинаризации или нет).

### Модель классификации наличия связи

Для модели классификации наличия связи было решено использовать перцептрон с активационной функцией ReLU для скрытых слоёв и функцией активации softmax для выходного слоя. Перцептрон и функция активации ReLU были описаны в пункте 2.2.2.

Функция активации softmax применяется в машинном обучении для задач классификации, когда модель машинного обучения должна всегда выдавать один класс на выходе. Особенностью данной активационной функции является то, что она работает сразу со всеми нейронами слоя, и она, по сути, нормирует выходы этого слоя нейронов [53]:

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

где  — выходной вектор нейронов слоя до функции активации,

 — количество нейронов в слое (размерность выходного вектора нейронов слоя).

Таким образом выходы модели машинного обучения приобретают смысл вероятности принадлежности объекта определённому классу.

Особенностью данной модели являются то, что на вход ей подаются сразу два векторных представления. Проблема заключается в том, что все традиционные искусственные нейронные сети могут работать только с одним входным вектором. Для решения этой проблемы применяется специальный слой конкатенации, который берёт переданные входные векторы и склеивает их в один. После этого следующие слои могут работать с ними как с единым целым.

Практика показала, что для модели классификации наличия связи чаще всего требуется один скрытый слой размером, соответствующим размерности конкатенации векторных представлений. Выходной слой всегда состоит из двух нейронов (один из них предсказывает отсутствие связи, тогда как другой нейрон показывает наличие связи), и для этого слоя используется функция активации softmax, описанная выше. Графическое изображение данной структуры приведено на рисунке 7.

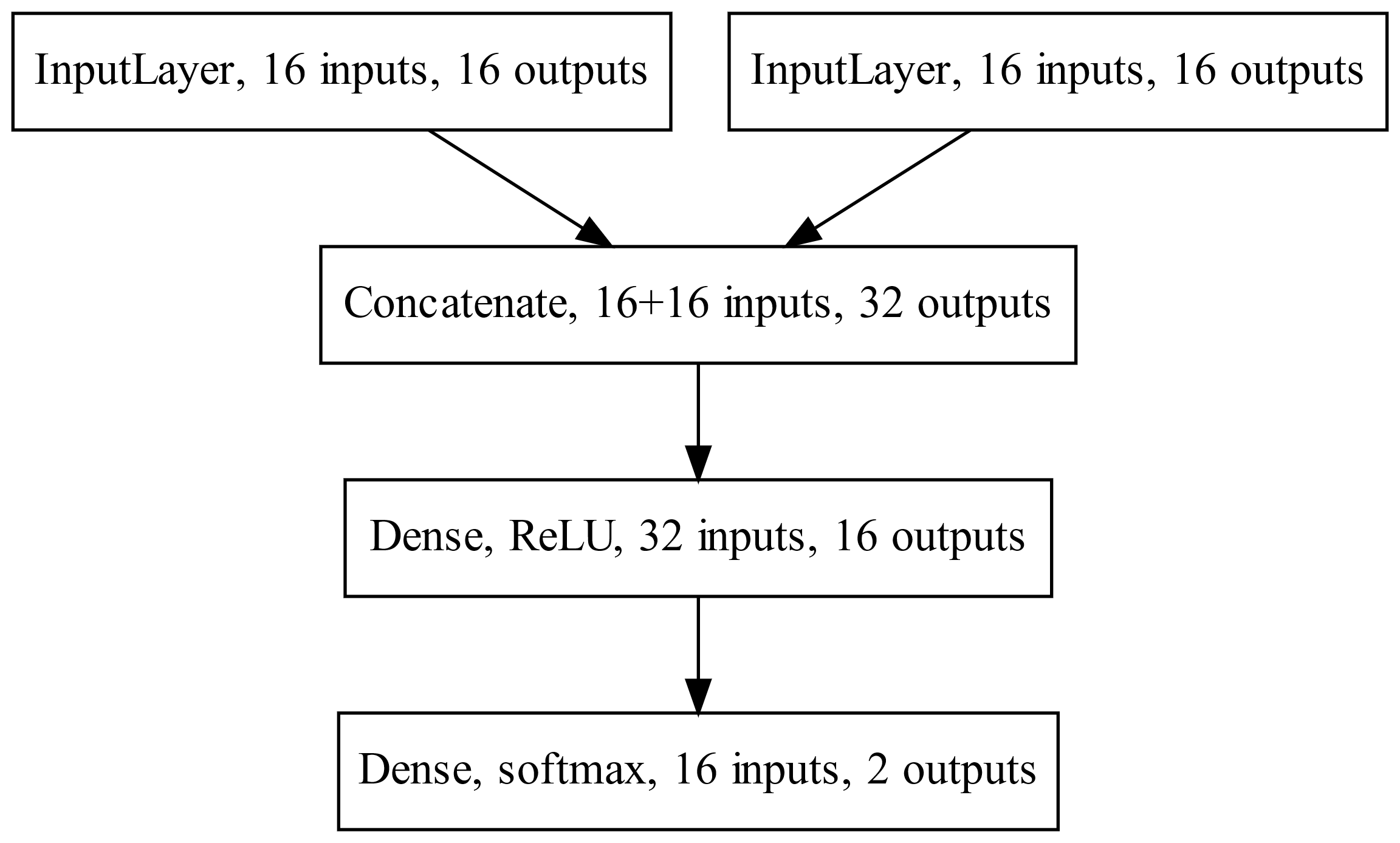


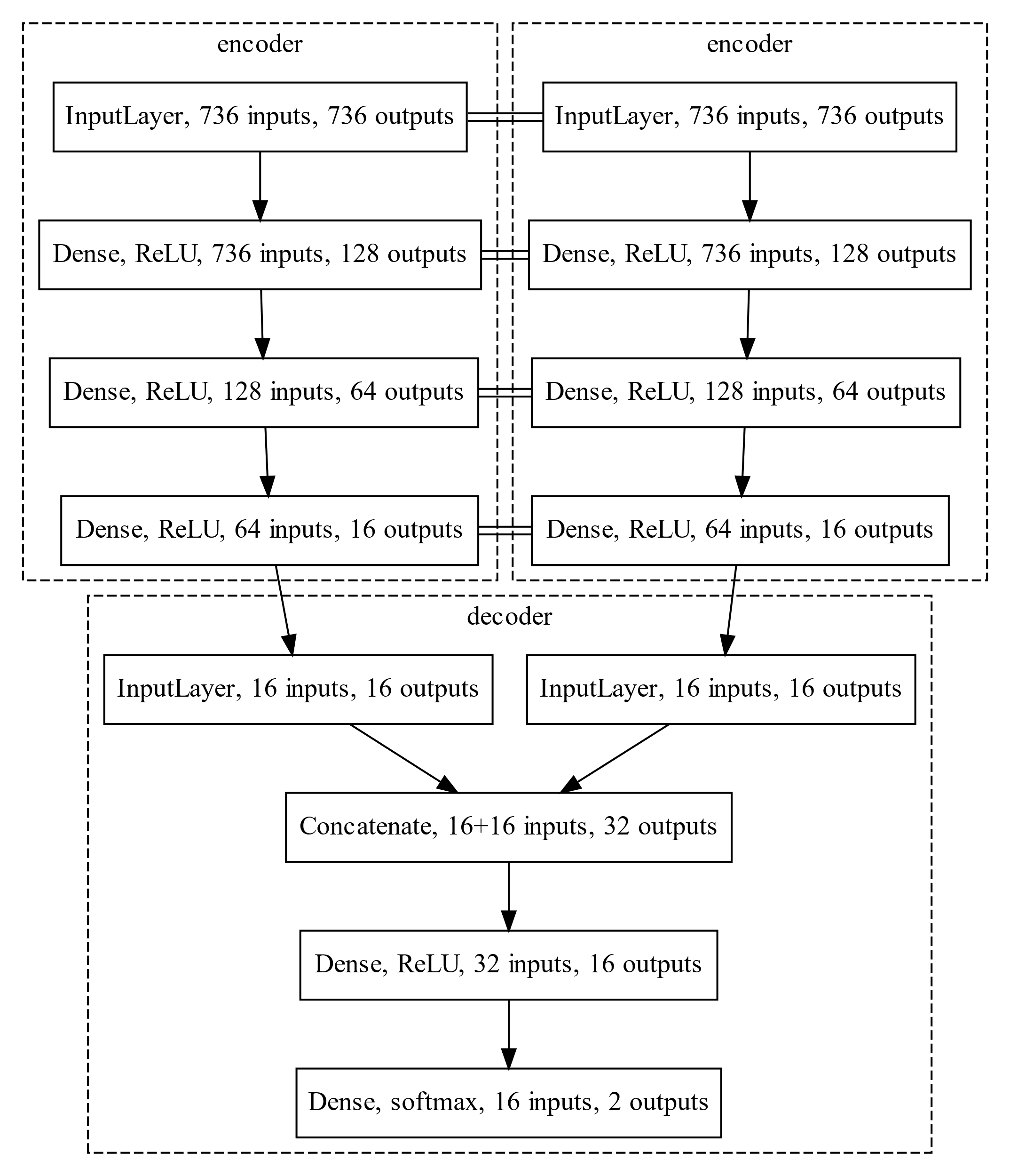
Рисунок 7 — Графическое изображение модели классификации наличия связи

### Процесс обучения моделей

Главная сложность обучения описанных выше моделей заключается в том, что они взаимосвязаны. Кроме того, непонятно, какую функцию потери использовать для обучения модели генерации векторных представлений. В методе Structural Deep Network Embedding, описанном в подпункте 1.1.2.4.3, предлагается обучать модель генерации векторного представления вместе с моделью расшифровки векторного представления обратно в строку матрицы смежности, таким образом функцию потерь можно определить как разницу между входом первой модели и выходом второй модели. Данная архитектура искусственных нейронных сетей называется «автокодировщик» и довольно широко применяется для генерации векторных представлений в различных предметных областях [54]. Кроме того, в данном методе предлагалось обучать параллельно сразу две копии модели с одинаковыми весами для того, чтобы накладывать штраф на разность векторных представлений (таким образом обеспечивается отсутствие выбросов среди элементов векторного представления). Однако у данного подхода для разрабатываемого метода сжатия графов есть два недостатка:

* требуется обучение дополнительной модели, которая будет использоваться только для вычисления функции потерь;
* модель генерации векторных представлений будет пытаться сохранить всю информацию, требующуюся для восстановления всей строки матрицы смежности по одному векторному представлению, хотя такая задача не стоит.

В результате было решено вместо модели расшифровки векторных представлений использовать модель классификации наличия связи. Это позволяет обучить модель генерации векторных представлений оставлять ровно ту информацию, которая нужна модели классификации наличия связи. Однако, так как модель классификации наличия связи требует на вход пару векторных представлений, то параллельно обучаются две копии модели генерации векторного представления с общими весами, и выход каждой из копий подаётся в качестве входа в модель классификации наличия связи. Общий вид модели для обучения приведен на рисунке 8.



В рамку с подписью encoder обведена модель генерации векторного  
представления. В рамку с подписью decoder обведена модель  
классификации наличия связи. Двойная линия показывает, что  
у указанных слоёв используются общие веса.

Рисунок 8 — Общий вид модели для обучения

В качестве функции потерь используется перекрёстная энтропия. Минимизация перекрёстной энтропии часто используется для оптимизации функций, выходом которых являются вероятностные распределения [55]. В качестве метода оптимизации используется метод AdaMax, так как он показывает хорошие результаты для обучения моделей построения векторных представлений [56] [57]. В качестве метрики качества модели использовалась точность предсказания [58].

Для получения более стабильных результатов обучения была применена -регуляризация, или регуляризация Тихонова, которая добавляет штраф на квадрат весов нейронной сети [59] [60]:

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

Эта процедура позволяет выравнивать веса искусственной нейронной сети и избегать выбросов среди них. Для всех слоёв (кроме входных, где веса отсутствуют), как видно в пунктах 2.3.3 и 2.3.4, применена -регуляризация с параметром , что заметно улучшает сходимость при обучении моделей.

### Процедура квантования модели классификации наличия связи

Так как основным направлением при разработке данного метода сжатия графов является уменьшение размера итогового информационного представления графа, то необходимо уменьшать не только сами векторные представления, но и модель классификации наличия связи. Помимо тривиальных способов уменьшения размера модели вида уменьшения количества слоёв и нейронов, модель ещё можно подвергнуть процедуре квантования.

Все модели машинного обучения по умолчанию работают с весами, представленными числами с плавающей запятой. Данные числа занимают 4 байта или 32 бита [61]. В процедуре квантования данные числа заменяются однобайтовыми целыми числами (соответственно они занимают 8 бит, то есть в 4 раза меньше). Данная процедура, во-первых, уменьшает размер модели приблизительно в те же 4 раза (так как основной информационный объём моделей машинного обучения занимают как раз веса модели), а во-вторых, ускоряют вычисления, так как с целыми числами вычислителям работать заметно проще. Недостатком является потеря точности вычислений, однако её можно частично компенсировать с помощью уточняющей процедуры обучения с учётом квантования [62].

Для проведения уточняющей процедуры обучения с учётом квантования необходимо преобразовать модель в модель, учитывающую квантование при своём обучении. Для языка программирования Python в случае использования библиотеки машинного обучения TensorFlow это можно выполнить с помощью функции tfmot.quantization.keras.quantize\_model, которая поставляется в библиотеке tensorflow\_model\_optimization (обозначается как tfmot). После этого обучение можно выполнить повторно, и веса будут подстроены с учётом эмуляции шума квантования. Данная процедура довольно точно воспроизводит поведение квантованной модели.

Процедура обучения с учётом квантования — довольно новая технология, и на момент написания расчётно-пояснительной записки рекомендовалось ставить ежедневную сборку библиотеки машинного обучения TensorFlow, поскольку в релизе 2.2.0, выпущенном 6 мая 2020 года, не хватало поддержки некоторых возможностей для полноценной работы обучения с учётом квантования. Также в текущей версии библиотеки оптимизации моделей машинного обучения tensorflow\_model\_optimization имеется ошибка, из-за которой по умолчанию слой конкатенации не может быть обработан. Однако, так как данный тип слоя не содержит никаких весов, можно явно определить, что к слою не нужно применять никакие преобразования для квантования. Это делается функцией tfmot.quantization.keras.quantize\_annotate\_layer, с помощью которой определяется, что к слою конкатенации следует применять преобразования с помощью класса NoOpQuantizeConfig из библиотеки tensorflow\_model\_optimization.python.core.quantization.  
keras.default\_8bit.default\_8bit\_quantize\_configs. Применение данных функции и класса показано в пункте 2.2.3. После этого при выполнении преобразования модели в модель, учитывающую квантование при своём обучении, требуется явным образом добавить класс NoOpQuantizeConfig в контекст преобразования. Это делается следующим образом:

with tfmot.quantization.keras.quantize\_scope(

{'NoOpQuantizeConfig': NoOpQuantizeConfig}):

quant\_decoder = (

tfmot.quantization.keras.quantize\_model(decoder))

Получившаяся модель с учётом квантования при обучении в визуализации пакета TensorFlow приведена на рисунке 9.

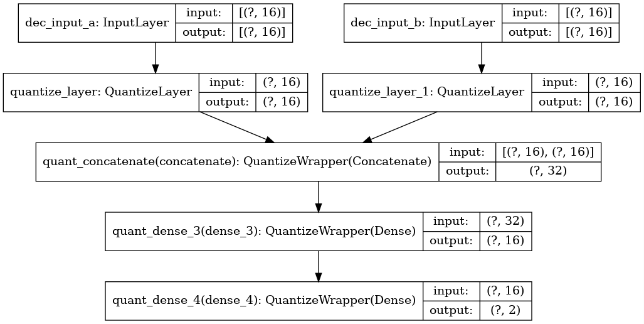


Рисунок 9 — Модель классификации наличия связи с учётом квантования,  
визуализированная пакетом TensorFlow

После этого можно собрать модель для обучения заново, заменив модель классификации наличия связи на модель, учитывающую квантование при своём обучении. При этом сохраняются уже выученные веса, что позволяет достаточно быстро дообучить модель с учётом квантования. Итоговая модель для обучения в визуализации пакета TensorFlow приведена на рисунке 10.

После выполнения повторного обучения с учётом квантования необходимо произвести, собственно, квантование модели. В TensorFlow квантованные модели работают с помощью библиотеки TensorFlow Lite, которая позволяет использовать готовые модели машинного обучения на мобильных устройствах и на устройствах «интернета вещей», поэтому преобразование выполняется с помощью функций из соответствующей библиотеки [63]:

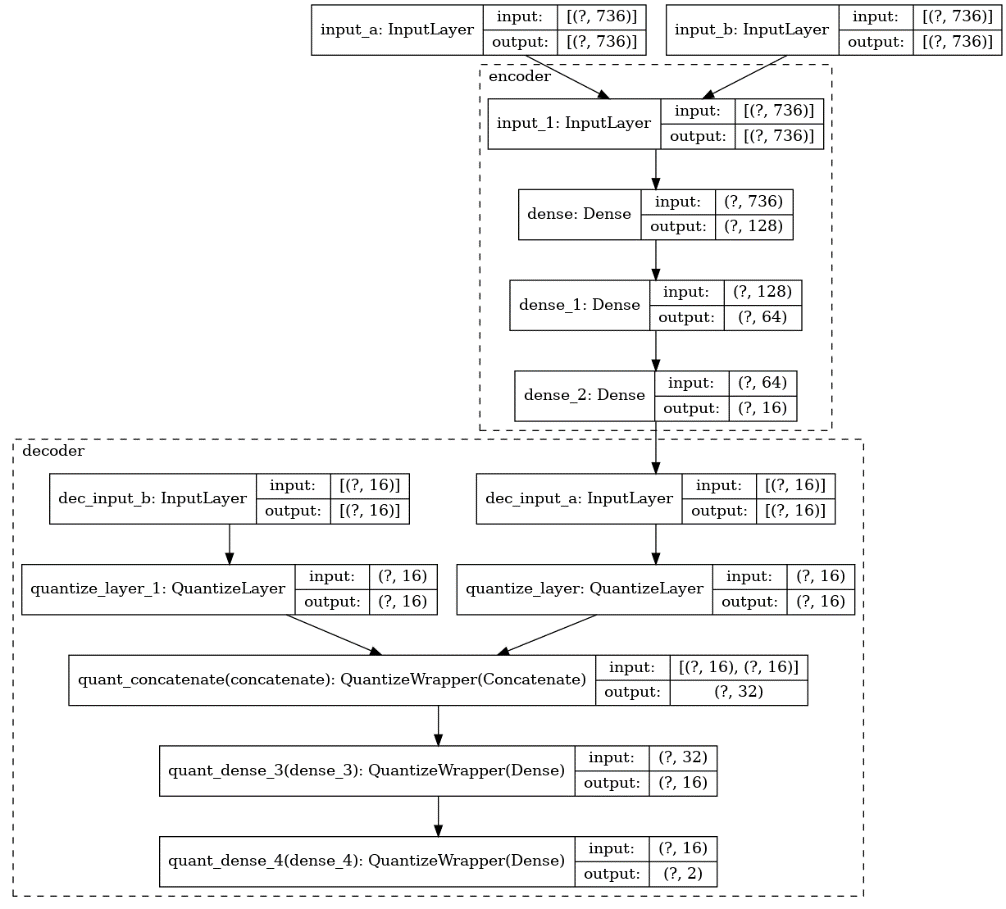
converter = tf.lite.TFLiteConverter.from\_keras\_model(

quant\_decoder)

converter.optimizations = [

tf.lite.Optimize.OPTIMIZE\_FOR\_SIZE]

tflite\_quant\_model = converter.convert()



Модель encoder из-за особенностей визуализации приведена один раз,  
но она на самом деле используется дважды, как и в прошлый раз.

Рисунок 10 — Модель для обучения с учётом квантования,  
визуализированная пакетом TensorFlow

После кода, приведенного выше, в переменной tflite\_quant\_model остаётся бинарное представление полученной модели, которое после записи в файл будет готово к использованию.

### Очистка бинарного представления модели сжатия графов

Бинарное представление обученной квантованной модели, полученное в пункте 2.2.5, всё ещё не является оптимальным бинарным представлением модели классификации наличия связи. Проблема заключается в том, что в данном бинарном представлении перечислены названия всех слоёв и название самой модели, и данные названия, очевидно, не нужны для использования модели. Более того, процедура квантования оставляет достаточно длинные названия всех элементов модели, что заметно увеличивает размер модели (учитывая, что для рассмотренных выше графов модель классификации наличия связи занимает приблизительно 2 килобайта, и все названия в ней с учётом добавленных процедурой квантования занимают ещё целый килобайт). Таким образом желательно убрать длинные названия из бинарного представления модели, так как в них нет необходимости для работы с моделью.

К сожалению, библиотека TensorFlow Lite не поддерживает редактирование моделей. Единственный вариант убрать все длинные названия из модели — убрать их из бинарного представления модели. Это можно сделать, если знать, в каком формате закодирована модель. TensorFlow Lite для своих моделей использует протокол сериализации структурированных данных FlatBuffers, поскольку он оптимизирован для доступа к данным без их распаковки. Для сравнения работа с данными из файла Protocol Buffers требует полной распаковки всего файла, копирования данных из файла в отдельные области памяти и так далее, тогда как FlatBuffers может предоставлять доступ к данным прямо из файла и может полностью избегать копирования данных. Это преимущество очень важно для TensorFlow Lite, поскольку позволяет максимально быстро начинать работу с моделью. Еще одним преимуществом Protocol Buffers является автоматическая генерация сериализаторов и десериализаторов (то есть исходного кода для чтения и записи данных в созданный формат данных) для ещё большего количества популярных языков программирования по сравнению с Protocol Buffers: C++, C#, C, Go, Java, Kotlin, JavaScript, Lobster, Lua, TypeScript, PHP, Python, Rust и Swift [64].

Таким образом, требуется взять описание формата модели TensorFlow Lite, сгенерировать для него десериализатор и сериализатор, после чего воспользоваться ими для очистки ненужных названий. Описание формата занимает довольно много места, однако его можно без проблем скачать из репозитория TensorFlow [65]. После этого из него можно сгенерировать модуль для работы с данным форматом, и использовать его для удаления всех строк.

### Обработка потери точности

К сожалению, не всегда возможно достичь стопроцентной точности классификации наличия связи с помощью модели машинного обучения. Этому могут способствовать следующие причины: структура графа слишком сложна для ограниченного по размеру векторного представления, в графе присутствуют некоторые исключения, которые модель не смогла учесть, или же в самом графе присутствуют некоторые ошибки, которые были исправлены с помощью обобщающих свойств модели. Есть два варианта, как можно поступить с потерей точности при векторном представлении.

Во-первых, можно оставить всё как есть, таким образом получив сжатие с потерями. Данный вариант возможен и может быть даже интересен в случае, если далеко не все связи графа представлены правильно. С помощью сжатия с потерями можно получить новые знания о структуре графа. Однако очевидно, что данный вариант может быть применён далеко не всегда.

Во-вторых, можно подготовить списки исключений, на которых модель даёт неправильный ответ, чтобы уже по данным спискам можно было разобраться, какой ответ правильный. Для того, чтобы это сделать, необходимо запустить модель на всех возможных парах векторных представлений узлов, и записать пары, на которых модель даёт неправильный ответ. После этого, согласно пункту 2.3.1, в поле edges можно записать пары узлов, на которых модель дала ложноотрицательный результат, а в поле excluded\_edges можно записать пары узлов, на которых модель дала ложноположительный результат. Таким образом получим гибридное представление графа, в некоторой степени сочетающее в себе компактность векторного представления и точность списка связей.

Выбор варианта обработки потери точности зависит от предметной области, в которой используется граф, от задач, решаемых методом сжатия графа, от целей, преследуемых пользователями данного метода сжатия графа. По умолчанию предполагается, что обработка потери точности выполняется с помощью списком исключений, однако при желании эту обработку можно не выполнять.

## Программная реализация метода сжатия графов

Метод сжатия графов на основе векторного представления было решено реализовать на языке программирования Python с использованием библиотеки машинного обучения TensorFlow [66]. Основное преимущество этой библиотеки для данной работы заключается в том, что она позволяет конвертировать модели машинного обучения в довольно компактный формат, пригодный для использования с библиотекой машинного обучения TensorFlow Lite, работающей в том числе на мобильных устройствах и устройствах «интернета вещей» [63].

### Формат данных

За основу формата данных для хранения графа был выбран протокол сериализации структурированных данных Protocol Buffers, поскольку он оптимизирован под минимальный размер сериализованных данных. В частности, описание формата (названия объектов, полей, атрибутов) хранится отдельно и не входит в передаваемые данные. Также небольшие целые числа (такие как идентификаторы узлов, например) хранятся меньшим числом байт с помощью формата величин переменной длины, описанного в подпункте 1.1.2.1, что позволяет сэкономить значительное число байтов в случае применения этого формата для графов. Кроме того, незаполненные поля в данных просто не передаются, что также позволяет экономить место. Еще одним преимуществом Protocol Buffers является автоматическая генерация сериализаторов и десериализаторов (то есть исходного кода для чтения и записи данных в созданный формат данных) для многих популярных языков программирования: C++, C#, Dart, Go, Java, JavaScript, Objective-C, PHP, Python, Ruby [67].

Учитывая, что в Protocol Buffers неиспользуемые поля не занимают место, имеет смысл сделать единый формат для хранения графа, поддерживающих и представление в виде списка связей, и в виде векторного представления. Это позволит удобно сравнивать оба представления, ведь для этого достаточно будет лишь закодировать граф обоими способами и сравнить размер получившегося файла. В результате было создано следующее описание структуры данных на языке Protocol Buffers:

syntax = "proto3";

message Graph {

message Node {

uint64 id = 1;

bytes data = 2;

bytes embedding = 3;

}

message Edge {

// Not directed for now

uint64 start = 1;

uint64 end = 2;

}

repeated Node nodes = 1;

repeated Edge edges = 2;

bytes tflite\_network = 3;

repeated Edge excluded\_edges = 4;

}

Значения после знака «равно» являются уникальными идентификаторами поля в сообщении Protocol Buffers, остальной синтаксис является интуитивно понятным. Разберем, зачем нужны какие типы и поля в них:

* Node — описание типа узла графа.
  + id — уникальный идентификатор узла графа.
  + data — поле, характеризующее некоторые свойства узла графа. В данной работе не используется, но может быть полезно для практического применения.
  + embedding — векторное представление связей узла. В случае представления связей в виде списка связей данное поле не используется.
* Edge — описание типа связи графа.
  + start — узел с одной из сторон связи.
  + end — узел с другой стороны связи.
* nodes — список узлов в графе в виде объектов типа Node.
* edges — список связей в графе в виде объектов типа Edge. Используется в качестве списка исключений (непокрытых связей) в случае векторного представления связей в графе.
* tflite\_network — поле для хранения нейронной сети, которая по двум векторным представлениям предсказывает, есть ли связь между узлами или она отсутствует.
* excluded\_edges — список связей, которые на самом деле отсутствуют в графе, в виде объектов типа Edge. Используется в качестве списка исключений (излишних связей) в случае векторного представления связей в графе.

После создания данного текстового описания формата графа с помощью следующей команды был вызван компилятор proto-файлов в исходный код модуля сериализации и десериализации данных на языке программирования Python:

protoc --python\_out=. graph.proto

В результате был автоматически сгенерирован модуль для языка программирования Python (использующий библиотеку google.protobuf), который можно использовать для сериализации данных графа и сравнения объема данных. Данный модуль подключается следующим образом:

import graph\_pb2

После этого сохранение данных графа в виде списка связей выглядит следующим образом:

graph = graph\_pb2.Graph()

for i, xi in enumerate(nodes):

node = graph.nodes.add()

node.id = i

for x, y in edges:

edge = graph.edges.add()

edge.start = x

edge.end = y

with open("plain\_graph.bin", "wb") as f:

f.write(graph.SerializeToString())

Сохранение данных графа в векторном представлении выглядит следующим образом:

graph = graph\_pb2.Graph()

for i, xi in enumerate(nodes):

node = graph.nodes.add()

node.id = i

node.embedding = np.packbits(xi).tobytes()

with open(tflite\_quant\_model\_file, 'rb') as f:

graph.tflite\_network = f.read()

with open("networked\_graph.bin", "wb") as f:

f.write(graph.SerializeToString())

### Построение матрицы смежности графа

Как было описано в пункте 2.2.1, сначала необходимо преобразовать связи графа в матрицу смежности. Отличие требуемой матрицы смежности от классической заключается в том, что узел считается связанным с самим собой. Это отличие упрощает реализацию алгоритмов обучения и позволяет повышать точность получающейся модели. Блок-схема алгоритма преобразования графа вида в матрицу смежности приведена на рисунке 11. Данный алгоритм предполагает, что множество состоит из последовательных натуральных чисел от 1 до , то есть . Если это равенство нарушается, следует перенумеровать вершины, как это было описано в подпункте 1.1.2.1.

Данный алгоритм на языке программирования Python с использованием библиотеки NumPy выглядит следующим образом [68]:

M = np.zeros((len(V), len(V)), dtype=np.uint8)

for i in range(len(V)):

M[i][i] = 1

for i, j in E:

M[i][j] = 1

M[j][i] = 1

После этого в переменной M получаем готовую для использования методом сжатия графов матрицу смежности.

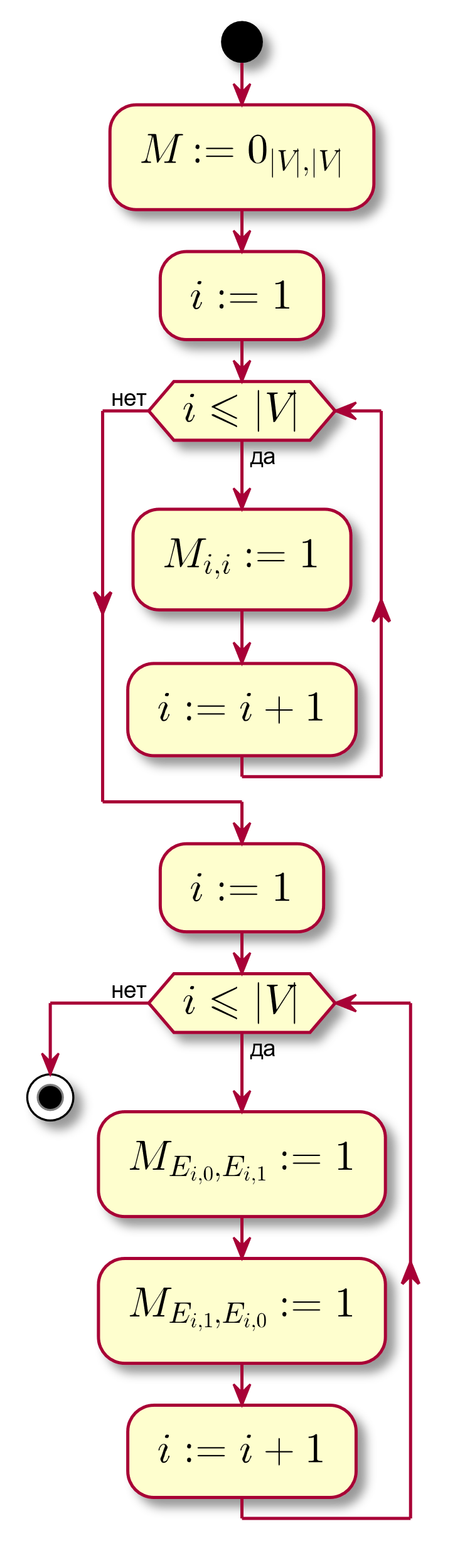


Рисунок 11 — Блок-схема алгоритма построения матрицы смежности графа

### Модель генерации векторных представлений

Модель генерации векторных представлений для простейшего графа на языке программирования Python с использованием фреймворка TensorFlow записывается следующим образом [66]:

encoder = tf.keras.Sequential([

tf.keras.Input(shape=(len(V),)),

tf.keras.layers.Dense(

8, activation=binarize,

kernel\_regularizer=tf.keras.regularizers.l2(

0.001))

], name='encoder')

Та же модель для графа составов команд Чемпионата мира по футболу 2014 на языке программирования Python с использованием фреймворка TensorFlow записывается следующим образом [52]:

encoder = tf.keras.Sequential([

tf.keras.Input(shape=(len(V),)),

tf.keras.layers.Dense(

128, activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(

0.001)),

tf.keras.layers.Dense(

64, activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(

0.001)),

tf.keras.layers.Dense(

16, activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(

0.001))

], name='encoder')

Видно, что использование библиотеки машинного обучения TensorFlow позволяет довольно легко описывать подобные модели.

### Модель классификации наличия связи

Из-за наличия двух входов модель классификации наличия связи не является простой последовательной моделью, поэтому все слои в данной модели приходится связывать вручную. В результате данная модель на языке программирования Python с использованием фреймворка TensorFlow записывается следующим образом:

decoder\_input\_a = tf.keras.Input(

shape=(EMBEDDING\_SIZE,), name='dec\_input\_a')

decoder\_input\_b = tf.keras.Input(

shape=(EMBEDDING\_SIZE,), name='dec\_input\_b')

decoder\_input =

tfmot.quantization.keras.quantize\_annotate\_layer(

tf.keras.layers.Concatenate(),

NoOpQuantizeConfig())(

[decoder\_input\_a, decoder\_input\_b])

decoder\_dense = tf.keras.layers.Dense(

EMBEDDING\_SIZE, activation='relu',

kernel\_regularizer=tf.keras.regularizers.l2(

0.001))(decoder\_input)

decoder\_output = tf.keras.layers.Dense(

2, activation='softmax',

kernel\_regularizer=tf.keras.regularizers.l2(

0.001))(decoder\_dense)

decoder = tf.keras.Model(

[decoder\_input\_a, decoder\_input\_b], decoder\_output,

name='decoder')

Смысл функций квантования, примененных к слою конкатенации, сохранённому в переменную decoder\_input, описан в пункте 2.2.5.

По большому счёту данный синтаксис не глобально сложнее того, что потребовался для модели генерации векторных представлений в пункте 2.3.3, но при этом предоставляет большую гибкость построения модели.

### Процесс обучения моделей

Как было описано в пункте 2.2.4, довольно проблематично обучить отдельно модель генерации векторных представлений и модель классификации наличия связи. Вместо этого предлагается обучать объединенную модель, где модель генерации векторных представлений используется дважды (но с общими весами), и рассматриваются сразу пары узлов. На языке программирования Python с использованием библиотеки машинного обучения TensorFlow данная структура записывается следующим образом:

input\_a = tf.keras.Input(shape=(len(V),), name='input\_a')

input\_b = tf.keras.Input(shape=(len(V),), name='input\_b')

encoded\_a = encoder(input\_a)

encoded\_b = encoder(input\_b)

decoded = decoder(

{'dec\_input\_a': encoded\_a, 'dec\_input\_b': encoded\_b})

model = tf.keras.Model(

[input\_a, input\_b], decoded, name='graph\_coder')

Далее необходимо скомпилировать модель, задав параметры обучения. Данные параметры задаются следующим образом:

model.compile(

optimizer=tf.keras.optimizers.Adamax(

learning\_rate=0.001),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

Смысл этих параметров описан в пункте 2.2.4.

Далее необходимо непосредственно начинать обучение модели. Однако, прежде чем это будет возможно сделать, необходимо подготовить данные для обучения. Как описывалось в пункте 2.3.2, сначала была построена матрица смежности. После этого эту матрицу смежности нужно преобразовать в отдельные строки, которые будут использоваться для обучения.

На вход функции обучения модели библиотека машинного обучения TensorFlow ожидает списки входных и выходных данных. В качестве входных данных будут использоваться все возможные попарные комбинации строк матрицы смежности. В качестве выходных данных будет использоваться признак наличия связи между соответствующими вершинами графа. Блок-схема алгоритма подготовки входных данных представлена на рисунке 12.

Данный алгоритм на языке программирования Python с использованием библиотеки NumPy выглядит следующим образом [68]:

X\_A = np.tile(M, (len(M), 1))

X\_B = np.repeat(M, len(M), axis=0)

Y = np.zeros(len(V) \* len(V), dtype=np.uint8)

c = 0

for i in range(len(V)):

for j in range(len(V)):

Y[c] = matrix[i][j]

c += 1

Видно, что за счёт функций np.tile и np.repeat запись исходных данных для обучения сильно упрощается, и получается достаточно лаконичной.

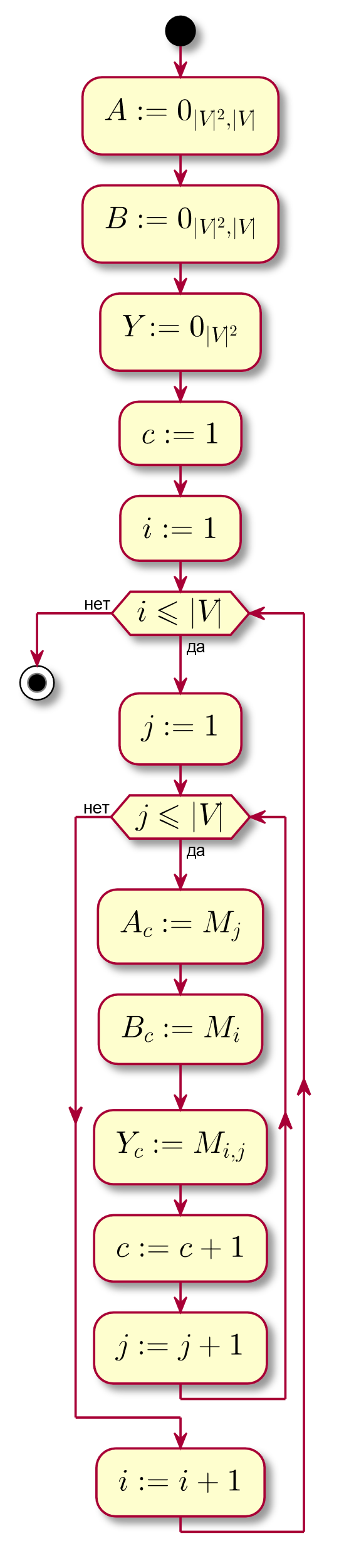


Рисунок 12 — Блок-схема алгоритма подготовки входных данных для обучения

После этого можно непосредственно запускать обучение объединённой модели. На языке программирования Python с использованием библиотеки машинного обучения TensorFlow это делается следующим образом:

model.fit({'input\_a': X\_A, 'input\_b': X\_B},

{'decoder': Y},

batch\_size=BATCH\_SIZE,

epochs=MAX\_EPOCHS,

callbacks=[stop\_callback]);

Функция stop\_callback позволяет остановить обучение после достижения требуемой точности (в случае данной выпускной квалификационной работы требуемая точность — 100%).

### Очистка бинарного представления модели

Как было описано в пункте 2.2.6, необходимо из бинарного представления модели удалить все строки, так как они не влияют на вычисления и лишь занимают место. TensorFlow Lite хранит модели в формате данных, сгенерированным библиотекой FlatBuffers по его описанию, поэтому прежде всего необходимо сгенерировать модули для работы с данным форматом данных. Для языка программирования Python это делается следующим образом:

flatc --python --gen-object-api schema.fbs

После этого можно декодировать бинарное представление модели, заменить все строки пустыми (или строками длиной в один символ в случае тензоров, так как их название не может быть пустым), и закодировать эту модель обратно. Это можно сделать следующим образом:

import tflite.Model as tflite\_schema

model\_obj = tflite\_schema.Model.GetRootAsModel(

bytearray(tflite\_quant\_model), 0)

model\_fb = tflite\_schema.ModelT.InitFromObj(model\_obj)

model\_fb.description = ''

for subgraph in model\_fb.subgraphs:

subgraph.name = ''

for tensor in subgraph.tensors:

tensor.name = 'T'

fb\_builder = flatbuffers.Builder(0)

model\_offset = model\_fb.Pack(fb\_builder)

fb\_builder.Finish(model\_offset, b"TFL3")

model\_data = fb\_builder.Output()

После проведения данных манипуляций модель начинает занимать минимально возможный объём, в котором её можно использовать для классификации наличия связей, и дальнейшее уменьшение размера информационного представления графа лежит в плоскости оптимизации параметров модели, таких как количество слоёв, количество нейронов в слое, размер векторного представление и применение функции бинаризации к нему.

### Обработка потери точности

В случае, если требуется сжатие без потерь, нужно проверить точность получившейся модели и найти все случаи, когда модель классификации наличия связи ошибается. К сожалению, для оценки точности получившейся модели недостаточно точности, посчитанной во время обучения — после квантования модель может потерять часть точности. Таким образом требуется снова проверить получившуюся модель на всех парах векторных представлений и убедиться, что она выдаёт адекватные значения.

Блок-схема алгоритма оценки точности получившейся модели и сбора информации об ошибках модели приведена на рисунке 13.

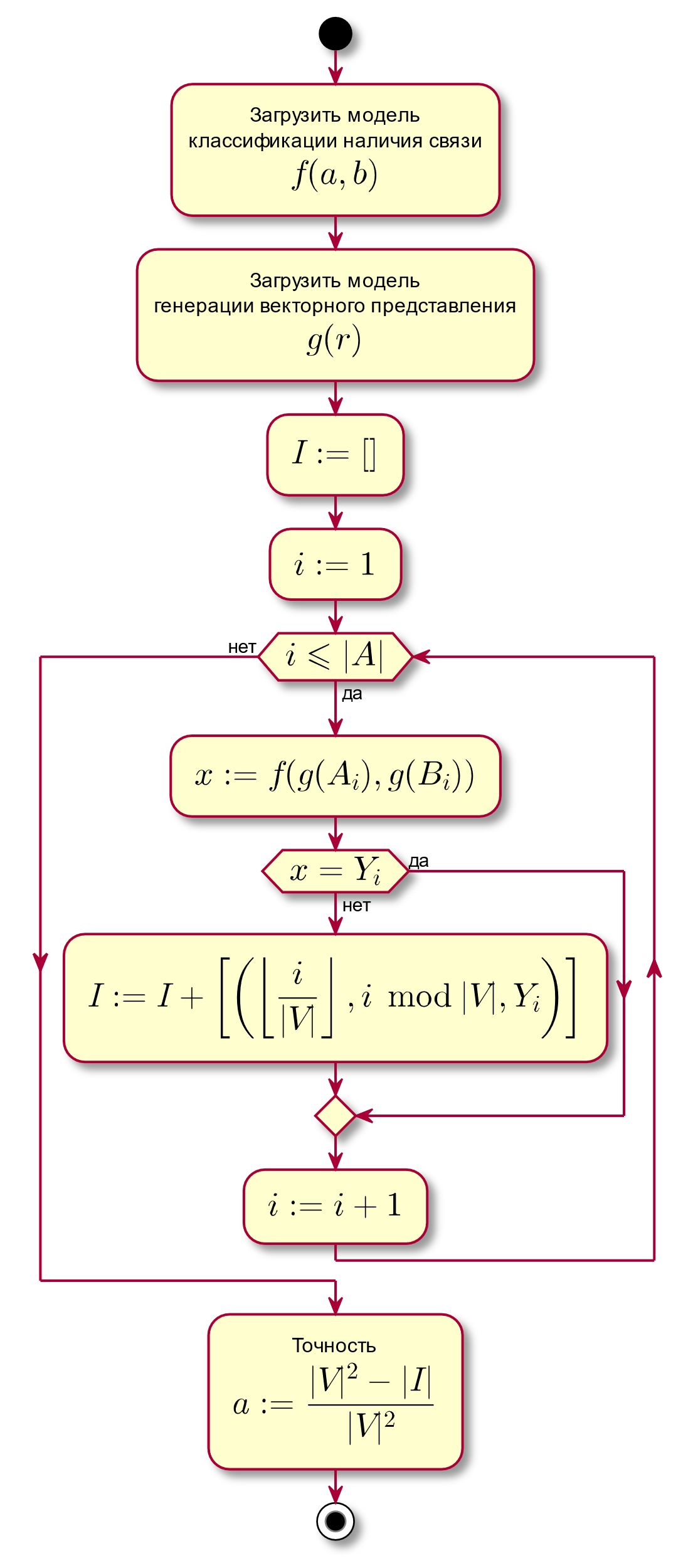


Рисунок 13 — Блок-схема алгоритма оценки точности итоговой модели

На языке программирования Python с использованием библиотеки машинного обучения TensorFlow Lite и библиотеки NumPy данная оценка выполняется следующим образом:

ml = tf.lite.Interpreter(model\_path=quant\_model\_file)

ml.allocate\_tensors()

input\_a\_index = ml.get\_input\_details()[0]["index"]

input\_b\_index = ml.get\_input\_details()[1]["index"]

output\_index = ml.get\_output\_details()[0]["index"]

I = []

with range(len(X\_A)) as t:

for i in t:

input\_a = encoder.predict(np.array([X\_A[i]]))

input\_b = encoder.predict(np.array([X\_B[i]]))

ml.set\_tensor(input\_a\_index, input\_a)

ml.set\_tensor(input\_b\_index, input\_b)

ml.invoke()

predictions = ml.get\_tensor(output\_index)

if np.argmax(predictions) != Y[i]:

I.append((i // len(V), i % len(V), Y[i]))

a = (len(V) \* len(V) – len(I)) / (len(V) \* len(V))

## Разработка модели для оценки границ применимости разработанного метода сжатия графов

Для оценки границ применимости разработанного метода сжатия графов было проведено моделирование информационного объема графа в зависимости от метода представления связей данного графа. В качестве методов представления связей, помимо предлагаемого векторного представления, были рассмотрены матрица смежности и список рёбер.

Моделирование было произведено в программном комплексе «Конструктор экспертных систем миварный (КЭСМИ) Wi!Mi РАЗУМАТОР». Данный программный комплекс позволяет эффективно переиспользовать различные части модели и, благодаря технологии миварного логического вывода, вычислять только необходимые для запрошенных выходов параметры [69]. Впоследствии построенную в КЭСМИ модель можно будет дополнить логикой подбора оптимального для заданного графа метода сжатия, чтобы автоматизировать принятие данного решения и избежать использования сложного с вычислительной точки зрения метода сжатия графов на основе векторного предложения в случае, если параметры графа выходят за границы применимости данного метода [70]. Таким образом будет создана гибридная информационная система, сочетающая в себе миварный модуль решения и нейросетевую модель извлечения знаний из графа [71].

### Составление формул для вычисления информационного объема графа в различных представлениях

#### Параметры графа

С точки зрения рассматриваемой задачи для расчетов необходимы два параметра графа:

* число вершин в графе (порядок графа);
* число ребер в графе (размер графа).

Для упрощения задачи будем считать, что с вершинами может быть связана некоторая информация объемом , а рёбра ненаправленные, и с ними не связана никакая другая информация.

На порядок и размер графа, а также на объем информации об узле графа наложены очевидные ограничения на их неотрицательность:

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

Кроме того, так как мы рассматриваем неориентированный простой граф, со связями которого не связана никакая дополнительная информация, действует следующее ограничение:

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

Для удобства исследования различных графов также будет использоваться такое понятие как плотность графа  — отношение числа рёбер графа к числу рёбер полного графа:

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

Эта величина, очевидно, также не может быть отрицательной:

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

Кроме того, в наших условиях эта величина также не может быть больше единицы:

|  |  |  |
| --- | --- | --- |
|  |  | (20) |

Для того, чтобы КЭСМИ мог вывести из плотности графа число вершин или рёбер в графе, выведем соответствующие значения из формулы выше:

|  |  |  |
| --- | --- | --- |
|  |  | (21) |

|  |  |  |
| --- | --- | --- |
|  |  | (22) |

#### Информационное представление графа

За основу формата данных для хранения графа был выбран протокол сериализации структурированных данных Protocol Buffers, поскольку он оптимизирован под минимальный размер сериализованных данных. В частности, описание формата (названия объектов и их полей) хранится отдельно и не входит в передаваемые данные. Также небольшие целые числа (такие как идентификаторы узлов, например) хранятся меньшим числом байт, что позволяет сэкономить достаточно байт для графов. Кроме того, незаполненные поля в данных просто не передаются, что также позволяет экономить место [21] [67].

Для упрощения задачи было решено сделать единый формат для хранения графа, поддерживающий все запланированные представления. Поскольку в Protocol Buffers незаполненные поля не передаются, потерь из-за наличия в описании неиспользуемых полей не будет. В результате было создано следующее описание структуры данных на языке Protocol Buffers:

syntax = "proto3";

message Graph {

message Node {

uint64 id = 1;

bytes data = 2;

bytes embedding = 3;

}

message Edge {

// Not directed for now

uint64 start = 1;

uint64 end = 2;

}

repeated Node nodes = 1;

bytes adjacency\_matrix = 2;

repeated Edge edges = 3;

bytes tflite\_network = 4;

repeated Edge excluded\_edges = 5;

}

Соответственно, для всех вариантов представления будет заполнено поле nodes (в случае векторного представления — с полем embedding, для остальных представлений без него). Далее:

* для матрицы смежности будет заполнено поле adjacency\_matrix (которое будет занимать бит);
* для списка ребер будет заполнено поле edges;
* для векторного представления будет заполнено поле tflite\_network (которое будет содержать модель машинного обучения, определяющую по векторным представлениям узлов наличие связи между ними), а также, по необходимости, с помощью полей edges и excluded\_edges будут указаны связи, для которых обученная нейронная сеть по каким-либо причинам ошибается (в поля будут записаны ложноотрицательные и ложноположительные связи соответственно).

Также про Protocol Buffers необходимо знать, как именно в нем кодируются данные, чтобы вычислить их размер. Важнейшее, что нужно знать — это способ кодирования чисел. В Protocol Buffers числа хранятся в формате чисел варьируемой длины: в этом формате число занимает ровно столько байт, сколько для него нужно, а длина последовательности байт, обозначающих число, определяется по старшему биту этих байт: если бит установлен, то байт не последний. Длину числа в байтах в данном формате можно определить следующим образом [21]:

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

Таким образом числа от 0 до 127 занимают всего 1 байт, числа от 128 до 16 383 занимают 2 байта, от 16 383 до 2 097 151 — 3 байта, и так далее. Кодировка отрицательных чисел выходит за рамки данной работы.

Поля в Protocol Buffers записываются следующим образом [21]:

* перед каждым полем записывается идентификатор поля (для описанного выше формата он всегда будет занимать 1 байт);
* если поле — число, то оно записывается в описанном выше формате чисел варьируемой длины;
* если поле — байты или вложенное сообщение, сначала записывается длина этих данных в описанном выше формате чисел варьируемой, а потом сами эти данные;
* массивы (отмеченные repeated в описании формата сообщения) записываются как несколько повторений одного и того же поля.

Рассчитаем, сколько места занимает поле nodes:

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

|  |  |  |
| --- | --- | --- |
|  |  | (26) |

В случае векторного представления будет ещё введён параметр , отражающий размер векторного представления связей узла в байтах и тогда в поле nodes нужны следующие поправки:

|  |  |  |
| --- | --- | --- |
|  |  | (27) |

|  |  |  |
| --- | --- | --- |
|  |  | (28) |

#### Представление графа в виде матрицы смежности

Матрица смежности является одним из самых используемых представлений связей графа. Для простоты на каждую ячейку матрицы будем выделять 1 байт. Это упрощает доступ к данным матрицы, а уменьшение размера ячейки до 1 бита даст лишь линейное уменьшение объема данных. Задачей данной работы является показать порядок объема графа в различных представлениях, и здесь линейная разница в размере не сыграет большой роли. Таким образом, размер представления графа в виде матрицы смежности рассчитывается следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | (29) |

|  |  |  |
| --- | --- | --- |
|  |  | (30) |

#### Представление графа в виде списка рёбер

Список рёбер является другим популярнейшим представлением связей графа. Его эффективность прямо зависит от плотности графа: чем рёбер больше, тем больше занимает список рёбер. По сути, список рёбер — это информационное представление множества . В списке рёбер каждому ребру соответствует кортеж из идентификаторов вершин, инцидентных данному ребру. Рассчитаем объем графа, представленного таким образом:

|  |  |  |
| --- | --- | --- |
|  |  | (31) |

|  |  |  |
| --- | --- | --- |
|  |  | (32) |

|  |  |  |
| --- | --- | --- |
|  |  | (33) |

#### Векторное представление связей графа

Векторное представление графов является инновационной областью науки, которая активно развивается. Векторные представления обычно используют для сравнения двух объектов: в контексте графов сравнивают либо графы целиком, либо отдельные узлы графа между собой. В данной работе рассматривается немного другое использование той же техники: по сгенерированным для узлов векторам отдельной нейронной сетью будем предсказывать, есть ли связь между узлами или нет. Для векторного представления необходимо ввести следующие параметры:

* — размер векторного представления связей узла в байтах;
* — нейронная сеть, предсказывающая наличие связи между узлами по паре связанных с этими узлами векторов;
* — вероятность ложноположительного срабатывания нейронной сети;
* — вероятность ложноотрицательного срабатывания нейронной сети.

Если в каком-то применении по какой-либо причине и пренебрежительно малы, и компенсация ошибок нейронной сети не требуется, можно принять и .

Рассчитаем объем графа, представленного таким образом:

|  |  |  |
| --- | --- | --- |
|  |  | (34) |

|  |  |  |
| --- | --- | --- |
|  |  | (35) |

### Реализация формул для вычисления информационного объема графа в КЭСМИ

Как обсуждалось в подразделе 2.4, моделирование было произведено в программном комплексе «Конструктор экспертных систем миварный (КЭСМИ) Wi!Mi РАЗУМАТОР» (далее КЭСМИ). Модели в КЭСМИ состоят из следующих типов объектов: классы, параметры, отношения и правила [72].

Класс — абстрактная сущность, обобщающее понятие. Класс может содержать в себе параметры и другие классы. Класс также имеет название, уровень иерархии и описание. В любой модели КЭСМИ должен существовать хотя бы один класс. Класс с высшим уровнем иерархии называют корневым. При создании новой модели, по умолчанию также создается корневой класс.

Параметр — это объект, содержащий значение определенного типа: числового или текстового. Как и класс, параметр имеет название, уровень иерархии и описание. Параметр может быть задан пользователем, а может быть вычислен из других параметров с помощью формул или сложных отношений.

Связь — ассоциация между объектами, значимая для рассматриваемой предметной области. Произвольное количество входных объектов посредством связи преобразуется в один (или набор) выходных объектов.

Отношение — вид связи, использующий абстрактные переменные, описывающий их взаимодействие.

Правило — вид связи, привязывающий отношение к конкретным объектам.

В КЭСМИ существуют четыре типа отношений:

* формула — математическая формула, состоящая из переменных и операторов, позволяющая из одного или нескольких параметров вычислить другой параметр;
* условное отношение — продукционное отношение вида «если …, то …, иначе …», позволяющее присвоить значение параметра по условию, зависящему от других параметров;
* ограничение — продукционное отношение, накладывающее ограничение на значение параметров;
* сложное отношение — программируемое отношение для опытного пользователя, задающее алгоритм вычисления значения выходных параметров по входным параметрам с помощью программы на языке программирования JavaScript.

Соответственно, в первую очередь необходимо задать свойства базового класса модели. Настройка свойств базового класса модели изображена на рисунке 14.

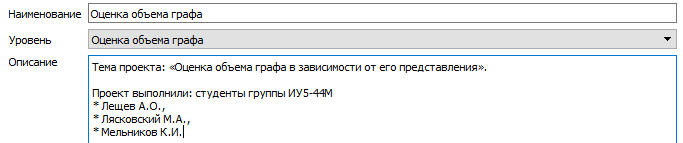


Рисунок 14 — Свойства базового класса модели

Далее был реализован класс «Параметры графа». Данный класс собирает в себе все параметры и формулы, описанные в подпункте 2.4.1.1. Свойства параметра , отвечающего за число рёбер в графе, приведены на рисунке 15. Свойства параметра , отвечающего за число вершин в графе, приведены на рисунке 16. Свойства параметра , отвечающего за плотность графа, приведены на рисунке 17. Свойства параметра , отвечающего за объём информации о вершине графа, приведены на рисунке 18.

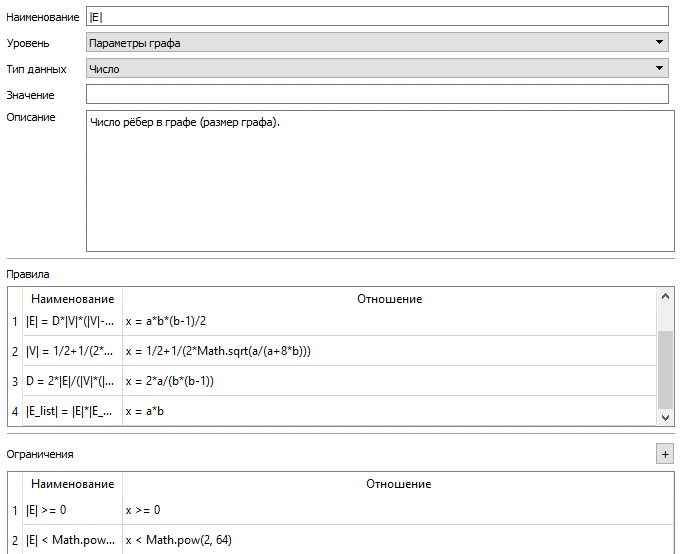


Рисунок 15 — Свойства параметра

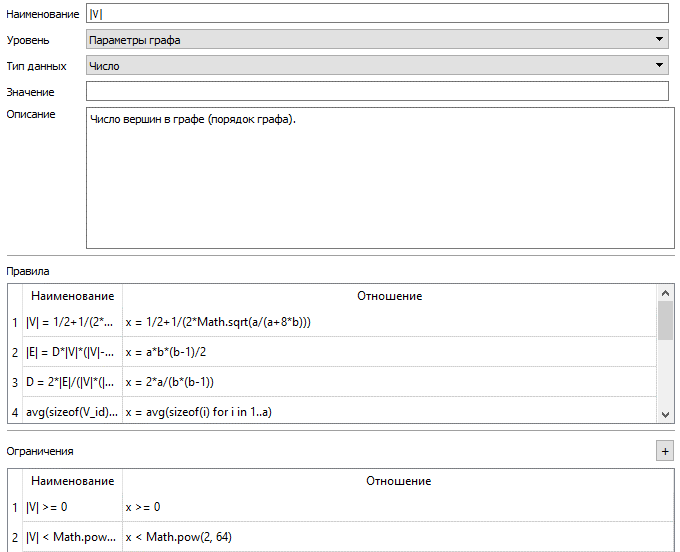


Рисунок 16 — Свойства параметра

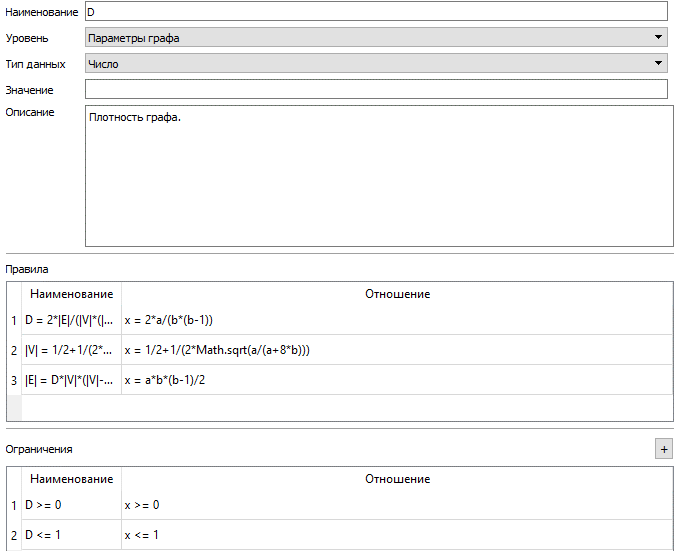


Рисунок 17 — Свойства параметра

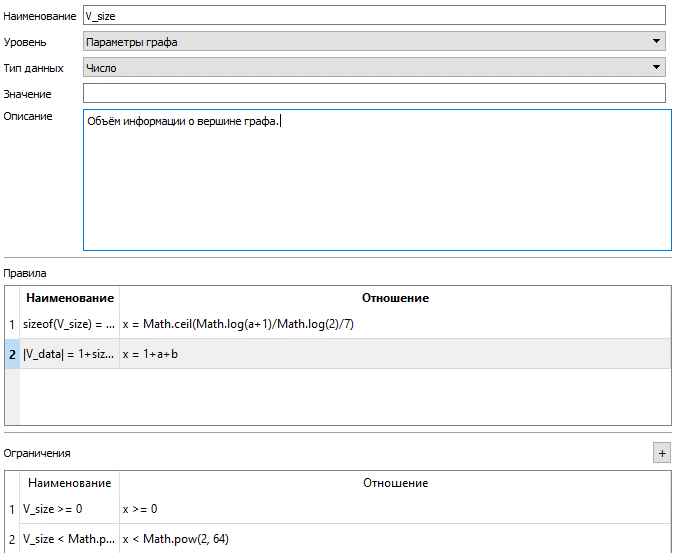


Рисунок 18 — Свойства параметра

Далее был реализован класс «Векторное представление». Данный класс собирает в себе все параметры и формулы, описанные в подпункте 2.4.1.5. Свойства параметра , отвечающего за информационный объём графа с векторным представлением связей, приведены на рисунке 19. Свойства параметра , отвечающего за размер модели классификации наличия связи в байтах, приведены на рисунке 20. Свойства параметра , отвечающего за вероятность ложноотрицательного срабатывания модели классификации наличия связи, приведены на рисунке 21. Свойства параметра , отвечающего за вероятность ложноположительного срабатывания модели классификации наличия связи, приведены на рисунке 22. Свойства параметра , отвечающего за размер векторного представления связей узла в байтах, приведены на рисунке 23.

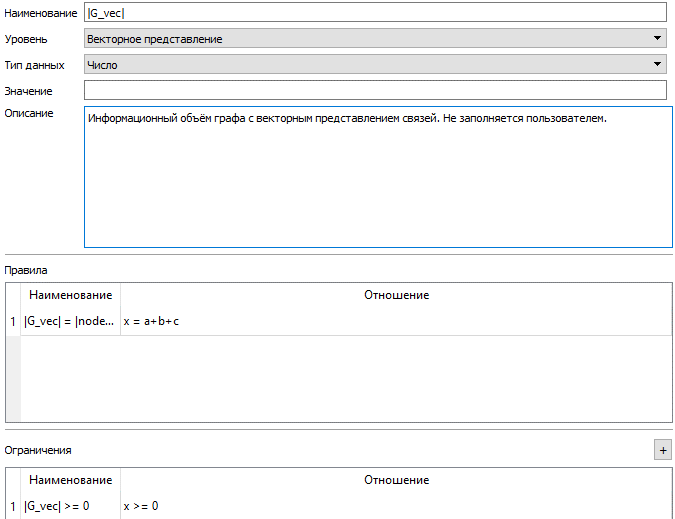


Рисунок 19 — Свойства параметра

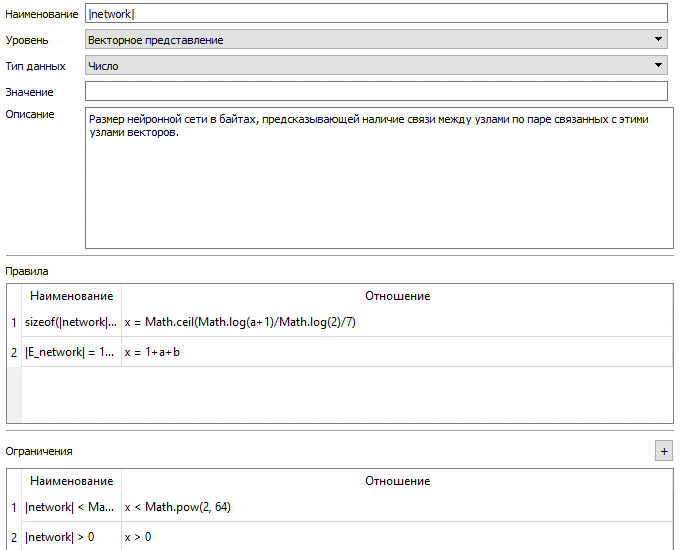


Рисунок 20 — Свойства параметра

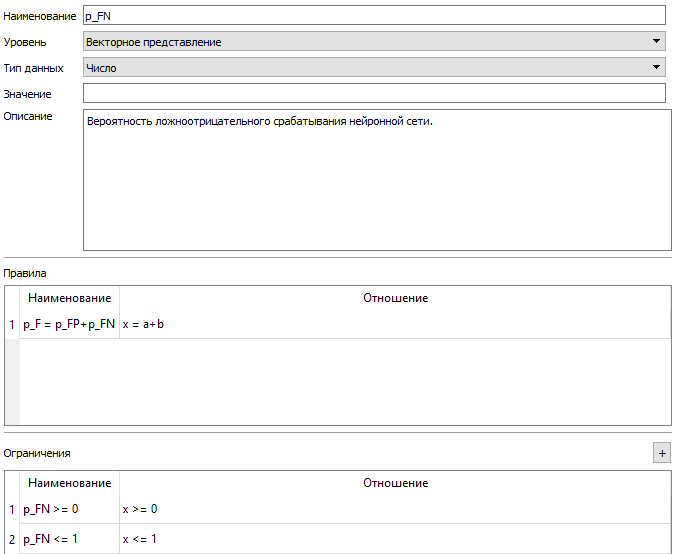


Рисунок 21 — Свойства параметра

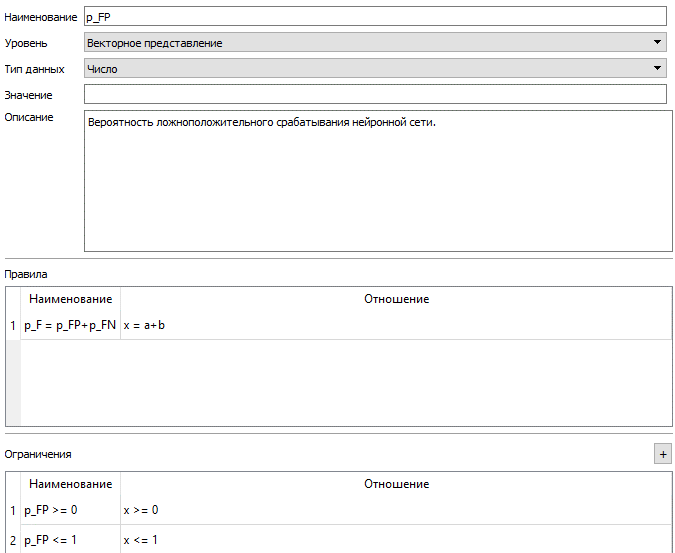


Рисунок 22 — Свойства параметра

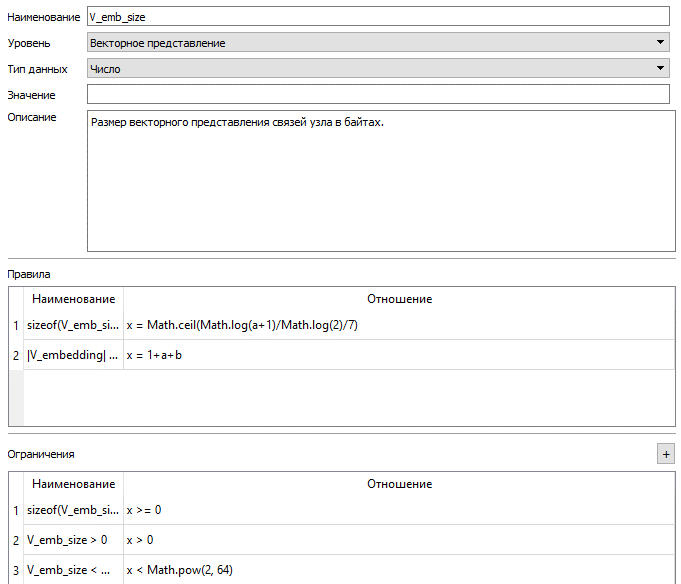


Рисунок 23 — Свойства параметра

Для возможности переиспользования отношений был выделен отдельный класс «Промежуточные параметры векторного представления», содержащий в себе параметры, не описанные в подпункте 2.4.1.5, но необходимые для более понятной реализации модели. Свойства параметра , отвечающего за информационный объём массива сообщений о связях, неправильно определяемых моделью классификации наличия связи, приведены на рисунке 24. Свойства параметра , отвечающего за информационный объём поля модели классификации наличия связи, приведены на рисунке 25. Свойства параметра , отвечающего за информационный объём массива вершин графа с учётом векторного представления связей, приведены на рисунке 26. Свойства параметра , отвечающего за информационный объём поля векторного представления связей одного узла, приведены на рисунке 27. Свойства параметра , отвечающего за информационный объём всех полей вершины графа статического размера с учётом векторного представления, приведены на рисунке 28. Свойства параметра , отвечающего за максимальный информационный объём информации о вершине графа с учётом векторного представления, приведены на рисунке 29. Свойства параметра , отвечающего за вероятность ложного срабатывания модели классификации наличия связи, приведены на рисунке 30. Свойства параметра , отвечающего за информационный объём размера модели классификации наличия связи, приведены на рисунке 31. Свойства параметра , отвечающего за информационный объём размера векторного представления связей узла в байтах, приведены на рисунке 32. Пример правила, связывающего параметры вычисления размера графа с векторным представлением связей, приведён на рисунке 33.

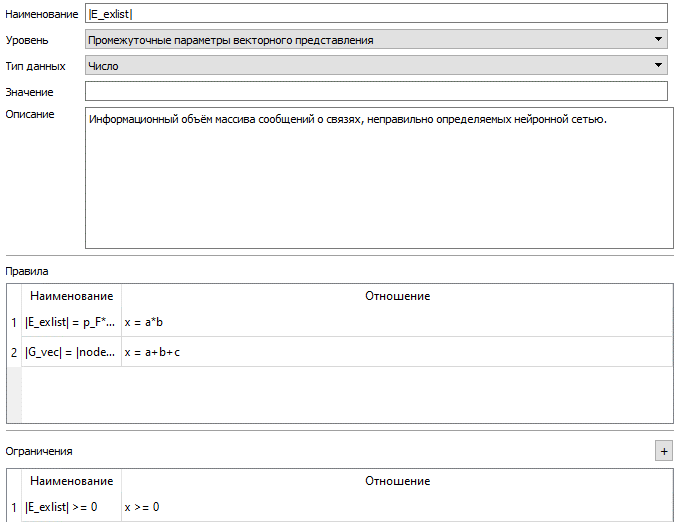


Рисунок 24 — Свойства параметра

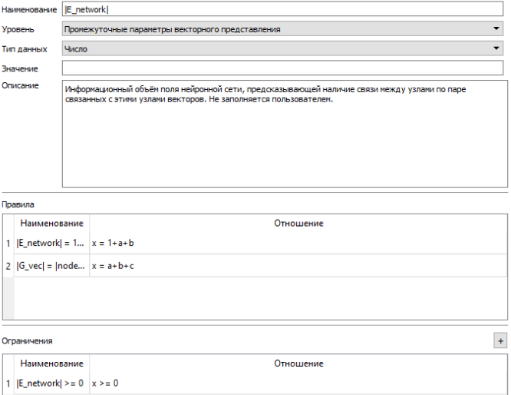


Рисунок 25 — Свойства параметра

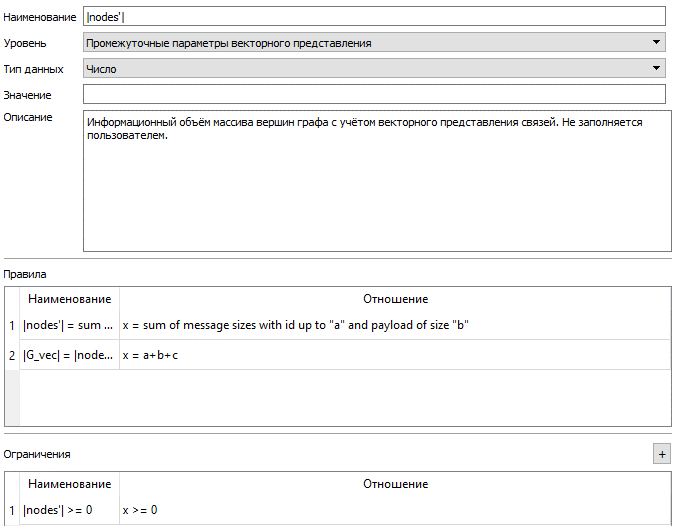


Рисунок 26 — Свойства параметра

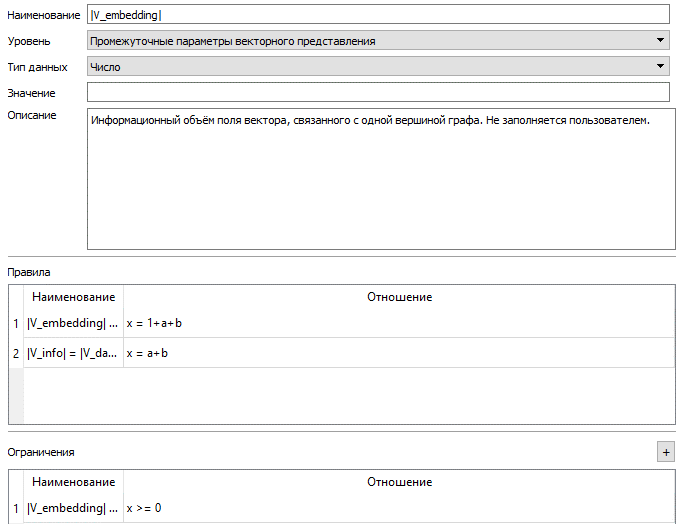


Рисунок 27 — Свойства параметра

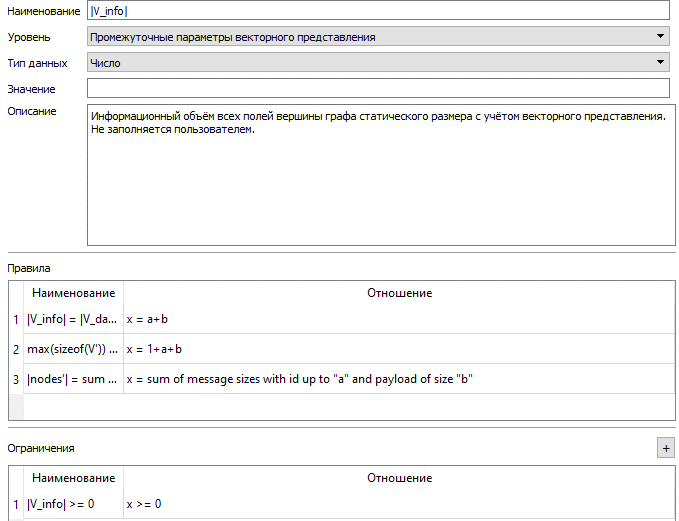


Рисунок 28 — Свойства параметра

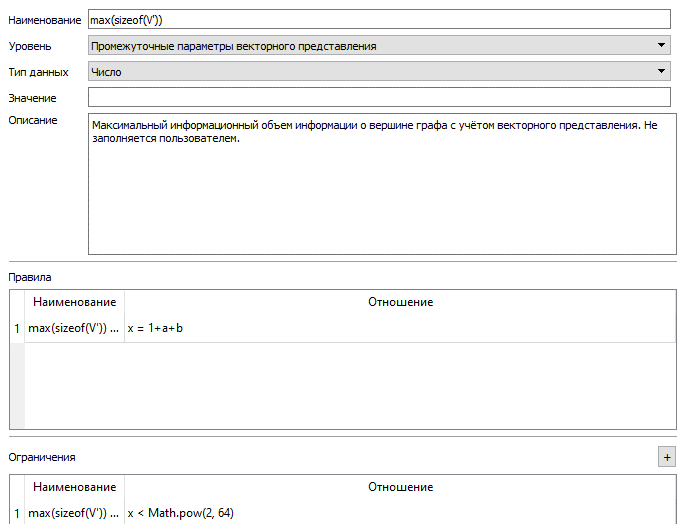


Рисунок 29 — Свойства параметра

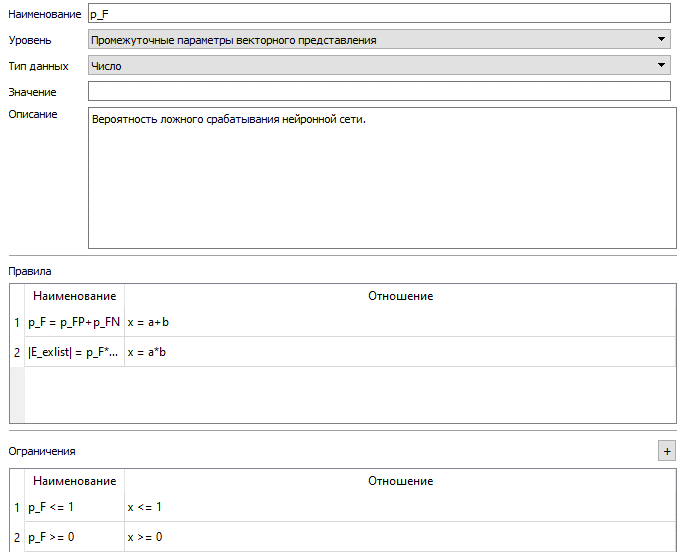


Рисунок 30 — Свойства параметра

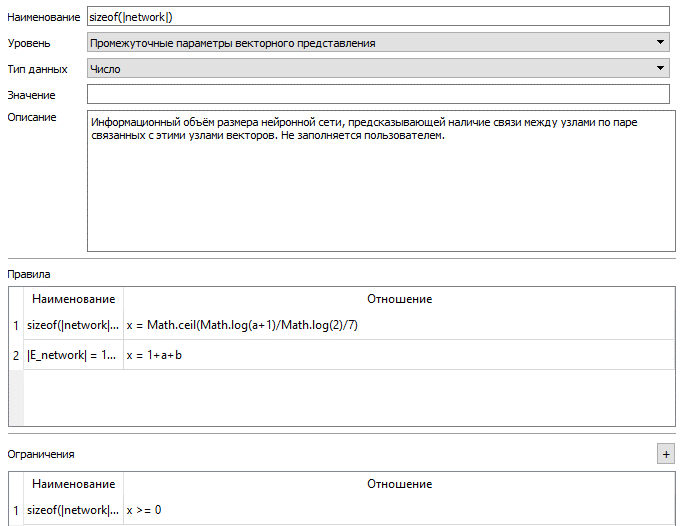


Рисунок 31 — Свойства параметра

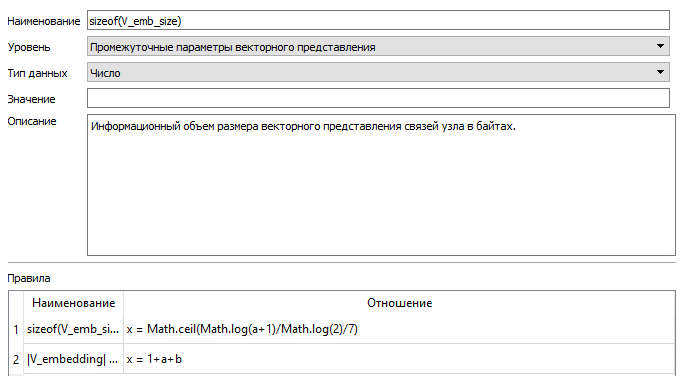


Рисунок 32 — Свойства параметра

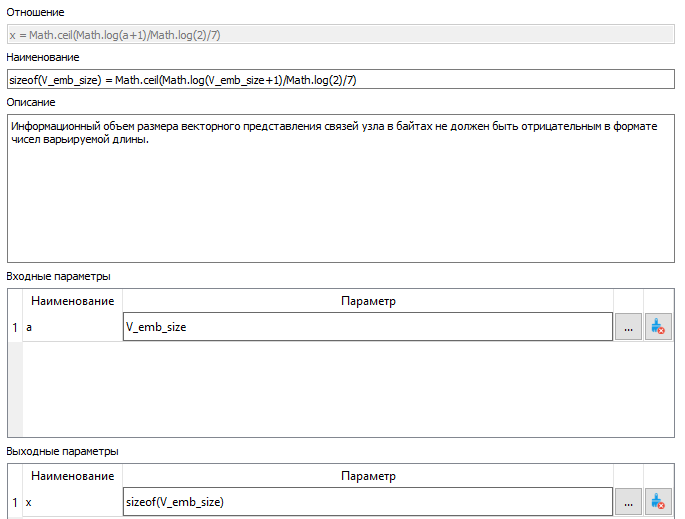


Рисунок 33 — Пример правила, связывающего параметры вычисления размера графа с векторным представлением связей

Далее был реализован класс «Матрица смежности». Данный класс собирает в себе все параметры и формулы, описанные в подпункте 2.4.1.3. Свойства параметра , отвечающего за информационный объём графа с представлением связей в виде матрицы смежности, приведены на рисунке 34.

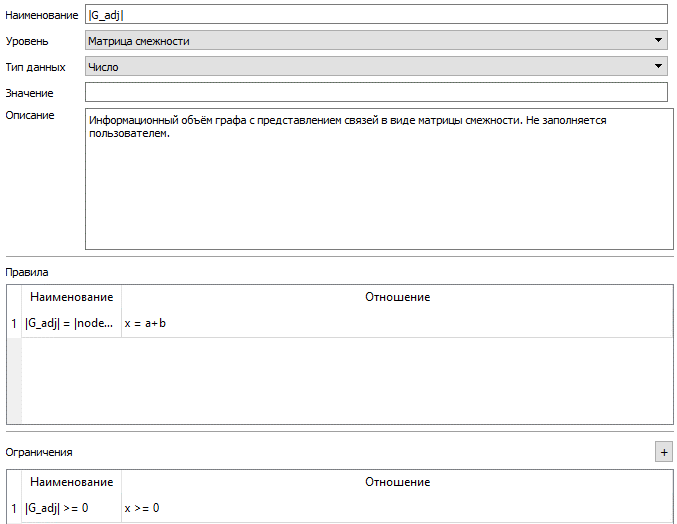


Рисунок 34 — Свойства параметра

Для возможности переиспользования отношений был выделен отдельный класс «Промежуточные параметры матрицы смежности», содержащий в себе параметры, не описанные в подпункте 2.4.1.3, но необходимые для более понятной реализации модели. Свойства параметра , отвечающего за информационный объём поля матрицы смежности, приведены на рисунке 35. Свойства параметра , отвечающего за квадрат числа вершин в графе, приведены на рисунке 36. Свойства параметра , отвечающего за информационный объём размера квадрата числа вершин в графе, приведены на рисунке 37. Пример правила, связывающего параметры вычисления размера графа с матрицей смежности, приведён на рисунке 38.

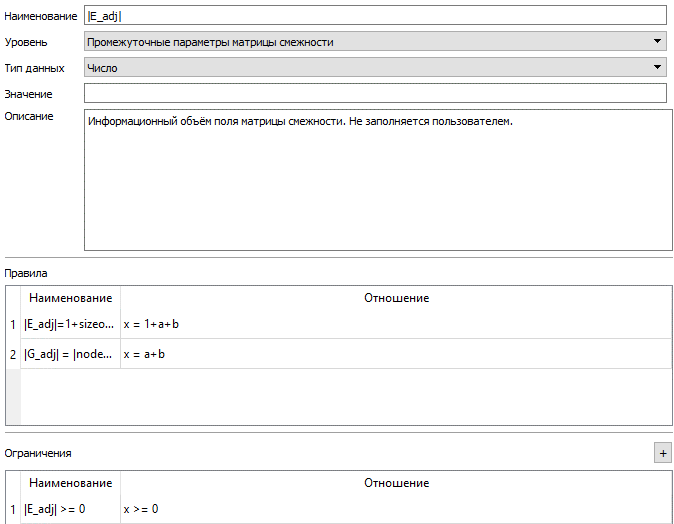


Рисунок 35 — Свойства параметра

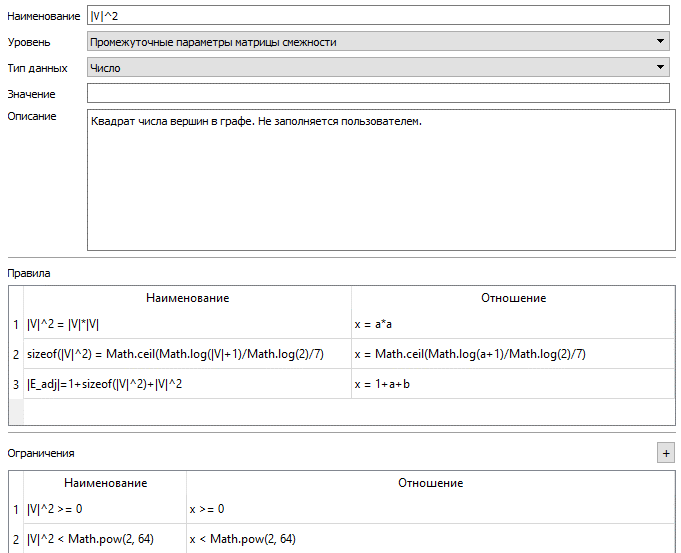


Рисунок 36 — Свойства параметра

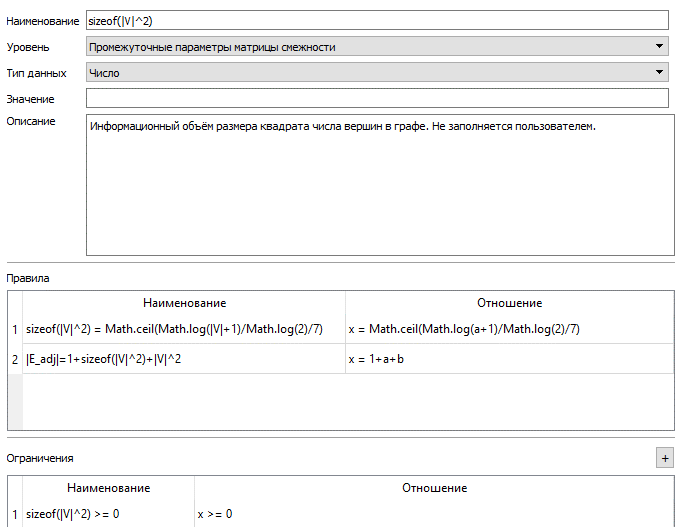


Рисунок 37 — Свойства параметра

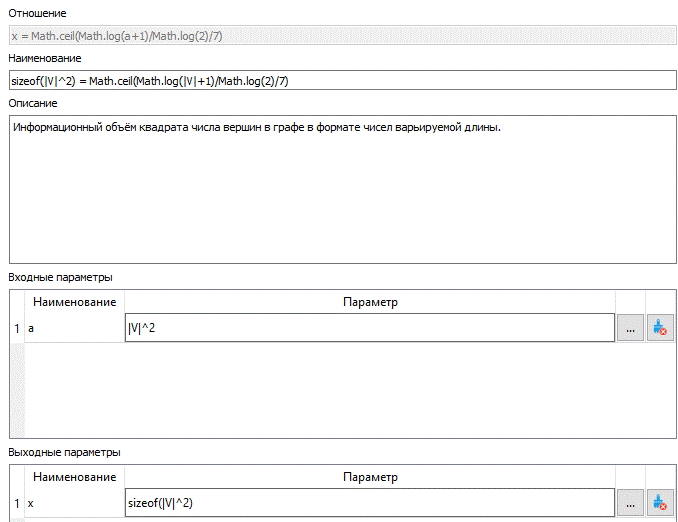


Рисунок 38 — Пример правила, связывающего параметры вычисления размера графа с матрицей смежности

Далее был реализован класс «Список связей». Данный класс собирает в себе все параметры и формулы, описанные в подпункте 2.4.1.4. Свойства параметра , отвечающего за информационный объём графа с представлением связей в виде списка связей, приведены на рисунке 39.

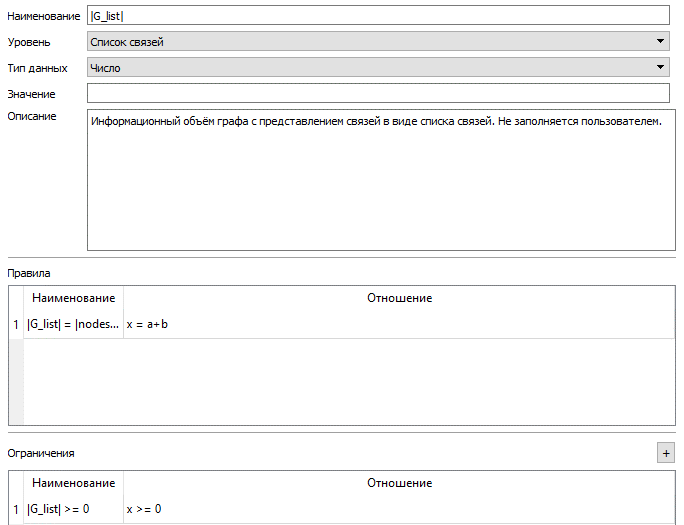


Рисунок 39 — Свойства параметра

Для возможности переиспользования отношений был выделен отдельный класс «Промежуточные параметры матрицы смежности», содержащий в себе параметры, не описанные в подпункте 2.4.1.3, но необходимые для более понятной реализации модели. Свойства параметра , отвечающего за информационный объём сообщения об одной связи, приведены на рисунке 40. Свойства параметра , отвечающего за информационный объём массива сообщений о связях, приведены на рисунке 41. Свойства параметра , отвечающего за информационный объём сообщения о связи с заголовком, приведены на рисунке 42. Свойства параметра , отвечающего за среднее арифметическое размера идентификатора вершины, приведены на рисунке 43. Свойства параметра , отвечающего за информационный объём размера сообщения об одной связи, приведены на рисунке 44. Пример правила, связывающего параметры вычисления размера графа со списком связей, приведён на рисунке 45.

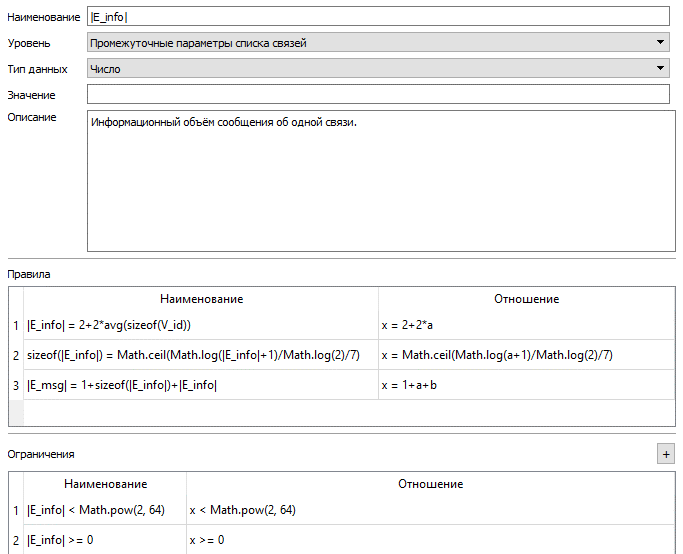


Рисунок 40 — Свойства параметра

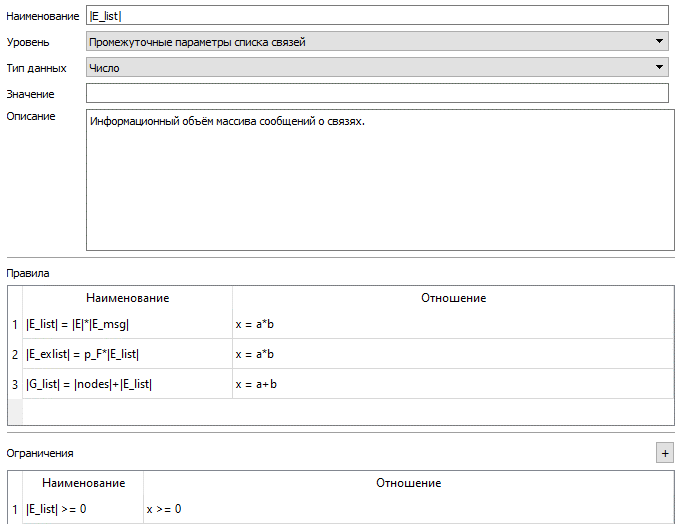


Рисунок 41 — Свойства параметра

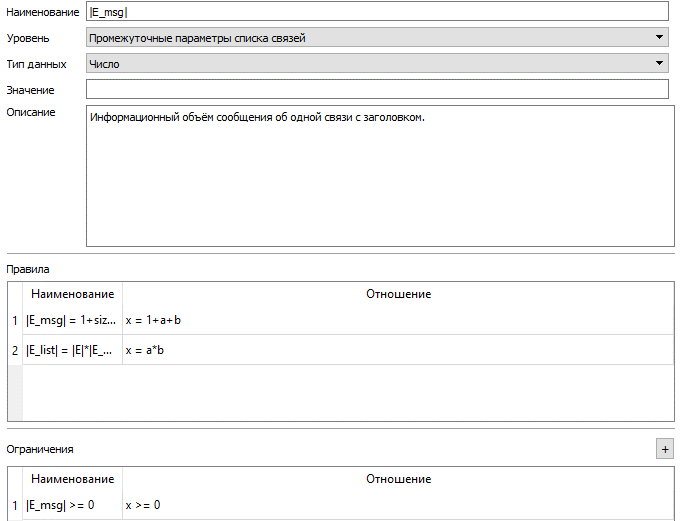


Рисунок 42 — Свойства параметра

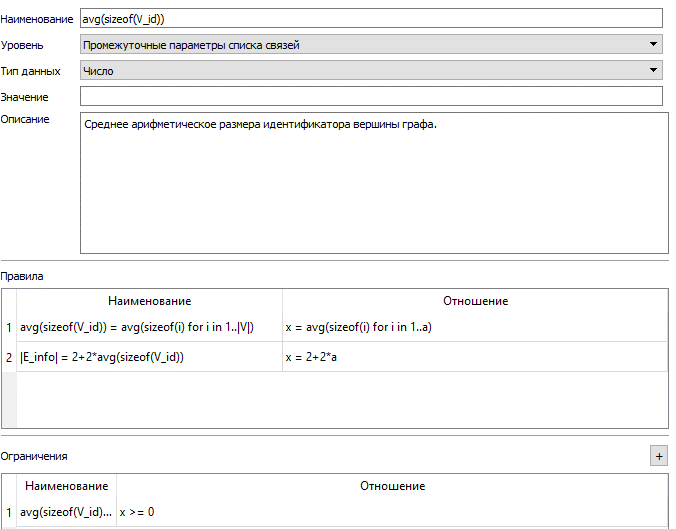


Рисунок 43 — Свойства параметра

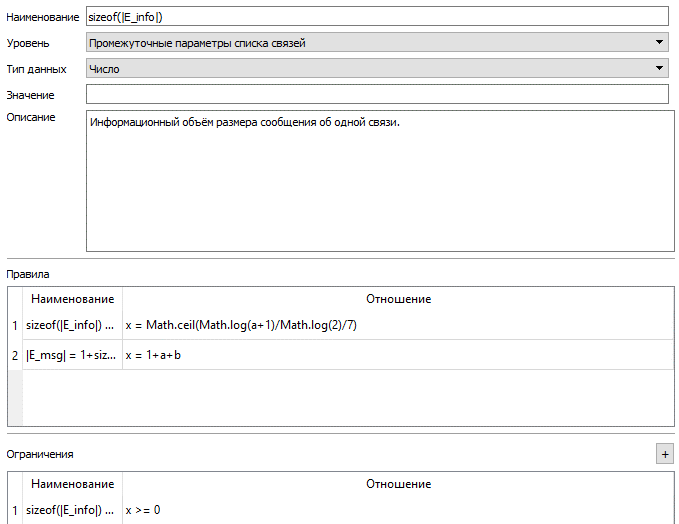


Рисунок 44 — Свойства параметра

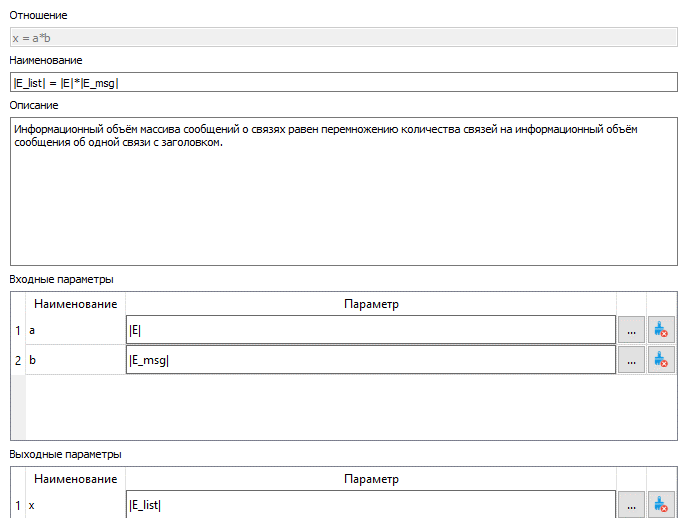


Рисунок 45 — Пример правила, связывающего параметры вычисления размера графа со списком связей

Также для параметров, использующихся всеми тремя способами представления связей, был выделен отдельный класс «Общие промежуточные параметры». Свойства параметра , отвечающего за информационный объём массива вершин графа, приведены на рисунке 46. Свойства параметра , отвечающего за информационный объём поля дополнительной информации о вершине графа, приведены на рисунке 47. Свойства параметра , отвечающего за максимальный информационный объём информации о вершине графа, приведены на рисунке 48. Свойства параметра , отвечающего за информационный объём максимального идентификатора вершины графа, приведены на рисунке 49. Свойства параметра , отвечающего за информационный объём длины дополнительной информации о вершине графа, приведены на рисунке 50.

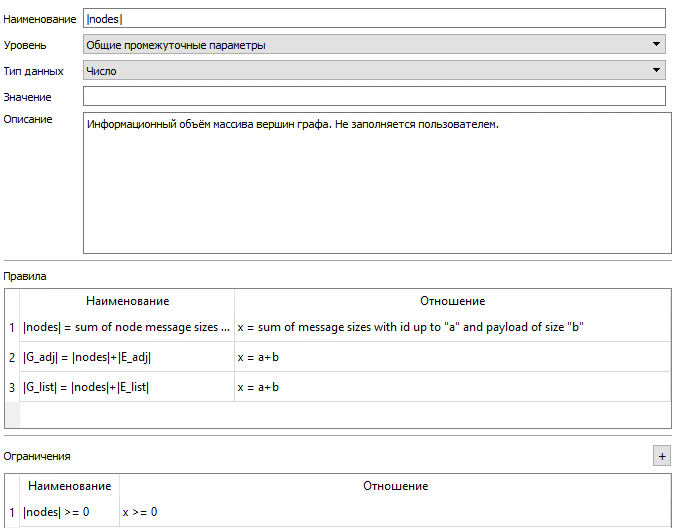


Рисунок 46 — Свойства параметра

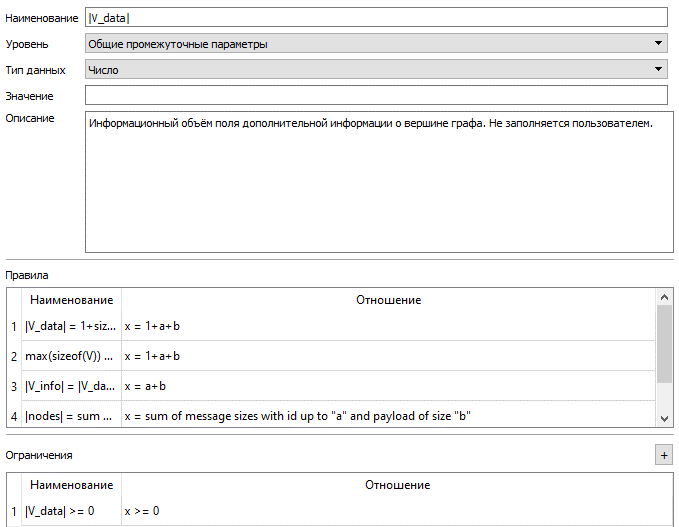


Рисунок 47 — Свойства параметра

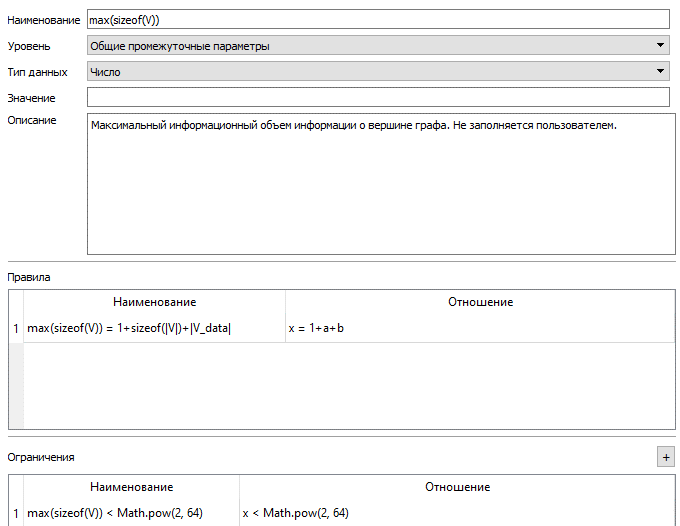


Рисунок 48 — Свойства параметра

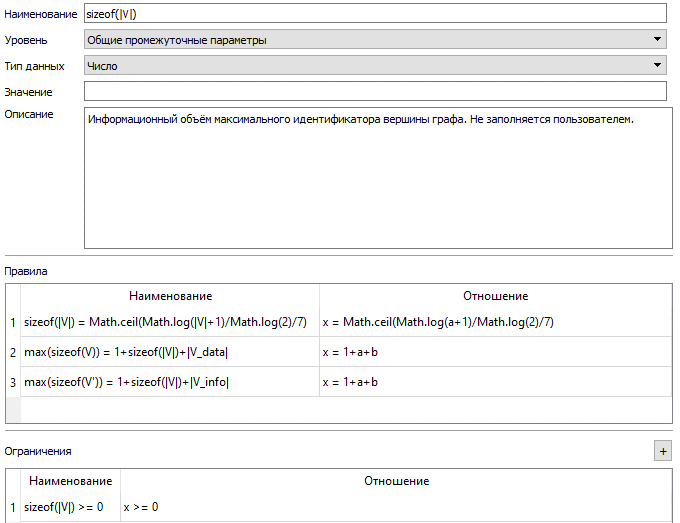


Рисунок 49 — Свойства параметра

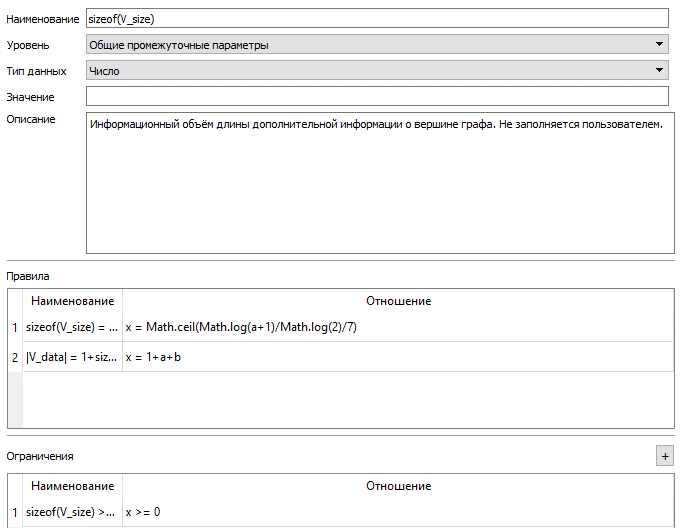


Рисунок 50 — Свойства параметра

Таким образом модель оценки объёма графа в КЭСМИ была реализована. Было произведено тестирование модели, введённые и полученные данные изображены на рисунке 51. Граф решения данной модели приведен на рисунке 52.

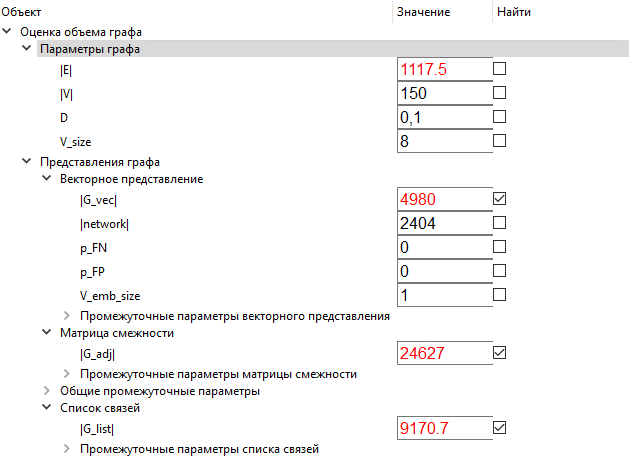


Рисунок 51 — Тестирование модели оценки объёма графа в КЭСМИ

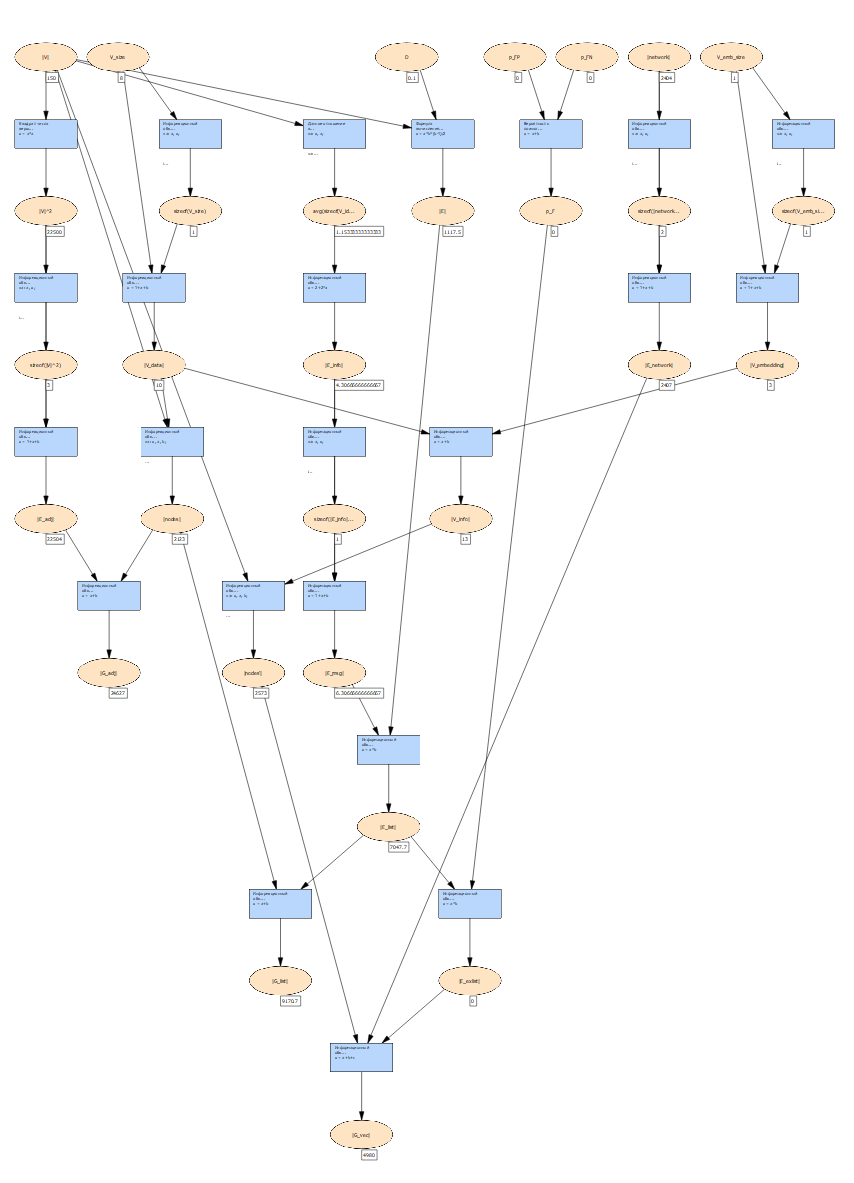


Рисунок 52 — Граф решения модели оценки размеров графа в КЭСМИ

## Оценка границ применимости разработанного метода сжатия графов

Моделирование было выполнено с параметрами, полученными в ходе обучения нейронной сети на специально сгенерированном графе, описанном в подразделе 3.1 [73]. Таким образом, объём нейронной сети был равен 2404 байта, объём векторного представления связей вершины был равен 1 байту (8 битам), вероятность ложноположительного срабатывания нейронной сети и вероятность ложноотрицательного срабатывания нейронной сети были равны нулю. Объём информации о вершине графа не столь важен, поскольку от него не зависит объём представлений рёбер и он, по сути, обозначает константу, прибавляемую к объёму представлений рёбер; в ходе моделирования этот параметр был равен 8 байтам.

Были рассчитаны размеры графа в различных представлениях в зависимости от числа вершин в графе и от плотности. Зависимость объём графа в векторном представлении от числа вершин и плотности изображена на рисунке 53. Зависимость объём графа с представлением связей в виде матрицы смежности от числа вершин и плотности изображена на рисунке 54. Зависимость объём графа с представлением связей в виде списка связей от числа вершин и плотности изображена на рисунке 55. Число вершин в графе и плотность графа изменялись с целью определения области, в которой векторное представление рёбер становится более эффективным. При (что примерно соответствует графу, рассмотренному в подразделе 3.1) векторное представление становится наиболее эффективным, если в графе больше сотни вершин, как это видно на рисунке 56. Был получен ожидаемый квадратичный рост для объёмов матрицы смежности и списка рёбер (отличие которых во многом сводится к константе перед квадратом числа вершин) и практически линейный рост объёма векторного представления рёбер.

Рисунок 53 — Зависимость объём графа в векторном представлении от числа вершин и плотности

Рисунок 54 — Зависимость объём графа с представлением связей в виде матрицы смежности от числа вершин и плотности

Рисунок 55 — Зависимость объём графа с представлением связей в виде списка связей от числа вершин и плотности

Рисунок 56 — Зависимость информационного объёма графа в различных представлениях от числа вершин в графе (при )

Также был рассмотрен случай : в графе содержится ровно половина всех возможных рёбер. Зависимости изображены на рисунке 57. Дальнейшее увеличение плотности графа бессмысленно, поскольку в этом случае становится эффективнее хранить не список присутствующих рёбер, а список отсутствующих рёбер, что сводит эту задачу к случаю, когда . Видно, что увеличение плотности графа влияет только на объём списка рёбер, что логично: остальные методы зависят только от количества вершин в графе, тогда как объём списка рёбер линейно зависит от плотности графа (то есть его рост можно оценить как ).

Рисунок 57 — Зависимость информационного объёма графа в различных представлениях от числа вершин в графе (при )

Таким образом предложенный метод сжатия графов на основе векторного представления рёбер эффективен в тех случаях, когда число вершин измеряется сотнями (или большими порядками). Для достаточно плотных графов векторное представление становится эффективнее списка связей, если в графе содержится от 50 вершин. Если модель классификации наличия связи неидеально предсказывает наличие рёбер по векторным представлениям, возможна компенсация ошибок с помощью списков ошибочных рёбер, как было описано в пункте 2.3.7, в таком случае граница применимости предлагаемого метода дополнительно смещается в сторону увеличения числа узлов.

# Анализ эффективности разработанного метода сжатия графа

## Исходный граф для сжатия

С целью демонстрации возможности сжатия графа на основе векторного представления был сгенерирован граф, аналогичный изображенному на рисунке 58. Узлы изображены точками, связи — линиями. В центре изображения находятся 7 связанных друг с другом центральных узлов, с каждым из центральных узлов связан кластер из *S* связанных друг с другом узлов. На данном рисунке , но дальше рассматриваются графы с  вплоть до 500 (с шагом 20).

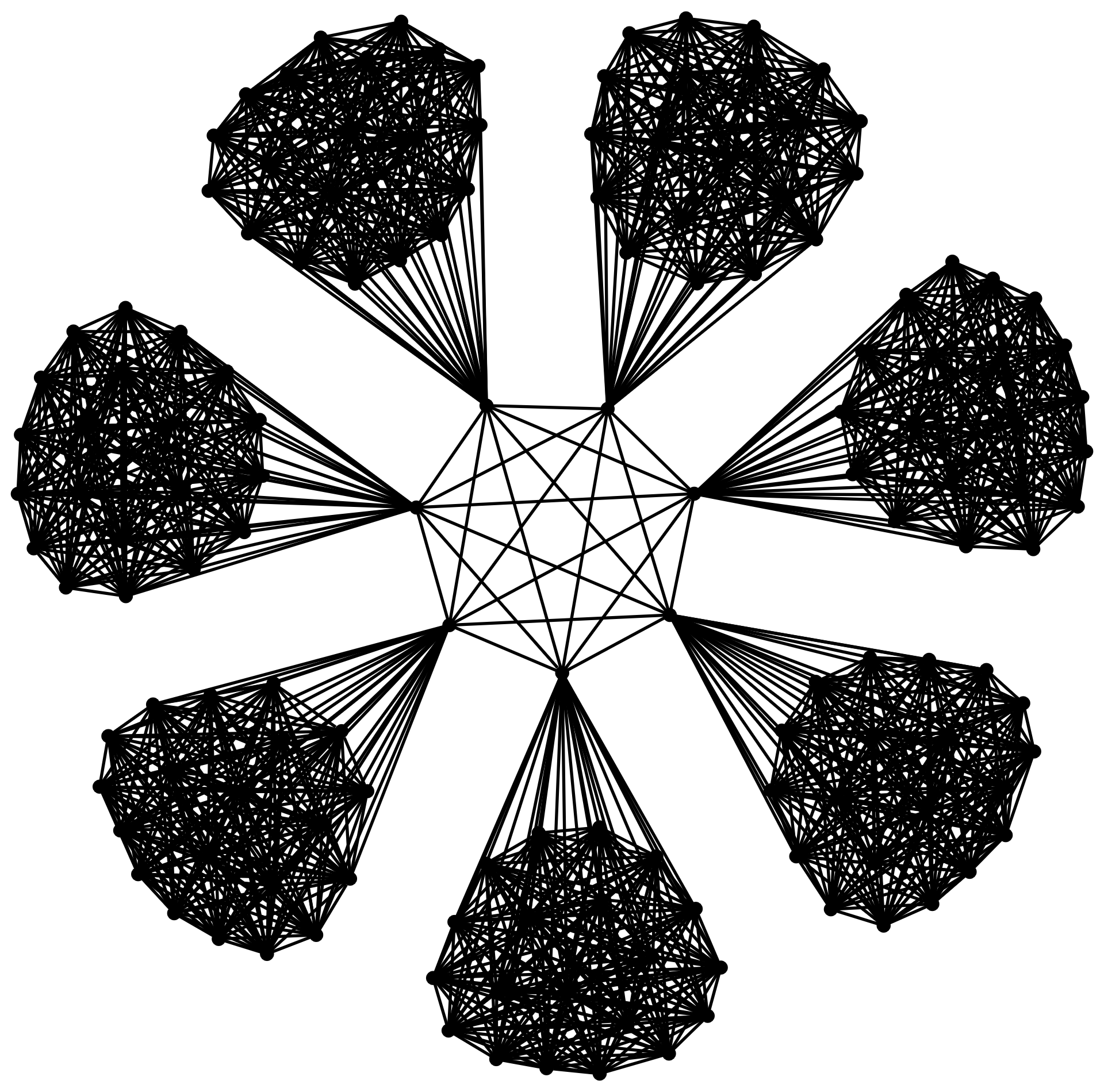


Рисунок 58 — Изображение одного из сгенерированных графов

Данный граф специально сформирован таким образом, чтобы хорошо подходить для векторного представления из восьми элементов [73].

## Анализ результатов сжатия

Была произведена оценка объемов данных графа в зависимости от его представления. График объема данных графа в зависимости от представления и количества узлов графа приведен на рисунке 59. Для лучшего понимания скорости роста векторного представления тот же график приведен в логарифмическом масштабе на рисунке 60.

Рисунок 59 — Объем данных графа в зависимости от представления

Рисунок 60 — Объем данных графа в зависимости от представления,  
логарифмический масштаб

Однако очевидно, что при практическом применении данные графа будут передаваться в сжатом виде. Для оценки качества сжатия была использована утилита xz из пакета XZ Utils [74] с максимальными параметрами сжатия. Данная утилита использует модифицированный алгоритм цепи Лемпеля-Зив-Маркова, который на данный момент дает наилучшее сжатие без потерь за практически применимое время. График объема сжатых данных графа в зависимости от представления и количества узлов графа приведен на рисунке 61. Для лучшего понимания скорости роста векторного представления тот же график приведен в логарифмическом масштабе на рисунке 62.

Рисунок 61 — Объем сжатых данных графа в зависимости от представления

Рисунок 62 — Объем сжатых данных графа в зависимости от представления,  
логарифмический масштаб

На последнем графике видно, что для малых графов имеет смысл использовать представление графа на основе списка связей, однако с ростом числа узлов в графе векторное представление начинает становиться выгоднее.

## Апробация алгоритма на других графах

Алгоритм был опробован на графе составов команд Чемпионата мира по футболу 2014 [52]. Данный граф состоит из 736 вершин и 9228 рёбер. Каждой вершине соответствует член команды Чемпионата мира по футболу 2014. Футболисты из одной национальной команды и из одного клуба связаны. Граф изображен на рисунке 63.

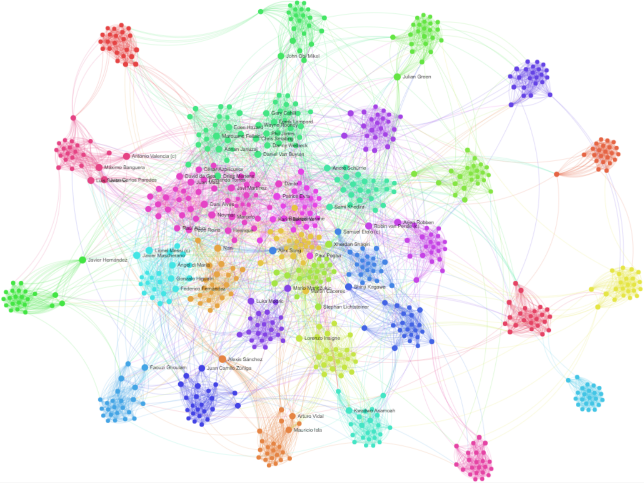


Рисунок 63 — Граф составов команд Чемпионата мира по футболу 2014

Список связей для данного графа в формате, описанном в пункте 2.3.1, занимает порядка 64 килобайтов. Варианты сжатия графа составов команд Чемпионата мира по футболу 2014 приведены в таблице 1. Видно, что все представленные варианты векторного представления хранят граф эффективнее, чем классический список связей, причём эффективность представления сильно зависит от параметров разработанного метода сжатия графов.

Таблица 1  
Варианты сжатия графа составов команд Чемпионата мира по футболу 2014

|  |  |  |  |
| --- | --- | --- | --- |
| Номер варианта реализации | В1 | В2 | В3 |
| Количество элементов векторного представления | 16 | 512 | 384 |
| Функция активации последнего слоя модели генерации векторного представления | ReLU | Бинаризация | Бинаризация |
| Объём векторного представления в байтах | 16 | 64 | 48 |
| Точность представления | 99,4% | 99,97% | 99,95% |
| Объём модели в килобайтах | 2,5 | 4 | 3 |
| Объём связей в векторном представлении без компенсации ошибок, в килобайтах | 14 | 50 | 37,5 |
| Объём связей в векторном представлении с компенсацией ошибок, в килобайтах | 22 | 51 | 39,5 |

Кроме того, были опробованы различные графы, сгенерированные алгоритмом R-MAT. Данный алгоритм позволяет быстро сгенерировать реалистичный граф с заданными параметрами [75] [76]. Результат работы разработанного метода сжатия графов стабильно оказывался самим эффективным информационным представлением для графов с параметрами, определенными в подразделе 2.5.

ЗАКЛЮЧЕНИЕ

В данной работе был предложен метод сжатия графов на основе векторного представления связей. Этот метод позволяет сохранить большую часть связей в графе и, возможно, получить лишние связи, которых не было изначально. В зависимости от области применения подобный результат может оказаться даже преимуществом, так как алгоритм открывает новые, ранее незамеченные связи, и появляется возможность для их исследования. С другой стороны, есть возможность предварительно проверить точность векторного представления связей в графе, и составить списки исключений в случае, когда необходимо сжатие без потерь.

Для реализации предложенного метода сжатия графов были найдены подходящие структуры искусственных нейронных сетей для решения задач генерации векторного представления и классификации наличия связей, а также был создан подходящий формат данных для хранения графа как в традиционном виде (в виде списка связей), так и в векторном представлении для удобства сравнения представлений графов.

Было проведено сравнение разработанного метода сжатия графов на основе векторного представления с традиционными способами хранения графов, такими как список связей и матрица смежности. Было выявлено, что предлагаемый метод сжатия графов применим для достаточно плотных и больших графов.

Была показана эффективность разработанного метода сжатия графов для различных графов. Таким образом были выполнены все поставленные задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Komorowski M. a history of storage cost (update) [Electronic resource] // matt komorowski. 2014. URL: http:/​/​www.mkomo.com/​cost-per-gigabyte-update (accessed: 11.10.2018).

2. Scales J. What Happened When I Peeked Into My Node\_Modules Directory [Electronic resource] // Medium. 2016. URL: https:/​/​medium.com/​s/​silicon-satire/​i-peeked-into-my-node-modules-directory-and-you-wont-believe-what-happened-next-b89f63d21558 (accessed: 13.10.2018).

3. Gentle J. Electron is flash for the desktop [Electronic resource] // Seph. 2016. URL: https:/​/​josephg.com/​blog/​electron-is-flash-for-the-desktop/ (accessed: 14.10.2018).

4. Wilde T. How game sizes got so huge, and why they'll get even bigger [Electronic resource] // PC Gamer. 2018. URL: https:/​/​www.pcgamer.com/​how-game-sizes-got-so-huge-and-why-theyll-get-even-bigger/ (accessed: 18.10.2018).

5. Bergmans W. Summary of the multiple file compression benchmark tests [Electronic resource] // Maximum Compression. 2011. URL: https:/​/​www.maximumcompression.com/​data/​summary\_mf.php (accessed: 19.11.2018).

6. Henecka W., Roughan M. Lossy Compression of Dynamic, Weighted Graphs // 2015 3rd International Conference on Future Internet of Things and Cloud. 2015. pp. 427–434.

7. Gilbert A.C., Levchenko K. Compressing Network Graphs // Proceedings of the LinkKDD workshop at the 10th ACM Conference on KDD. 2004. Vol. 124. pp. 1–12.

8. Харари Ф. Теория графов. М.: Мир, 1973. 300 с.

9. Математический энциклопедический словарь. М.: Советская энциклопедия, 1988. 845 с.

10. Mikolov T., Sutskever I., Chen K., Corrado G., Dean J. Distributed Representations of Words and Phrases and Their Compositionality // Proceedings of the 26th International Conference on Neural Information Processing Systems. 2013. Vol. 2. pp. 3111–3119.

11. Lebret R., Collobert R. Word Embeddings through Hellinger PCA // Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics. 2014. pp. 482–490.

12. Levy O., Goldberg Y. Linguistic Regularities in Sparse and Explicit Word Representations // Proceedings of the Eighteenth Conference on Computational Language Learning. 2014. pp. 171–180.

13. Gao Z., Fu G., Ouyang C., Tsutsui S., Liu X., Yang J., Gessner C., Foote B., Wild D., Ding Y., Yu Q. edge2vec: Representation learning using edge semantics for biomedical knowledge discovery // BMC Bioinformatics. 2019. Vol. 20:306. pp. 1–15.

14. Bandyopadhyay S., Biswas A., Murty N., Narayanam R. Beyond Node Embedding: A Direct Unsupervised Edge Representation Framework for Homogeneous Networks // arXiv. 2019. pp. 1–8.

15. Godec P. Graph Embeddings — The Summary [Electronic resource] // Towards Data Science. 2018. URL: https:/​/​towardsdatascience.com/​graph-embeddings-the-summary-cc6075aba007 (accessed: 08.05.2019).

16. Сафронов В. Где и как врубиться в эмбеддинги графов [Электронный ресурс] // Хабр. 2018. URL: https:/​/​habr.com/​company/​ods/​blog/​418727/ (дата обращения: 28.09.2018).

17. Nguyen D.Q. An overview of embedding models of entities and relationships for knowledge base completion [Electronic resource] // arXiv. 2018. URL: https:/​/​arxiv.org/​pdf/​1703.08098 (accessed: 11.10.2018).

18. Feng J., Huang M., Wang M., Zhou M., Hao Y., Zhu X. Knowledge Graph Embedding by Flexible Translation // Proceedings, Fifteenth International Conference on Principles of Knowledge Representation and Reasoning. 2016. pp. 557–560.

19. Besta M., Hoefler T. Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations // arXiv. 2018. pp. 1–54.

20. Cutting D., Pedersen J. Optimizations for Dynamic Inverted Index Maintenance // Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 1989. pp. 405–411.

21. Protocol Buffers. Encoding [Electronic resource] // Google Developers. 2020. URL: https:/​/​developers.google.com/​protocol-buffers/​docs/​encoding (accessed: 24.05.2020).

22. Зайцев В.В., Рыжков В.В., Сканави М.И. Элементарная математика. Повторительный курс. 3-е изд. М.: Наука, 1976. 591 с.

23. Adler M., Mitzenmacher M. Towards Compressing Web Graphs // Harvard Computer Science Group Technical Report TR-08-00. 2000. pp. 1–18.

24. Randall K.H., Stata R., Wiener J.L., Wickremesinghe R.G. The Link Database: Fast Access to Graphs of the Web // Proceedings of the Data Compression Conference. 2002. P. 122.

25. Asano Y., Ito T., Imai H., Toyoda M., Kitsuregawa M. Compact Encoding of the Web Graph Exploiting Various Power Laws // In: Advances in Web-Age Information Management. Berlin: Springer, 2003. pp. 37–46.

26. Webber J. A Programmatic Introduction to Neo4j // Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity. 2012. pp. 217–218.

27. Martínez-Bazan N., Gómez-Villamor S., Escalé-Claveras F. DEX: a High-Performance Graph Database Management System // 2011 IEEE 27th International Conference on Data Engineering Workshops. 2011. pp. 124–127.

28. Martínez-Bazan N., Águila-Lorente M.Á., Muntés-Mulero V., Dominguez-Sal D., Gómez-Villamor S., Larriba-Pey J.L. Efficient graph management based on bitmap indices // Proceedings of the 16th International Database Engineering & Applications Symposium. 2012. pp. 110–119.

29. Lehmann N., Pérez J. Implementing Graph Query Languages over Compressed Data Structures: A Progress Report // CEUR Workshop Proceedings. 2015. Vol. 1378. pp. 96–100.

30. Alvarez S., Brisaboa N.R., Ladra S., Pedreira Ó. A Compact Representation of Graph Databases // Proceedings of the Eighth Workshop on Mining and Learning with Graphs. 2010. pp. 18–25.

31. Khandelwal A., Yang Z., Ye E., Agarwal R., Stoica I. ZipG: A Memory-efficient Graph Store for Interactive Queries // Proceedings of the 2017 ACM International Conference on Management of Data. 2017. pp. 1149–1164.

32. Agarwal R., Khandelwal A., Stoica I. Succinct: Enabling Queries on Compressed Data // Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation. 2015. pp. 337–350.

33. Kang U., Tong H., Sun J., Lin C.Y., Faloutsos C. GBASE: A Scalable and General Graph Management System // Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. 2011. pp. 1091–1099.

34. Kang U., Tong H., Sun J., Lin C.Y., Faloutsos C. GBASE: an efficient analysis platform for large graphs // The VLDB Journal. 2012. Vol. 21. No. 5. pp. 637–650.

35. King D., Rossignac J. Guaranteed 3.67V bit encoding of planar triangle graphs // 11th Canadian Conference on Computational Geometry. 1999. pp. 146–149.

36. Farzan A., Munro J.I. Succinct Representations of Arbitrary Graphs // In: Algorithms - ESA 2008. Berlin: Springer, 2008. pp. 393–404.

37. Kannan S., Naor M., Rudich S. Implicit Representation of Graphs // Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. 1988. pp. 334–343.

38. Goyal P., Ferrara E. Graph Embedding Techniques, Applications, and Performance: A Survey // Knowledge-Based Systems. 2018. Vol. 151. pp. 78–94.

39. Hamilton W.L., Ying R., Leskovec J. Representation Learning on Graphs: Methods and Applications // Bulletin of the Technical Committee on Data Engineering. 2017. Vol. 40. No. 3. pp. 52–74.

40. Cai H., Zheng V.W., Chang K.C.C. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications // IEEE Transactions on Knowledge and Data Engineering. 2018. Vol. 30. No. 9. pp. 1616–1637.

41. Kutuzov A., Dorgham M., Oliynyk O., Biemann C., Panchenko A. Learning Graph Embeddings from WordNet-based Similarity Measures // Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics. 2019. pp. 125–135.

42. Liben-Nowell D., Kleinberg J. The Link-Prediction Problem for Social Networks // Journal of the American Society for Information Science and Technology. 2007. Vol. 58. No. 7. pp. 1019–1031.

43. Perozzi B., Al-Rfou R., Skiena S. DeepWalk: Online Learning of Social Representations // Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2014. pp. 701–710.

44. Grover A., Leskovec J. node2vec: Scalable Feature Learning for Networks // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016. pp. 855–864.

45. Wang D., Cui P., Zhu W. Structural Deep Network Embedding // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016. pp. 1225–1234.

46. Дистель Р. Теория графов. Новосибирск: Издательство института математики, 2002. 333 с.

47. Coleman T.F., Moré J.J. Estimation of Sparse Jacobian Matrices and Graph Coloring Blems // SIAM Journal on Numerical Analysis. 1983. Vol. 20. No. 1. pp. 187–209.

48. Rosenblatt F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain // Psychological Review. 1958. Vol. 65. No. 6. pp. 386–408.

49. Mitchell T.M. Machine Learning. 1st ed. Singapore: WCB/McGraw-Hill, 1997. 432 pp.

50. Nair V., Hinton G.E. Rectified Linear Units Improve Restricted Boltzmann Machines // Proceedings of the 27th International Conference on International Conference on Machine Learning. 807–814 2010.

51. Ramachandran P., Zoph B., Le Q.V. Searching for Activation Functions // Proceedings of the 6th International Conference on Learning Representations. 2018. pp. 1–13.

52. Vermeij A. World Cup Special: How are the World’s Top Players Connected? [Electronic resource] // Kenedict Innovation Analytics. 2014. URL: https:/​/​www.kenedict.com/​world-cup-special-how-are-the-worlds-top-players-connected/ (accessed: 20.06.2020).

53. Bridle J.S. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition // Neurocomputing. 1990. Vol. 68. pp. 227–236.

54. Liou C.Y., Cheng W.C., Liou J.W., Liou D.R. Autoencoder for words // Neurocomputing. 2014. Vol. 139. pp. 84–96.

55. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 800 pp.

56. Kingma D.P., Ba J.L. Adam: A Method for Stochastic Optimization // Proceedings of the 3rd International Conference on Learning Representations. 2015. pp. 1–15.

57. TensorFlow Core v2.2.0. tf.keras.optimizers.Adamax [Electronic resource] // TensorFlow. 2020. URL: https:/​/​www.tensorflow.org/​api\_docs/​python/​tf/​keras/​optimizers/​Adamax?version=nightly (accessed: 03.06.2020).

58. JCGM 200:2012. International vocabulary of metrology — Basic and general concepts and associated terms (VIM). 3rd ed. 2012. 91 pp.

59. Boyd S.P., Vandenberghe L. Convex Optimization. Cambridge: Cambridge University Press, 2004. 716 pp.

60. Тихонов А.Н. О некорректных задачах линейной алгебры и устойчивом методе их решения // Доклады Академии наук СССР. 1965. Т. 163. № 3. С. 591–594.

61. ISO/IEC 60559:2020. Floating-point arithmetic. 2020. 84 pp.

62. TensorFlow Model Optimization. Quantization aware training [Electrnoic resource] // TensorFlow. 2020. URL: https:/​/​www.tensorflow.org/​model\_optimization/​guide/​quantization/​training (accessed: 03.06.2020).

63. TensorFlow Lite [Electronic resource] // TensorFlow. 2020. URL: https:/​/​www.tensorflow.org/​lite/ (accessed: 02.06.2020).

64. FlatBuffers [Electronic resource] // GitHub. 2020. URL: https:/​/​google.github.io/​flatbuffers/ (accessed: 05.06.2020).

65. tensorflow/tensorflow. tensorflow/lite/schema/schema.fbs at master [Electronic resource] // GitHub. 2020. URL: https:/​/​github.com/​tensorflow/​tensorflow/​blob/​master/​tensorflow/​lite/​schema/​schema.fbs (accessed: 05.06.2020).

66. TensorFlow Core v2.2.0 [Electronic resource] // TensorFlow. 2020. URL: https:/​/​www.tensorflow.org/​api\_docs/​python/​tf?version=nightly (accessed: 30.06.2020).

67. Protocol Buffers [Electronic resource] // Google Developers. 2019. URL: https:/​/​developers.google.com/​protocol-buffers/ (accessed: 19.10.2019).

68. NumPy Manual [Electronic resource] // NumPy. 2020. URL: https:/​/​numpy.org/​doc/​stable/ (accessed: 30.06.2020).

69. Varlamov O.O. Wi!Mi Expert System Shell as the Novel Tool for Building Knowledge-Based Systems with Linear Computational Complexity // International Review of Automatic Control. 2018. Vol. 11. No. 6. pp. 314–325.

70. Варламов О.О. Миварные технологии как некоторые направления искусственного интеллекта // Проблемы искусственного интеллекта. 2015. Т. 1. № 0. С. 23–37.

71. Черненький В.М., Терехов В.И., Гапанюк Ю.Е. Структура гибридной интеллектуальной информационной системы на основе метаграфов // Нейрокомпьютеры: разработка, применение. 2016. № 9. С. 3–13.

72. Варламов О.О., Чибирова М.О., Хадиев А.М., Антонов П.Д., Сергушин Г.С., Шошев И.А., Назаров К.В. Практикум по созданию миварных экспертных систем. Учебное пособие. М.: Издательство НИИ МИВАР, 2016. 184 с.

73. Лещев А.О., Лясковский М.А., Мельников К.И. Оценка применимости векторного представления рёбер графа с целью уменьшения информационного объема графа // Наука без границ. 2020. Т. 46. № 6. С. 67–73.

74. XZ Utils [Electronic resource] // The Tukaani Project. 2018. URL: https:/​/​tukaani.org/​xz/ (accessed: 18.12.2019).

75. Chakrabarti D., Zhan Y., Faloutsos C. R-MAT: A Recursive Model for Graph Mining // Proceedings of the Fourth SIAM International Conference on Data Mining. 2004. pp. 442–446.

76. Khorasani F., Gupta R., Bhuyan L.N. Scalable SIMD-Efficient Graph Processing on GPUs // Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques. 2015. pp. 39–50.

631106

1. Техническое задание

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

|  |  |
| --- | --- |
| УТВЕРЖДАЮ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. | СОГЛАСОВАНО  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

**ИССЛЕДОВАНИЕ МЕТОДОВ СЖАТИЯ ГРАФОВ  
НА ОСНОВЕ ВЕКТОРНОГО ПРЕДСТАВЛЕНИЯ**

Техническое задание

(вид документа)

Писчая бумага

(вид носителя)

5

(количество листов)

|  |  |
| --- | --- |
|  | ИСПОЛНИТЕЛЬ  студент группы ИУ5-44М  \_\_\_\_\_\_\_\_\_\_\_ Лещев А. О.  «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

Москва — 2020 г.

СОДЕРЖАНИЕ

[1 Наименование 3](#_Toc44245385)

[2 Основание для проведения работы 3](#_Toc44245386)

[3 Исполнитель 3](#_Toc44245387)

[4 Цель работы 3](#_Toc44245388)

[5 Требования к решаемым задачам 4](#_Toc44245389)

[6 Этапы работы 4](#_Toc44245390)

[7 Документация, предъявляемая по окончанию работы 5](#_Toc44245391)

[8 Порядок приема работы 5](#_Toc44245392)

[9 Дополнительные условия 5](#_Toc44245393)

1 Наименование

Исследование методов сжатия графов на основе векторного представления.

2 Основание для проведения работы

Основанием для проведения данной работы является задание на выполнение выпускной квалификационной работы магистра, подписанное руководителем квалификационной работы и утверждённое заведующим кафедрой ИУ5 «Системы обработки информации и управления» Московского государственного технического университета имени Н.Э. Баумана.

3 Исполнитель

Студент Московского государственного технического университета имени Н.Э. Баумана группы ИУ5-44М Лещев Артем Олегович.

4 Цель работы

Целью работы является исследование возможности применения векторного представления графов для сжатия их информационного представления и разработка соответствующего метода сжатия графа на основе векторного представления.

5 Требования к решаемым задачам

В процессе выполнения работы должны быть решены следующие задачи:

1. Проведен обзор и анализ публикаций по теме векторного представления вершин графа.
2. Проведен анализ существующих методов сжатия графов.
3. Описан разработанный метод сжатия графов.
4. Разработана программная реализация описанного метода сжатия графов.
5. Разработана модель для оценки эффективности разработанного метода.
6. Описан процесс моделирования.
7. Проанализированы результаты проведенного моделирования.

6 Этапы работы

Основными этапами являются:

1. Критический анализ существующих подходов к решению задачи:
   1. Описание предметной области.
   2. Обзор и анализ публикаций по теме векторного представления вершин графа и методов сжатия графов.
   3. Определение направления исследования, формулировка цели и задачи.
2. Разработка метода сжатия графов:
   1. Изложение общих теоретических положений.
   2. Описание разработанного метода сжатия графов.
   3. Программная реализация метода сжатия графов.
   4. Разработка модели для оценки границ применимости разработанного метода.
3. Анализ результатов моделирования:
   1. Описание процесса моделирования.
   2. Анализ результатов проведенного моделирования.

7 Документация, предъявляемая по окончанию работы

По окончанию работы предоставляются:

* задание на выполнение выпускной квалификационной работы магистра;
* календарный план;
* расчетно-пояснительная записка;
* техническое задание;
* текст программы;
* графическая часть;
* рецензия на выпускную квалификационную работу магистра;
* акт проверки выпускной квалификационной работы.

8 Порядок приема работы

Прием работы осуществляется в установленном порядке по её окончании.

9 Дополнительные условия

Дополнительные условия не предусмотрены.

1. Вывод модели оценки размеров графа в консоль

Шаг № 0

Отношение: x = a\*a (Значение переменной равно квадрату значения другой переменной.)

Правило: |V|^2 = |V|\*|V| (Квадрат числа вершин в графе равен числу вершин в графе, умноженному на самого себя.)

Входные параметры:

|V|=150;

Формула:

x = a\*a

Результат: |V|^2=22500;

------------------------------------

Шаг № 1

Отношение: x = Math.ceil(Math.log(a+1)/Math.log(2)/7) (Расчёт длины числа в формате чисел варьируемой длины.)

Правило: sizeof(|V|^2) = Math.ceil(Math.log(|V|+1)/Math.log(2)/7) (Информационный объём квадрата числа вершин в графе в формате чисел варьируемой длины.)

Входные параметры:

|V|^2=22500;

Формула:

var a, x;

if (a != 0) {

x = Math.ceil(Math.log(a+1)/Math.log(2)/7);

} else {

x = 0;

}

Результат: sizeof(|V|^2)=3;

------------------------------------

Шаг № 2

Отношение: x = 1+a+b (Значение некоторых переменных является суммой других переменных плюс единица.)

Правило: |E\_adj|=1+sizeof(|V|^2)+|V|^2 (Информационный объём поля матрицы смежности равен сумме квадрата числа вершин в графе, размера квадрата и единицы.)

Входные параметры:

sizeof(|V|^2)=3;

|V|^2=22500;

Формула:

x = 1+a+b

Результат: |E\_adj|=22504;

------------------------------------

Шаг № 3

Отношение: x = Math.ceil(Math.log(a+1)/Math.log(2)/7) (Расчёт длины числа в формате чисел варьируемой длины.)

Правило: sizeof(V\_size) = Math.ceil(Math.log(V\_size+1)/Math.log(2)/7) (Информационный объём длины дополнительной информации о вершине графа в формате чисел варьируемой длины.)

Входные параметры:

V\_size=8;

Формула:

var a, x;

if (a != 0) {

x = Math.ceil(Math.log(a+1)/Math.log(2)/7);

} else {

x = 0;

}

Результат: sizeof(V\_size)=1;

------------------------------------

Шаг № 4

Отношение: x = 1+a+b (Значение некоторых переменных является суммой других переменных плюс единица.)

Правило: |V\_data| = 1+sizeof(V\_size)+V\_size (Информационный объём поля дополнительной информации о вершине графа равен сумме размера дополнительной информации о вершине графа, длины размера и единицы.)

Входные параметры:

sizeof(V\_size)=1;

V\_size=8;

Формула:

x = 1+a+b

Результат: |V\_data|=10;

------------------------------------

Шаг № 5

Отношение: x = sum of message sizes with id up to "a" and payload of size "b" (Данное отношение вычисляет общий информационный объём вложенных сообщений с увеличивающимся идентификатором вплоть до "a" и с данными размером "b".)

Правило: |nodes| = sum of node message sizes with id up to |V| and payload of size |V\_data| (Информационный объём массива вершин графа связей вычисляется как общий информационный объём вложенных сообщений узлов с увеличивающимся идентификатором вплоть до |V| и с данными размером |V\_data|.)

Входные параметры:

|V|=150;

|V\_data|=10;

Формула:

var x, a, b;

x = 0;

for (var i = 1; i <= 10; ++i) {

var power = i\*7;

var limit = Math.pow(2, power);

var prev\_limit = Math.pow(2, (i-1)\*7);

var payload\_size = 1+i+b;

var sizeof\_payload\_size = Math.ceil(Math.log(payload\_size+1)/Math.log(2)/7);

if (a < limit) {

var count = a-prev\_limit+1;

x += count\*(1+sizeof\_payload\_size+payload\_size);

break;

} else {

var count = limit-prev\_limit;

x += count\*(1+sizeof\_payload\_size+payload\_size);

}

}

Результат: |nodes|=2123;

------------------------------------

Шаг № 6

Отношение: x = a+b (Значение некоторых переменных является суммой других переменных.)

Правило: |G\_adj| = |nodes|+|E\_adj| (Информационный объём графа с представлением связей в виде матрицы смежности равен сумме информационного объёма массива вершин графа и информационного объёма поля матрицы смежности.)

Входные параметры:

|nodes|=2123;

|E\_adj|=22504;

Формула:

x = a+b

Результат: |G\_adj|=24627;

------------------------------------

Шаг № 7

Отношение: x = avg(sizeof(i) for i in 1..a) (Данное отношение вычисляет среднее арифметическое размера идентификатора вплоть до "a".)

Правило: avg(sizeof(V\_id)) = avg(sizeof(i) for i in 1..|V|) (Данное отношение вычисляет среднее арифметическое размера идентификатора вплоть до |V|.)

Входные параметры:

|V|=150;

Формула:

var x, a;

var sum = 0;

for (var i = 1; i <= 10; ++i) {

var power = i\*7;

var limit = Math.pow(2, power);

var prev\_limit = Math.pow(2, (i-1)\*7);

if (a < limit) {

var count = a-prev\_limit+1;

sum += count\*i;

break;

} else {

var count = limit-prev\_limit;

sum += count\*i;

}

}

if (a > 0) {

x = sum / a;

} else {

x = 0;

}

Результат: avg(sizeof(V\_id))=1.15333333333333;

------------------------------------

Шаг № 8

Отношение: x = 2+2\*a (Размер сообщения из двух одинаковых полей есть сумма размеров этих одинаковых полей плюс заголовки этих полей.)

Правило: |E\_info| = 2+2\*avg(sizeof(V\_id)) (Информационный объём сообщения об одной связи равен удвоенному среднему размеру идентификатора узла плюс заголовки полей идентификаторов.)

Входные параметры:

avg(sizeof(V\_id))=1.15333333333333;

Формула:

x = 2+2\*a

Результат: |E\_info|=4.30666666666667;

------------------------------------

Шаг № 9

Отношение: x = Math.ceil(Math.log(a+1)/Math.log(2)/7) (Расчёт длины числа в формате чисел варьируемой длины.)

Правило: sizeof(|E\_info|) = Math.ceil(Math.log(|E\_info|+1)/Math.log(2)/7) (Информационный объём размера сообщения об одной связи в формате варьируемых чисел.)

Входные параметры:

|E\_info|=4.30666666666667;

Формула:

var a, x;

if (a != 0) {

x = Math.ceil(Math.log(a+1)/Math.log(2)/7);

} else {

x = 0;

}

Результат: sizeof(|E\_info|)=1;

------------------------------------

Шаг № 10

Отношение: x = 1+a+b (Значение некоторых переменных является суммой других переменных плюс единица.)

Правило: |E\_msg| = 1+sizeof(|E\_info|)+|E\_info| (Информационный объём сообщения об одной связи с заголовком равен сумме размера сообщения об одной связи, длины размера и единицы.)

Входные параметры:

sizeof(|E\_info|)=1;

|E\_info|=4.30666666666667;

Формула:

x = 1+a+b

Результат: |E\_msg|=6.30666666666667;

------------------------------------

Шаг № 11

Отношение: x = a\*b\*(b-1)/2 (Формула вычисления числа рёбер в графе по его плотности.)

Правило: |E| = D\*|V|\*(|V|-1)/2 (Формула вычисления числа рёбер в графе по его плотности и числу вершин в этом графе.)

Входные параметры:

D=0.1;

|V|=150;

Формула:

x = a\*b\*(b-1)/2

Результат: |E|=1117.5;

------------------------------------

Шаг № 12

Отношение: x = a\*b (Значение переменной равно перемножению двух других переменных.)

Правило: |E\_list| = |E|\*|E\_msg| (Информационный объём массива сообщений о связях равен перемножению количества связей на информационный объём сообщения об одной связи с заголовком.)

Входные параметры:

|E|=1117.5;

|E\_msg|=6.30666666666667;

Формула:

x = a\*b

Результат: |E\_list|=7047.7;

------------------------------------

Шаг № 13

Отношение: x = a+b (Значение некоторых переменных является суммой других переменных.)

Правило: p\_F = p\_FP+p\_FN (Вероятность ложного срабатывания нейронной сети равна сумме вероятности ложноположительного срабатывания нейронной сети и вероятности ложноотрицательного срабатывания нейронной сети.)

Входные параметры:

p\_FP=0;

p\_FN=0;

Формула:

x = a+b

Результат: p\_F=0;

------------------------------------

Шаг № 14

Отношение: x = a\*b (Значение переменной равно перемножению двух других переменных.)

Правило: |E\_exlist| = p\_F\*|E\_list| (Информационный объём массива сообщений о связях, неправильно определяемых нейронной сетью, равен произведению вероятности ложного срабатывания нейронной сети на информационный объём массива сообщений о всех связях.)

Входные параметры:

p\_F=0;

|E\_list|=7047.7;

Формула:

x = a\*b

Результат: |E\_exlist|=0;

------------------------------------

Шаг № 15

Отношение: x = Math.ceil(Math.log(a+1)/Math.log(2)/7) (Расчёт длины числа в формате чисел варьируемой длины.)

Правило: sizeof(|network|) = Math.ceil(Math.log(|network|+1)/Math.log(2)/7) (Информационный объём размера нейронной сети, предсказывающей наличие связи между узлами по паре связанных с этими узлами векторов, в формате чисел варьируемой длины.)

Входные параметры:

|network|=2404;

Формула:

var a, x;

if (a != 0) {

x = Math.ceil(Math.log(a+1)/Math.log(2)/7);

} else {

x = 0;

}

Результат: sizeof(|network|)=2;

------------------------------------

Шаг № 16

Отношение: x = 1+a+b (Значение некоторых переменных является суммой других переменных плюс единица.)

Правило: |E\_network| = 1+sizeof(|network|)+|network| (Информационный объём поля нейронной сети, предсказывающей наличие связи между узлами по паре связанных с этими узлами векторов, равен сумме размера нейронной сети, длины размера и единицы.)

Входные параметры:

sizeof(|network|)=2;

|network|=2404;

Формула:

x = 1+a+b

Результат: |E\_network|=2407;

------------------------------------

Шаг № 17

Отношение: x = Math.ceil(Math.log(a+1)/Math.log(2)/7) (Расчёт длины числа в формате чисел варьируемой длины.)

Правило: sizeof(V\_emb\_size) = Math.ceil(Math.log(V\_emb\_size+1)/Math.log(2)/7) (Информационный объем размера векторного представления связей узла в байтах не должен быть отрицательным в формате чисел варьируемой длины.)

Входные параметры:

V\_emb\_size=1;

Формула:

var a, x;

if (a != 0) {

x = Math.ceil(Math.log(a+1)/Math.log(2)/7);

} else {

x = 0;

}

Результат: sizeof(V\_emb\_size)=1;

------------------------------------

Шаг № 18

Отношение: x = 1+a+b (Значение некоторых переменных является суммой других переменных плюс единица.)

Правило: |V\_embedding| = 1+sizeof(V\_emb\_size)+V\_emb\_size (Информационный объём поля вектора, связанного с одной вершиной графа, равен сумме размера вектора, длины размера вектора и единицы.)

Входные параметры:

sizeof(V\_emb\_size)=1;

V\_emb\_size=1;

Формула:

x = 1+a+b

Результат: |V\_embedding|=3;

------------------------------------

Шаг № 19

Отношение: x = a+b (Значение некоторых переменных является суммой других переменных.)

Правило: |V\_info| = |V\_data|+|V\_embedding| (Информационный объём всех полей вершины графа статического размера с учётом векторного представления равен сумме информационного объёма поля дополнительной информации о вершине графа и информационного объёма поля вектора, связанного с одной вершиной графа.)

Входные параметры:

|V\_data|=10;

|V\_embedding|=3;

Формула:

x = a+b

Результат: |V\_info|=13;

------------------------------------

Шаг № 20

Отношение: x = sum of message sizes with id up to "a" and payload of size "b" (Данное отношение вычисляет общий информационный объём вложенных сообщений с увеличивающимся идентификатором вплоть до "a" и с данными размером "b".)

Правило: |nodes'| = sum of node message sizes with id up to |V| and payload of size |V\_info| (Информационный объём массива вершин графа с учётом векторного представления связей вычисляется как общий информационный объём вложенных сообщений узлов с увеличивающимся идентификатором вплоть до |V| и с данными размером |V\_info|.)

Входные параметры:

|V|=150;

|V\_info|=13;

Формула:

var x, a, b;

x = 0;

for (var i = 1; i <= 10; ++i) {

var power = i\*7;

var limit = Math.pow(2, power);

var prev\_limit = Math.pow(2, (i-1)\*7);

var payload\_size = 1+i+b;

var sizeof\_payload\_size = Math.ceil(Math.log(payload\_size+1)/Math.log(2)/7);

if (a < limit) {

var count = a-prev\_limit+1;

x += count\*(1+sizeof\_payload\_size+payload\_size);

break;

} else {

var count = limit-prev\_limit;

x += count\*(1+sizeof\_payload\_size+payload\_size);

}

}

Результат: |nodes'|=2573;

------------------------------------

Шаг № 21

Отношение: x = a+b+c (Значение некоторых переменных является суммой трёх других переменных.)

Правило: |G\_vec| = |nodes'|+|E\_network|+|E\_exlist| (Информационный объём графа с векторным представлением связей равен сумме информационного объёма массива вершин графа с учётом векторного представления связей, информационного объёма поля нейронной сети, предсказывающей наличие связи между узлами по паре связанных с этими узлами векторов, и информационного объёма массива сообщений о связях, неправильно определяемых нейронной сетью.)

Входные параметры:

|nodes'|=2573;

|E\_network|=2407;

|E\_exlist|=0;

Формула:

x = a+b+c

Результат: |G\_vec|=4980;

------------------------------------

Шаг № 22

Отношение: x = a+b (Значение некоторых переменных является суммой других переменных.)

Правило: |G\_list| = |nodes|+|E\_list| (Информационный объём графа с представлением связей в виде списка связей равен сумме информационного объёма массива вершин графа и информационного объёма поля списка связей.)

Входные параметры:

|nodes|=2123;

|E\_list|=7047.7;

Формула:

x = a+b

Результат: |G\_list|=9170.7;

------------------------------------

1. Графическая часть выпускной квалификационной работы

В графическую часть выпускной квалификационной работы входят:

* существующие подходы по сжатию связей с помощью векторного представления;
* общая структура метода сжатия графов на основе векторного представления;
* структура модели для обучения;
* модифицированный алгоритм построения матрицы смежности;
* алгоритм подготовки входных данных для обучения;
* алгоритм оценки точности итоговой модели;
* информационный объём графа в различных представлениях;
* анализ эффективности разработанного алгоритма;
* анализ эффективности разработанного алгоритма с учётом сжатия алгоритмом общего назначения;
* апробация алгоритма на других графах.

