



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, искусственный интеллект и системы управления
КАФЕДРА Системы обработки информации и управления

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:

Исследование векторного представления метаграфов

Студент ИУ5-41М
(Группа)

(Подпись, дата) А.А. Фадеев
(И.О.Фамилия)

Руководитель ВКР

(Подпись, дата) Ю.Е. Гапанюк
(И.О.Фамилия)

Нормоконтролер

(Подпись, дата) Ю.Н. Кротов
(И.О.Фамилия)

РЕФЕРАТ

Данная выпускная квалификационная работа магистра посвящена исследованию векторного представления метаграфа, анализу возможностей и определению подходов для решения задачи вложения метаграфа, используя существующие алгоритмы эмбединга плоских графов.

Актуальность работы подтверждается следующими моментами: отсутствие методик для проведения процедуры вложения для метаграфа и, как следствие, отсутствие примеров практического применения метаграфовой модели знаний, как входных данных для моделей машинного обучения. Сама метаграфовая модель знаний является комплексной и универсальной, позволяет описывать связи любого рода, однако без эмбединга её невозможно использовать в качестве входных данных для моделей машинного обучения.

Создание алгоритма преобразования модели метаграфа к модели плоского графа занимает самое важное место в этой работе, поскольку это позволяет использовать существующие алгоритмы эмбединга плоских графов и получать векторное представление. Особое внимание уделяется полному сохранению всей исходной информации метаграфа и её корректности после осуществления операции эмбединга и получения векторного представления, а также корректности интерпретации иерархии. Также в работе исследуется возможность получения векторных представлений из метаграфа.

Недостатками данной работы являются небольшое количество исследований, применённых алгоритмов эмбединга графов и небольшой размер метаграфа, используемого в качестве демонстрационного примера.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 Анализ предметной области.....	8
1.1. Средства обработки графовых моделей данных.....	8
1.2. Графовые модели данных.....	9
1.3. Метаграфовая модель данных.....	10
1.4. Операции над метаграфом.....	16
1.5. Представление метаграфа.....	18
1.6. Предикатное представление метаграфа.....	19
2 Разработка метода эмбединга метаграфов.....	26
2.1. Особенности эмбединга метаграфов.....	26
2.2. Преобразование метаграфа в многодольный граф.....	26
2.3. Алгоритм преобразования метаграфа в трёхдольный граф.....	31
2.4. Подходы к эмбедингу графов.....	34
2.5. Входные данные эмбединга графов.....	38
2.6. Особенности многодольных (трёхдольных) графов.....	39
2.7. Выходные данные эмбединга графов.....	41
2.8. Алгоритмы эмбединга.....	43
2.8.1. Встраивание с сохранением близости высокого порядка (HOPE – High-Order Proximity preserved Embedding).....	45
2.8.2. Собственные карты Лапласа (Laplacian Eigenmaps).....	46
2.8.3. Node2Vec.....	48
3 Экспериментальная часть.....	50
3.1. Используемый метаграф.....	50
3.2. Преобразование метаграфа в трёхдольный плоский граф.....	51
3.3. Эксперименты.....	54
3.3.1. Эксперимент с алгоритмом Встраивание с сохранением близости высокого порядка (High-Order Proximity preserved Embedding, HOPE).....	55

3.3.2. Эксперимент с алгоритмом Собственные карты Лапласа (Laplacian Eigenmaps, LE).....	59
3.3.3. Эксперимент с алгоритмом Node2Vec	69
3.4. Обобщение результатов экспериментов	72
ЗАКЛЮЧЕНИЕ	74
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	76
ПРИЛОЖЕНИЕ А	82
ПРИЛОЖЕНИЕ Б.....	92

ВВЕДЕНИЕ

Графы широко и разнообразно представлены в реальной жизни: социальный граф, диффузионный граф в социальных сетях, граф цитирования в областях исследований, граф транспортно-пересадочных узлов, граф интересов пользователей в области электронной торговли, граф знаний и т. д. Анализ этих графов дает представление о том, как использовать информацию, скрытую в них, как именно делать это правильно, как оптимизировать эти графы с разных точек зрения. Эти вопросы с момента появления понятия «граф» привлекали много внимания, а с началом информационной эпохи графы буквально окружили нас. В настоящее время многие из сетей и графов, которыми мы окружены, не оптимальны. В течение своей истории они развивались поэтапно, и довольно часто развитие производилось не с прицелом на будущие изменения, а для получения краткосрочного результата. Появление вычислительных возможностей анализировать такие сложные сетевые, иерархические структуры подстегнуло интерес к разного рода исследованиям: уже несколько десятилетий развивается графовая аналитика. Графовая аналитика может принести пользу многим приложениям, таким как классификация узлов [1], кластеризация узлов [2], поиск и рекомендация узлов [3], прогнозирование связей [4] и так далее. Например, путем анализа графа, построенного на основе взаимодействия с пользователями в социальной сети, мы можем классифицировать пользователей, обнаруживать сообщества, рекомендовать друзей и прогнозировать взаимодействие между двумя пользователями. Также можно проанализировать граф транспортно-пересадочных узлов, обнаружить несовершенства в маршрутах различных видов транспорта, выяснить, какие ТПУ нуждаются в расширении и модернизации и так далее.

Для решения задачи вложения графа входные данные должны представлять собой графы, чтобы над ними можно было произвести операцию эмбединга. Стоит обратить внимание на факт того, что любой объект является

конструкцией из каких-то составляющих. Следовательно, объекты можно описывать с помощью графовых структур. Сложность объекта влияет на сложность вида графа, которым его можно представить. Следовательно, метаграф должен быть использован тогда, когда сам объект является слишком сложным для осуществления корректного описания с помощью различных видов плоских графов. Очень важно соблюдать прямую связь описания с данными решаемой задачи, чтобы в графе или метаграфе не было лишней информации, но при этом не было потеряно никаких фактов, влияющих на решение задачи. Наиболее удобен такой подход для решения задач с разнообразными сетями, где исходные данные изначально описаны различными графовыми структурами: социальные сети, веб-графы, биологические сети, сеть транспортных путей и так далее. Эти сети являются очень большими и сложными для обработки, поэтому для них очень актуальны вопросы обобщения графовых структур и векторного представления элементов графа (желательно небольшой размерности). Конечно, самое важное, чтобы векторное представление позволяло использовать себя в качестве входных данных для алгоритмов машинного обучения. Желательно, чтобы формат таких данных не ограничивал набор возможных моделей машинного обучения.

Преимущества описанного выше подхода обосновывают его выбор в данной работе: размерность входных данных сокращается без потери информации; уровень абстракции исследуемого объекта повышается; за счёт этого промежуточные данные и конечное векторное представление легко интерпретируются, а также появляется возможность смены задачи или используемых при исследовании методов.

Метаграфовая модель данных предлагает существенный уровень абстракции описания объектов, который позволяет реализовать описанный выше подход. Абстракция актуальна в сложных сетях, которые, как оговаривалось выше, в настоящее время обычно сложно структурированы и которые из-за этого сложно анализировать. Представленная в работе методика вложения (эмбединга) сложной метаграфовой модели, позволяет представлять

иерархичную и максимально общую формализованную модель метаграфа в виде набора низкоразмерных векторов. Такие вектора являются структурно-сохраненным сетевым представлением, «удобным» и частым типом входных данных в большинстве алгоритмов машинного и глубокого обучения.

1 Анализ предметной области

1.1. Средства обработки графовых моделей данных

Последние исследования и разработки известных производителей электронных микросхем, доказали, что будущие микропроцессоры и большие высокопроизводительные вычислительные машины будут гибридными [56]. В их основу будут положены компоненты 2 основных типов:

1. мультиядерные и многоядерные центральные процессоры: так как уже достигнут физический предел размеров транзисторов и технологического процесса, количество ядер процессоров будет и дальше увеличиваться для повышения мощности выпускаемых чипов;

2. специализированное оборудование и массивно-параллельные ускорители: графические процессоры в последние годы превзошли центральные процессоры в плане производительности вычислений над числами с плавающей точкой. Программирование графических процессоров стало таким же простым, как и программирование центральных процессоров.

Раньше любые задачи решали на центральных процессорах. Теперь же всё большее число задач можно решать на графическом процессоре, а некоторые задачи решаются на них эффективнее, чем на центральных процессорах. Лучшая эффективность объясняется повышенной пропускной способностью современных графических процессоров по сравнению с центральными. Благодаря повышенной пропускной способности сокращается время выполнения задач. Как правило, к задачам, решение которых возлагают на графические процессоры, относятся задачи, которые позволяют вести вычисления параллельно.

В данной работе ведётся исследование путей создания алгоритма, который бы формировал векторные представления метаграфов для обработки процессором, которая могла бы производиться независимо от его типа. Операции линейной алгебры над матрицами уже успешно выполняются на

различных процессорах. В данной работе предполагается: так как разнообразные операции линейной алгебры над матрицами уже успешно выполняются, то было бы разумным представить метаграфовую модель в виде набора тензоров или многомерных векторов.

Для плоских графов уже существует множество алгоритмов вложения (эмбединга) в векторное пространство, однако, не существует алгоритма для сложной иерархической метаграфовой модели знаний. Поэтому центральной частью работы является поиск и построение алгоритма эмбединга метаграфа в векторное представление, чтобы в дальнейшем эти результаты могли использоваться для создания систем по обработке метаграфов.

1.2. Графовые модели данных

Данный раздел посвящён обзору различных нестандартных, сложных графовых моделей, описывающих сложные сети.

С точки зрения обработки данных графы можно поделить на три категории: простые небольшие графы, плоские графы большой размерности и графы, обладающие комплексным и разнообразным описанием узлов. Обработка простых графов в настоящее время без проблем ведётся с помощью реляционных или графовых СУБД.

Со вторым видом графов всё не так просто из-за их огромного размера. Такие графы включают в себя миллионы вершин и рёбер, которые могут быть направленными и ненаправленными. Вершины в таких графах (имеющих специальное название «мультиграф») могут соединяться двумя и более рёбрами, и чаще всего в литературе именно такие сложные графы и называют сложной сетью, поскольку в ней максимально широко используются возможности классической графовой модели знаний.

Графы третьего вида – графы, в которых используется сложное описание вершин, рёбер и/или их расположения. Как правило, в таких моделях элементы располагаются не на плоскости, а в n -мерном пространстве, где $n \geq 3$. Часто в

таких моделях вводятся уникальные понятия, описывающие элементы модели. Такие свойства этого вида графов могут быть наиболее полезны при описании сложных моделей данных и знаний. Из подобных моделей на сегодняшний день лучше всего изучены три: гиперграф, гиперсеть и метаграф. В данной квалификационной работе исследуется только метаграфовая модель данных.

В настоящее время во многих прикладных задачах необходимо рассматривать графы очень большой размерности. Обработка такого графа (или мультиграфа) является чрезвычайно затратной с точки зрения использования памяти компьютера и вычислительных ресурсов в целом. В случае обработки метаграфов задача ещё больше усложняется за счет комплексного описания моделей.

1.3. Метаграфовая модель данных

В данном разделе представлено общее описание метаграфовой модели данных.

А. Базу и Р. Блэннинг в 2007 году издали монографию [5], в которой обобщили концепцию метаграфа на основе своих более ранних статей. Данная монография считается научным сообществом основополагающей в теории метаграфов.

Метаграфовая модель данных, описанная в работе [5], была адаптирована в рамках научных работ на кафедре «Системы обработки информации и управления» МГТУ им. Н.Э. Баумана и описана в статьях [6-9]. Применять данную модель следует для описания сложных сетей [10], семантики и прагматики различных информационных систем [11]. Также в работе [6] предлагается использовать метаграфовую модель данных как средство для описания гибридных интеллектуальных информационных систем. Ниже представлено формализованное описание метаграфовой модели данных в чётком соответствии с [6-9].

Метаграф (1) определяется в виде сущности:

$$MG = \langle V, MV, E, ME \rangle, \quad (1)$$

где MG – метаграф; V – множество вершин метаграфа; MV – множество метавершин метаграфа; E – множество рёбер метаграфа; ME – множество метарёбер метаграфа.

Вершина (2) метаграфа характеризуется множеством атрибутов:

$$v_i = \{atr_k\}, v_i \in V, \quad (2)$$

где v_i – вершина метаграфа; atr_k – атрибут.

Ребро метаграфа характеризуется множеством атрибутов, исходной и конечной вершиной и признаком направленности:

$$e_i = \langle v_S, v_E, eo, \{atr_k\} \rangle, e_i \in E, eo = true|false, \quad (3)$$

где e_i – ребро метаграфа; v_S – исходная вершина (метавершина) ребра; v_E – конечная вершина (метавершина) ребра; eo – признак направленности ребра ($eo = true$ – направленное ребро, $eo = false$ – ненаправленное ребро); atr_k – атрибут.

Метавершина (4) характеризуется следующим образом:

$$mv_i = \langle \{atr_k\}, \{ev_j\} \rangle, mv_i \in MV, ev_j \in (V \cup E^{eo=false} \cup MV \cup ME^{eo=false}), \quad (4)$$

где mv_i – метавершина метаграфа; atr_k – атрибут, ev_j – элемент, принадлежащий объединению множеств вершин, метавершин, рёбер и метарёбер метаграфа.

Таким образом, метавершина в дополнение к свойствам вершины включает вложенный фрагмент метаграфа. В рамках такой модели рёбра и метарёбра метавершин должны быть ненаправленными ($eo = false$).

Метаребро (5) характеризуется следующим образом:

$$\begin{aligned} me_i = \langle v_S, v_E, eo, \{atr_k\}, \{ev_j\} \rangle, me_i \in ME, eo = true|false, \\ ev_j \in (V \cup E^{eo=true} \cup MV \cup ME^{eo=true}), \end{aligned} \quad (5)$$

где me_i – ребро метаграфа; v_S – исходная вершина (метавершина) ребра; v_E – конечная вершина (метавершина) ребра; eo – признак направленности метаребра ($eo = true$ – направленное метаребро, $eo = false$ – ненаправленное метаребро); atr_k – атрибут, ev_j – элемент, принадлежащий объединению множеств вершин, метавершин, рёбер и метарёбер метаграфа.

Таким образом, метаребро в дополнение к свойствам ребра включает вложенный фрагмент метаграфа. При этом рёбра и метарёбра этого фрагмента могут быть только направленными, $eo = true$. Это свойство позволяет легче описывать процессы. Если его использование не нужно, то лучше вместо метаребра использовать метавершину, комплекс метавершин или разделить метарёбро на более мелкие составляющие, поскольку рёбра (или метарёбра) со свойством $eo = false$ прерывает цепочку процесса, отображаемого метаребром.

Стоит заметить, что определения метавершины и метарёбра означают, что в их непосредственный состав не могут входить одни и те же рёбра и метарёбра, поскольку в метавершину могут включаться только ненаправленные рёбра и метарёбра, а в метаребро – только направленные. Такое требование оригинальной модели показывает, что метавершина предназначена для описания данных и знаний, а метаребро – для описания процесса. Таким образом, метаграфовая модель позволяет в рамках единой модели описывать данные, знания и процессы.

Описание процесса не всегда является необходимостью, поэтому метарёбра полагается считать необязательным элементом метаграфовой модели знаний. В данной квалификационной работе рассматривается создание алгоритма эмбединга метаграфа, включающего вершины, рёбра и метавершины. В рамках научной работы на Кафедре подразумевается

усовершенствовать определение метаребра для более корректного и полного описания процессов, и предлагаемый в работе алгоритм планируется, по возможности, усовершенствовать путём добавления способа эмбединга метавершин.

Для осуществления операций изменения метаграфа в модели вводится понятие фрагмента метаграфа:

$$MG_i = \{ev_j\}, ev_j \in (V \cup E \cup MV \cup ME), \quad (6)$$

где MG_i – фрагмент метаграфа; ev_j – элемент, принадлежащий объединению множеств вершин, метавершин, рёбер и метарёбер метаграфа.

Таким образом, фрагмент метаграфа в общем виде может содержать произвольные вершины, метавершины, рёбра и метаребра без ограничений. Ограничения вводятся лишь на фрагменты метаграфа, входящие в метавершину и метаребро.

Пример описания небольшого метаграфа без метарёбер показан ниже на рисунке 1.

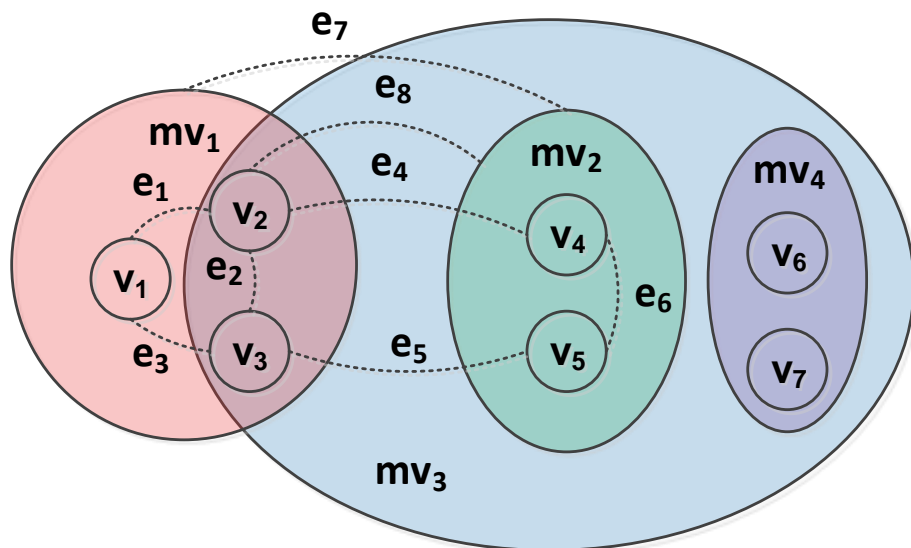


Рисунок 1 – Пример метаграфа

Данный метаграф содержит вершины, метавершины и ребра. На рис. 1 показаны четыре метавершины: mv_1 , mv_2 , mv_3 и mv_4 . Метавершина mv_1

содержит вершины v_1, v_2, v_3 и связывающие их ребра e_1, e_2, e_3 . Метавершина mv_2 содержит вершины v_4, v_5 и связывающее их ребро e_6 . Ребра e_4 , соединяющее вершину v_2 с вершиной v_4 , и e_5 , соединяющее вершины v_3 и v_5 , являются примерами ребер, соединяющих вершины, включенные в различные метавершины. В данном случае это метавершины mv_1 и mv_2 . Ребро e_7 соединяет метавершины mv_1 и mv_2 между собой. На рисунке также есть ребро e_7 , которое соединяет вершину v_2 и метавершину mv_2 . Метавершина mv_3 включает метавершины mv_2 и mv_4 , вершины v_2, v_3 и ребро e_2 из метавершины mv_1 а также ребра e_4, e_5 и e_8 . Метавершина mv_4 включает не соединенные ребрами вершины v_6 и v_7 .

Пример описания метаребра, наложенного на метаграф с рисунка 1, показан ниже на рисунке 2.

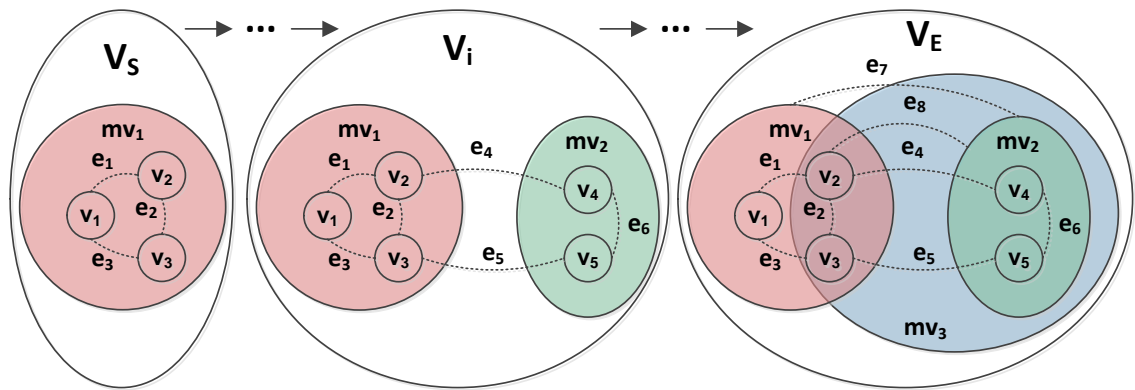


Рисунок 2 – Пример метаребра

Метаребро содержит метавершины $v_S, \dots, v_i, \dots, v_E$ и связывающие их рёбра. Исходная метавершина содержит фрагмент метаграфа. В процессе преобразования исходной метавершины v_S в конечную метавершину v_E происходит дополнение содержимого метавершины, добавляются новые вершины, связи, вложенные метавершины. При этом стоит обратить внимание, что направление обхода вершин метаребра строго задано. Такое метаребро удобно рассматривать как средство сохранения информации об изменениях состояний объектов во времени. Таким образом метаграфовая модель знаний за

счёт этой вспомогательной конструкции позволяет сохранять информацию о различных версиях модели.

Метавершина, имея набор атрибутов, обладает собственным набором свойств. Это позволяет метаграфовой модели данных обладать свойством эмерджентности, которое является ключевым для сложных сетей и которое позволяет описывать многоуровневые модели, их сложную иерархию и вложенность. Это свойство заключается в возможности обладания уникальными свойствами какой-либо совокупностью объектов модели. Обладая своими уникальными атрибутами, метавершина может выступать как единое целое в составе целого.

Метаграфовая модель является далеко не единственной для описания сложных сетей. В работах [12-13] подробно описаны модели гиперграфа и гиперсети. В работе [7] приводятся описания этих моделей и детальное сравнение с метаграфовой моделью, которое выявило следующие различия между моделями: По результатам сравнения моделей можно сделать следующие выводы:

- Модель гиперсети и гиперграфа фактически представляет собой попытку описания «сверху-вниз» (по уровням), а модель метаграфа попытку описания «снизу-вверх» (путем «выращивания» метавершин из более простых элементов);
- Модели гиперсети и гиперграфа, в отличие от метаграфовой модели, позволяют связывать только соседние слои;
- Модель метаграфа является более простой в описании, так как состоит из однородных элементов (метавершин и связей, как элементов метавершин);
- Модель метаграфа является более гибкой, так как не требует регулярности уровней. Произвольный подграф может быть превращен в метавершину.

Метаграфовая модель данных за счёт своей гибкости и простоты позволяет использовать её в более широком наборе задач, чем модели гиперграфа и гиперсети, поэтому в данной работе исследуется именно эта модель.

Однако, используется не оригинальная метаграфовая модель, описанная выше в выражениях (1) – (6), а немного изменённая. Метаребро в оригинальной модели имеет ограничение, связанное с тем, что рёбра и метаребра, входящие в её непосредственный состав, не могут быть направленными. В данной работе предлагается избавиться от этого условия. Определение фрагмента метаграфа уже удовлетворяет ему. Следовательно, переопределяется понятие метавершины, которое будет использоваться в рамках данной работы.

$$mv_i = \langle \{atr_k\}, MG_j \rangle, mv_i \in MV, MG_j = \{ev_l\}, ev_l \in (V \cup E \cup MV \cup ME),$$

где mv_i – метавершина метаграфа; atr_k – атрибут, MG_j – фрагмент метаграфа, ev_l – элемент, принадлежащий объединению множеств вершин, метавершин, рёбер и метаребер метаграфа.

Изменение оригинальной модели сделано не просто так. Целью оригинального ограничения является явное разделение ролей метавершины и метаребра. Однако, учитывая, что метаграф – это в первую очередь модель данных, нет необходимости ограничивать возможности описания ключевой сущностью модели. Во многих областях знаний, например, в области логистических или транспортных сетей, рёбра обязаны быть направленными, но они всё ещё будут описывать лишь данные и объекты. Введённое изменение касается лишь исследовательской части данной работы.

1.4. Операции над метаграфом

Как и говорилось выше, метаграф, как модель данных, не может существовать без возможности редактирования данных.

В работе [11] были рассмотрены основные операции над метаграфами на основе «информационных элементов метаграфа» (ИЭМ). В дальнейшем в работе [14] в качестве модели представления метаграфа вместо ИЭМ было предложено использование предикатного описания.

В качестве активного элемента метаграфовой модели используется метаграфовые агенты, рассмотренные в работе [14]. Определим метаграфовой агент следующим образом:

$$ag^M = \langle MG_D, R, AG^{ST} \rangle, R = \{r_j\};$$

где ag^M – метаграфовый агент, MG_D – фрагмент метаграфа, на основе которого выполняются правила агента, т.е. рабочий метаграф. R – набор правил (множество правил r_j); AG^{ST} – стартовое условие выполнения агента (фрагмент метаграфа, который используется для стартовой проверки правил, или стартовое правило).

Структура правила метаграфового агента:

$$r_i: MG_j \rightarrow OP^{MG}.$$

где r_i – правило; MG_j – фрагмент метаграфа, на основе которого выполняется правило; OP^{MG} – множество операций, выполняемых над метаграфом.

Существует два вида метаграфовых агентов: замкнутые и разомкнутые.

Разомкнутые правила в своей правой части не меняют фрагмент метаграфа, относящийся к левой части правила. Такие правила позволяют разделить входной и выходной фрагменты метаграфа и являются аналогом шаблона, который порождает выходной метаграф на основе входного.

Замкнутые правила в своей правой части меняют фрагмент метаграфа, относящийся к левой части правила. Изменение метаграфа в правой части правил заставляет срабатывать левые части других правил. Такое свойство опасно

зацикливанием метаграфового агента, если замкнутые правила были разработаны некорректно.

Таким образом, метаграфовый агент позволяет генерировать один метаграф на основе другого (с использованием разомкнутых правил) или модифицировать метаграф (с использованием замкнутых правил).

1.5. Представление метаграфа

При обработке графов большой размерности следует отказаться от традиционного представления графа непосредственно при его обработке, что показано на рисунке 3 ниже.

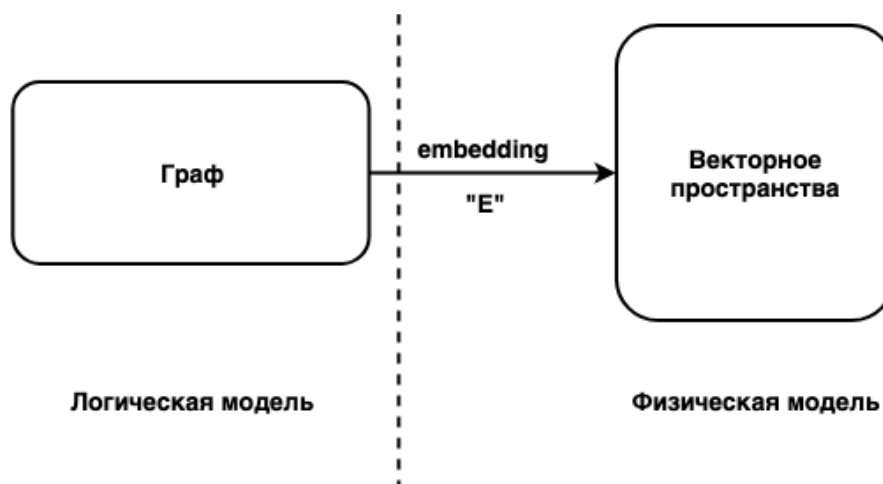


Рисунок 3 – Модель графа с точки зрения его обработки

Подходя к обработке метаграфа рекомендуется перевести его в вид, соответствующий оригинальной модели и постановке задачи. По аналогии с терминами реляционных баз данных, этот вид можно назвать «физической» моделью, а его оригинальный вид – «логической».

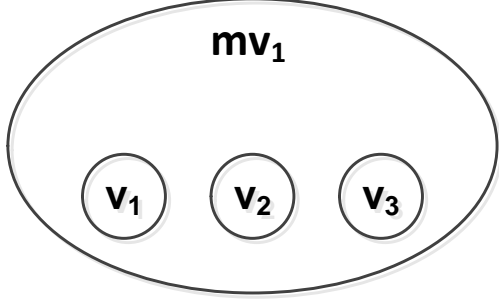
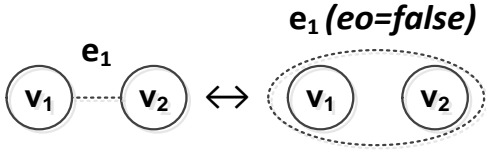
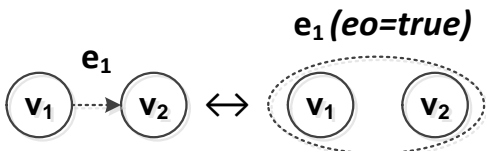
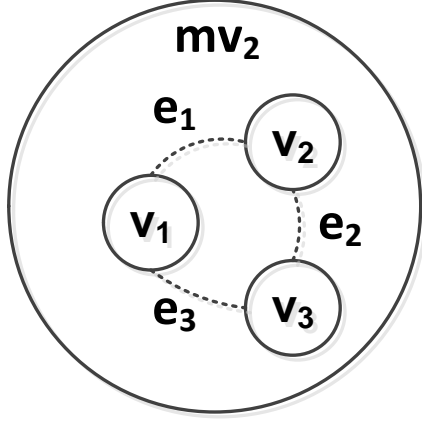
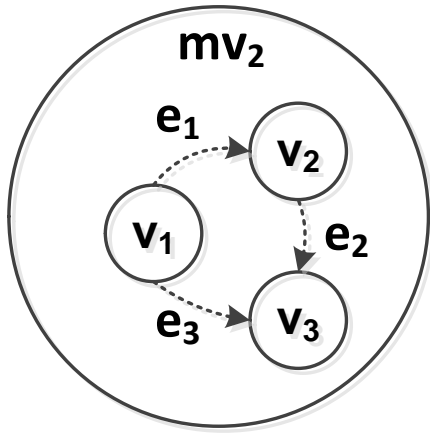
Логическая модель графа – это традиционная модель плоского графа или мультиграфа, включающая множества вершин и ребер. Тогда для метаграфа это определение звучит так: логическая модель метаграфа – это традиционная модель метаграфа, включающая множества вершин, рёбер, метавершин и метарёбер.


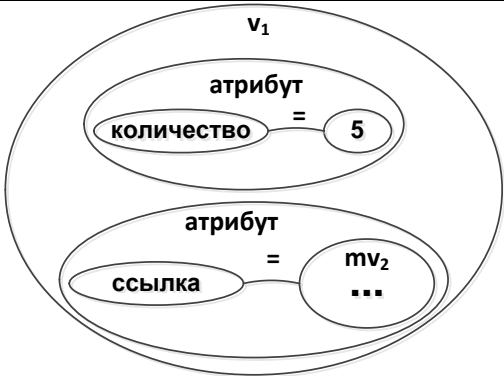
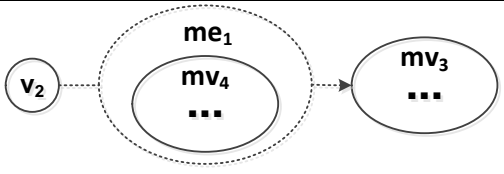
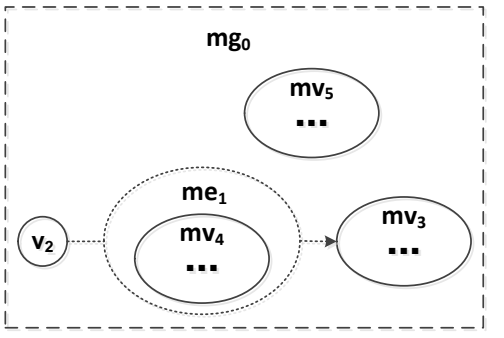
В качестве физической модели графа традиционно используются непрерывные векторные пространства. При этом нет никаких ограничений на использование других видов пространств. Операция преобразования графа в векторное пространство называется «векторным представлением» (связь “Е” на рисунке 3). В англоязычной литературе для обозначения такого преобразования традиционно используется термин «embedding», то есть «встраивание» или «вложение» графа в векторное пространство. В данной работе предлагается использовать именно этот подход для создания физической модели метаграфовой модели данных.

1.6. Предикатное представление метаграфа

В работе [14] в качестве модели представления метаграфа предлагается использовать предикатное описание. Язык Пролог на основе предикатов является классическим, он использует следующую форму предикатного описания: $predicate(atom_1, atom_2, \dots, atom_N)$. В работе [14] предлагается следующим образом расширить исходную форму предикатного описания языка Пролог: $predicate(atom, \dots, key = value, \dots, predicate(\dots), \dots)$. Помимо атомов, предлагаемых базовой формой предикатов, данная форма содержит пары ключ-значение и вложенные предикаты, что отлично подходит метаграфовой модели данных. Ниже, в таблице 1, представлены различные варианты предикатного описания различных элементов метаграфовой модели данных.

Таблица 1 – Соответствие метаграфовой модели предикатному описанию

Вариант	Фрагмент метаграфа	Предикатное описание
1		Metavertex(Name=mv ₁ , v ₁ , v ₂ , v ₃)
2		Edge(Name=e ₁ , v ₁ , v ₂ , eo=false)
3		1. Edge(Name=e ₁ , v ₁ , v ₂ , eo=true) 2. Edge(Name=e ₁ , v _S =v ₁ , v _E =v ₂ , eo=true)
4		Metavertex(Name=mv ₂ , v ₁ , v ₂ , v ₃ , Edge (Name=e ₁ , v ₁ , v ₂), Edge(Name=e ₂ , v ₂ , v ₃), Edge(Name=e ₃ , v ₁ , v ₃))
5		Metavertex(Name=mv ₂ , v ₁ , v ₂ , v ₃ , Edge(Name=e ₁ , v _S =v ₁ , v _E =v ₂ , eo=true), Edge(Name=e ₂ , v _S =v ₂ , v _E =v ₃ , eo=true), Edge(Name=e ₃ , v _S =v ₁ , v _E =v ₃ , eo=true))

Вариант	Фрагмент метаграфа	Предикатное описание
6		Attribute(количество, 5)
7		Vertex(Name=v1, Attribute(количество, 5), Attribute(ссылка, mv2))
8		Metaedge(Name=me1, vS=v2, vE=mv3, Metavertex(Name=mv4, ...), eo=true)
9		Metagraph(Name=mg0, Vertex(Name=v2, ...), Metavertex(Name=mv3, ...), Metavertex(Name=mv5, ...), Metaedge(Name=me1, vS=v2, vE=mv3, Metavertex(Name=mv4, ...), eo=true))

В таблице 1, в пункте 1 показан пример метавершины mv_1 , которая содержит 3 вложенных несвязных вершины v_1, v_2, v_3 . Именем предиката служит соответствующий элемент метаграфовой модели. В случае метавершины «Metavertex», в случае вершины «Vertex» и в случае ребра «Edge». Имя элементов задается именованным параметром «Name». Данный случай является простейшим, поскольку вложенные вершины не связаны друг с другом.

Пример ненаправленного ребра, который полностью соответствует формальному определению, показан в таблице 1, в пункте 2. В этом случае

метавершина помечается соответствующей аннотацией направленности, а предикат добавляется именованный параметр, соответствующий признаку направленности.

Пример направленного ребра показан в таблице 1, в пункте 3. В предлагаемой модели все параметры могут быть именованными. В пункте 3 именованные параметры соответствуют параметрам из формального определения ребер метаграфа.

Метавершина, содержащая вершины и ребра, может быть представлена с использованием предикатов более высоких порядков. В таблице 1, в пункте 4 имеется пример метавершины с ненаправленными ребрами, а в пункте 5 – пример метавершины с направленными ребрами. Метаграфовая модель данных предполагает описание вершин и ребер на одном уровне вложенности, что отражено и в предикатной модели. Фрагмент метаграфа может содержать произвольное количество вершин, метавершин и ребер.

Любой атрибут метаграфа может быть представлен как частный случай метавершины, куда вложены две связанные вершины. Одна вершина представляет из себя ключ атрибута, а вторая – его значение. В таблице 1, в пункте 6 представлен атрибут, содержащий целое число. Для Атрибута используется имя предиката «Attribute». Значения атрибутов могут иметь строковые, числовые и другие значения. Однако ключи могут быть только строковые. В пункте 7 показан пример вершины v_1 , содержащей атрибут из пункта 6, а также атрибут, ссылающийся на метавершину mv_2 .

Метаребро метаграфа также представляется с помощью предикатов высоких порядков. В пункте 8 таблицы 1 содержится пример метаребра me_1 , исходной вершиной которого является вершина v_2 , конечной метавершиной mv_3 , при этом метаребро содержит вложенную метавершину mv_4 . Для метаребра используется имя предиката «Metaedge».

Фрагмент метаграфа может содержать произвольное количество вершин, метавершин, ребер и метаребер метаграфа. Пункт 9 таблицы 1 содержит пример фрагмента метаграфа mg_0 , в который включена вершина v_2 , метавершины mv_3

и mv_5 , метаребро me_1 . Для фрагмента метаграфа используется имя предиката «Metagraph», для вершины – «Vertex».

Предикатное описание позволяет привести метаграфовую модель к текстовому виду. Также к преимуществам предикатной модели можно отнести простоту и взаимную независимость составных частей предикатного описания. Однако предикатное описание позволяет описать лишь данные и процессы, записанные в модели метаграфа, и не предоставляет инструментария для обработки этой информации.

Ниже, на рисунке 4, приведён пример предикатного описания метаграфа, приведённого выше на рисунке 1.

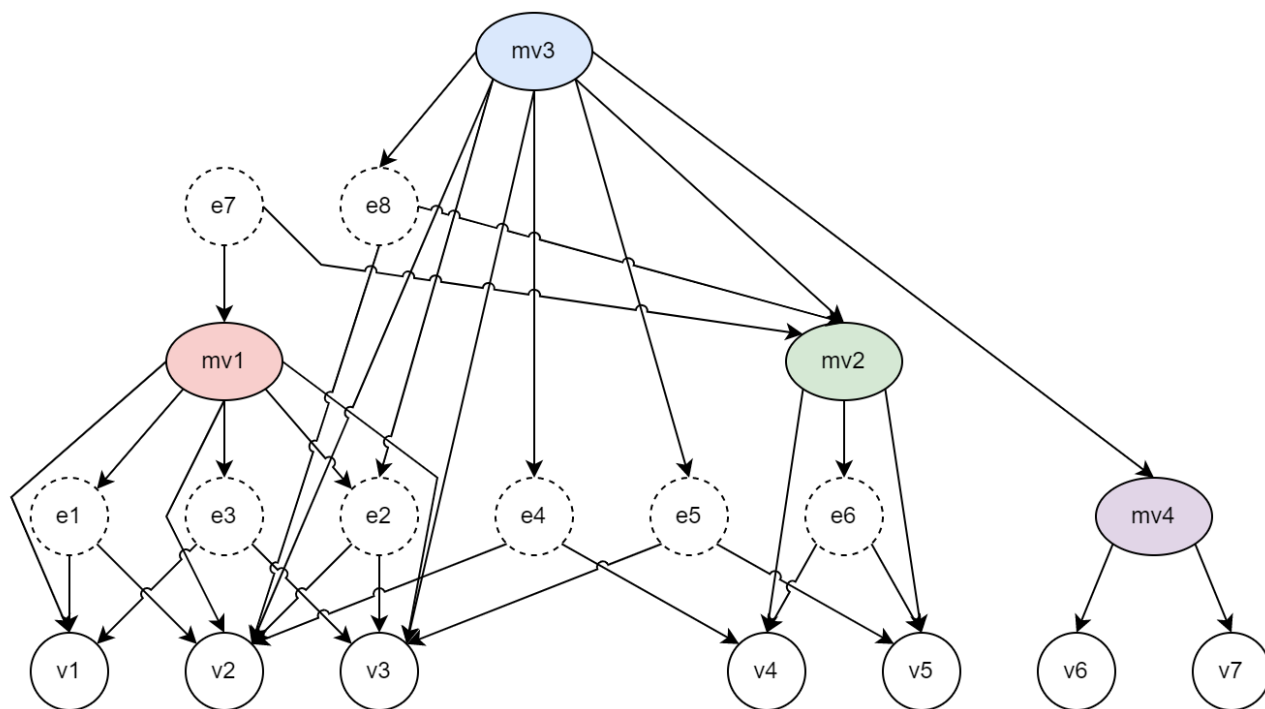


Рисунок 4 – Иерархическое предикатное представление метаграфа

На рисунке 4 предикаты показаны в виде вершин плоского графа. Вершины представляют собой белые круги, метавершины – цветные эллипсы, а рёбра – пунктирные круги. Иерархическая зависимость между предикатами сохраняется за счёт стрелок, которые всегда направлены к предикатам более низкого уровня. Поэтому изображение метаграфа с помощью предикатов позволяет подготовить метаграф к проведению операции эмбединга.

Граф, представленный на рисунке 4, является стратифицированным. Он разделён на слои, на уровни, хоть и довольно условным образом. Нижний уровень содержит предикаты-вершины. Второй уровень включает предикаты, соответствующие связям между элементами первого уровня. Более высокие уровни содержат метавершины и связи между ними. Данный пример условный потому, что метаграфовая модель данных позволяет, в отличие от моделей гиперграфа и гиперсети связывать не только соседние уровни. Это означает, что объект третьего и более высокого уровня может быть напрямую связан с объектом нижнего уровня. Между тем, количество уровней такого плоского графа косвенно указывает на сложность метаграфа и сложность его эмбединга.

Предикатное представление метавершин и ребер не имеет существенных различий. В данном примере получается так, что второй уровень графа содержит шесть ребер и метавершину mv_4 . Ребро обладает атрибутом направленности и может включать только два вложенных предиката, которые могут представлять из себя как метавершины, так и вершины. Метавершина же не имеет атрибута направленности и ограничения на количество вложенных предикатов. Таким образом ребро можно считать частным случаем метавершины.

Этот вывод может показаться парадоксальным с точки зрения классической теории графов, где вершины и рёбра считаются принципиально различными объектами. Однако это очень удобно и потенциально упрощает алгоритм эмбединга: можно рассматривать ребро как совокупность двух вложенных вершин.

Разница между двумя парами вершин, соединёнными и не соединёнными ребром, состоит в том, что первая пара вместе с ребром обладает свойством эмерджентности. Такая конструкция представляет собой отдельный объект, более высоко организованный чем две отдельные вершины.

Тогда метавершина обладает свойством эмерджентности в более высокой степени, ведь она включает как исходные вершины, так и соединяющие их ребра, и, возможно, метавершины и метаребра нижнего уровня.

Как видно из описания различных вариантов метаграфовых предикатов элементарные вершины представляют собой ноль-эмерджентность, то есть рассматриваются как атомарные элементы. Рёбра обеспечивают эмерджентность первого порядка, метавершины – второго, а метарёбра – третьего. Рассмотренный подход представляет собой основу метаграфового исчисления.

Все предлагаемые далее операторы предназначены для построения эмерджентных структур из вершин-предикатов.

2 Разработка метода эмбединга метаграфов

2.1. Особенности эмбединга метаграфов

Метаграфовая модель данных является молодой и, следовательно, неизученной и даже несовершенной. В настоящее время не существует методов прямого эмбединга метаграфов. Работа [13] предлагает метод изоморфного преобразования метаграфа в n -дольный граф, который в свою очередь возможно преобразовать в вектора с помощью алгоритмов, приведенных в работах [11-12].

В данной работе исследуется возможность эмбединга метаграфа через его плоский n -дольный граф.

2.2. Преобразование метаграфа в многодольный граф

Модель метаграфа не предоставляет информации о том, каким образом хранить его в какой-либо информационной системе. Эта модель знаний создавалась высокоуровневой для того, чтобы иметь возможность решать задачи как можно более широкого круга и хранить данные как можно более разноплановые. Метаграф молодая модель данных, она развивается и в будущем она наверняка потерпит изменения для улучшения представления данных о процессах.

Как и говорилось выше: метаграф является лишь логической моделью данных. Для использования метаграфов в реальной жизни необходимо иметь хотя бы одну физическую модель, к которой метаграф можно привести для хранения в базе данных. На данный момент такой общепринятой модели нет.

Ниже, на рисунке 5, изображён предикатный плоский граф из предыдущего раздела (см. рисунок 4) с небольшими изменениями, о введении которых говорится далее.

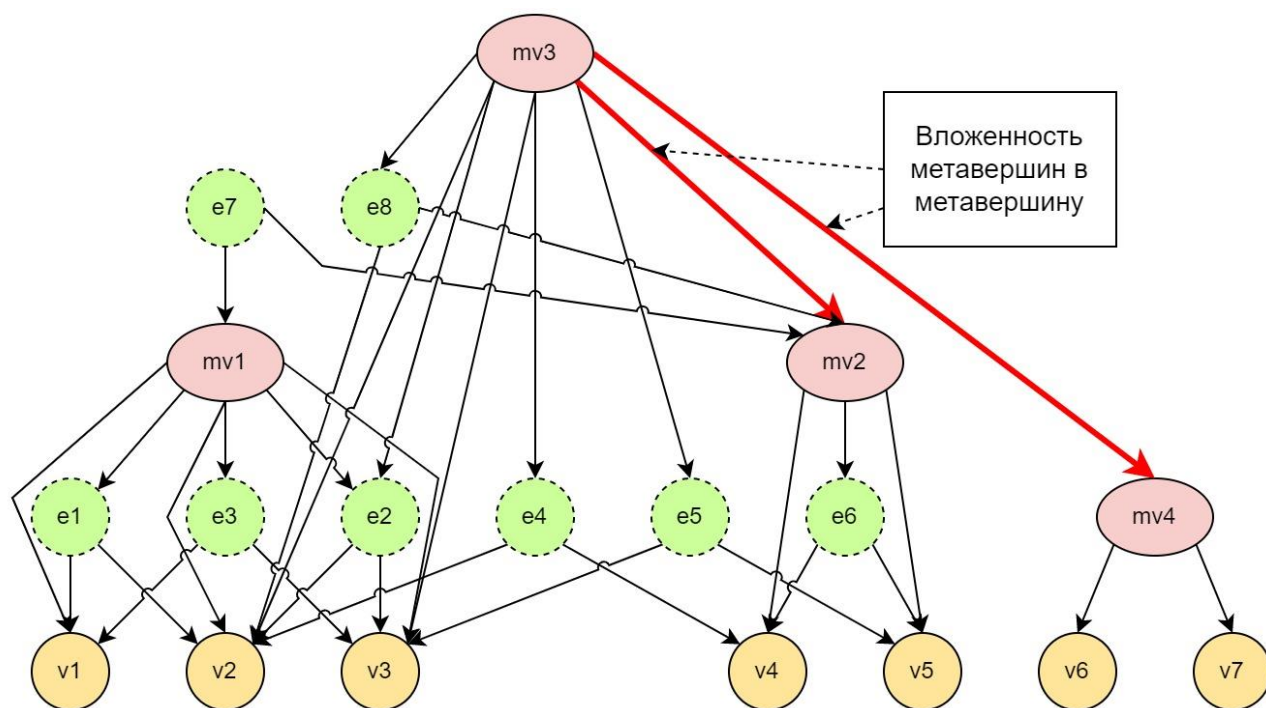


Рисунок 5 – Попытка разделения метаграфа на 3-дольный плоский граф

Как говорилось в предыдущем разделе, метавершину можно представить в виде композиции из обычной вершины и набора связей с другими вершинами, метавершинами и рёбрами.

Ни одна вершина не связана напрямую ни с одной из других вершин, поэтому объекты типа Вершина («Vertex») можно выделить в отдельный класс. Выделение классов производится для преобразования метаграфа в n -дольный граф.

Также очевидно, что ни одно ребро не связано с другими рёбрами, поэтому объекты типа Ребро («Edge») тоже выделяются в отдельный класс.

К сожалению, про метавершины из оригинальной метаграфовой модели такое сказать нельзя, однако, судя по всему, это не мешает использованию модели в начальном виде. На рисунке 5 отображены две связи между метавершинами mv_3 и mv_2 , а также mv_3 и mv_4 , которые говорят о включённости метавершин mv_2 и mv_4 в метавершину mv_3 . Это говорит о том, что выделение метавершин в отдельный класс при формировании n -дольного графа не в полной мере отвечает определению n -дольного графа.

Преобразование в n -дольный граф должно сохранять топологию оригинального метаграфа и позволять выделять классы узлов, кардинально различающихся по свойствам, и, к счастью, модель данных позволяет это сделать без введения незнакомых элементов. Для организации n -дольного графа, можно использовать фрагмент метаграфа как обёртку над элементами, входящими в состав метавершины. Этот элемент метаграфовой модели описан выше в разделах 1.3 и 1.6. Изменённый плоский граф с выделенными долями показан ниже, на рисунке 6.

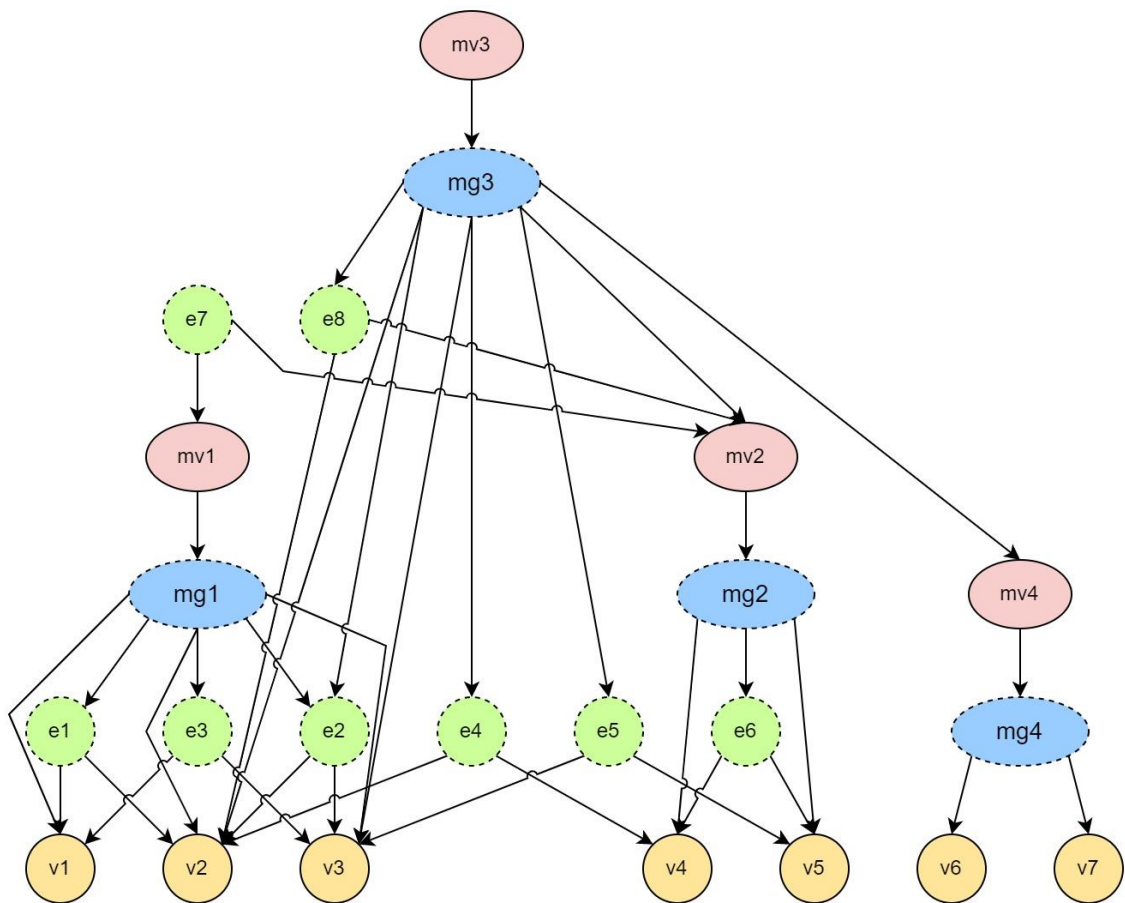


Рисунок 6 – Четырёхдольный граф

Подграфы, Фрагменты («Fragment») mg_{1-4} , не могут быть связаны друг с другом, поскольку каждая метавершина может иметь в своём составе только один фрагмент. С другой стороны, возможна ситуация, при которой один и тот же фрагмент будет находиться в составе двух и более метавершин. Такое может произойти не тогда, когда элементы одного фрагмента содержатся в другом,

более крупном фрагменте, а когда две метавершины различаются только набором атрибутов.

Метавершины mv_{1-4} благодаря введению элементов класса Фрагмент также образуют отдельный класс узлов плоского графа, поскольку могут быть связаны друг с другом только через посредничество элемента класса Фрагмент и таким образом теперь тоже не могут быть смежными узлами в «плоском» представлении.

С введением дополнительной «прослойки» на данном этапе преобразования метаграфа был получен полноценный четырёхдольный граф, состоящих из элементов четырёх типов: Вершина, Ребро, Метавершина, Фрагмент. Рисунок 6 показывает вариант преобразования метаграфа к плоскому четырёхдольному графу. На основании этого рисунка становится понятно, что графовые СУБД могут использоваться для хранения метаграфов. Рёбра, Метавершины и Фрагменты легко «преобразовать» в вершины.

Однако, несмотря на теоретическую правильность ввода нового класса Фрагмент создание таким образом четырёхдольного, а не трёхдольного графа не может не вызывать сомнений. Ввод этого класса для удовлетворения формального определения n -дольного класса является лишним с практической и, самое главное, смысловой точки зрения.

Метавершины в ходе работы с метаграфом могут меняться. Это означает, что использование несколькими из них одного и того же фрагмента станет невозможным, если изменится одна из таких метавершин. В таком случае придётся создавать копию фрагмента со всеми входящими в его состав элементами метаграфа, что не всегда удобно и может создать огромную дополнительную нагрузку на вычислительную систему.

Чтобы выделение отдельного класса узлов метаграфа имело смысл, нужно чтобы эти вершины имели какие-то свойства и были независимыми от других объектов. Фрагменты являются составными частями метавершин, что означает, что они явно привязаны к ним. Они не обладают собственными свойствами,

следовательно, не обладают свойством эмерджентности, так что с точки зрения метаграфа выделение фрагментов не нужно.

Помимо этого, важно понять, что будет происходить на практике по ходу выполнения алгоритма эмбединга. В начале будет произведён перевод метаграфа в плоский граф путём составления матрицы смежности и списка «вершин» плоского графа. Список нужен для сохранения информации о свойствах самих вершин. Это означает, что количество вершин-фрагментов по количеству примерно равно числу метавершин, что увеличивает матрицу смежности в среднем на 20-40% по сравнению с её размерами без введения фрагментов.

Если не вводить класс Фрагмент, то связь вложенности между метавершинами сохраняется, однако стоит заметить, что эта связь сохраняется и между парами узлов типов метавершина-вершина. Такая связь не обладает ни атрибутами, ни свойством направленности, что делало бы необходимым введение дополнительных элементов.

Таким образом, целесообразным будет использование трёхдольного графа со следующими классами: Вершина, Ребро, Метавершина. Пример такого графа, составленный на основе метаграфа с рисунка 1, показан ниже, на рисунке 7.

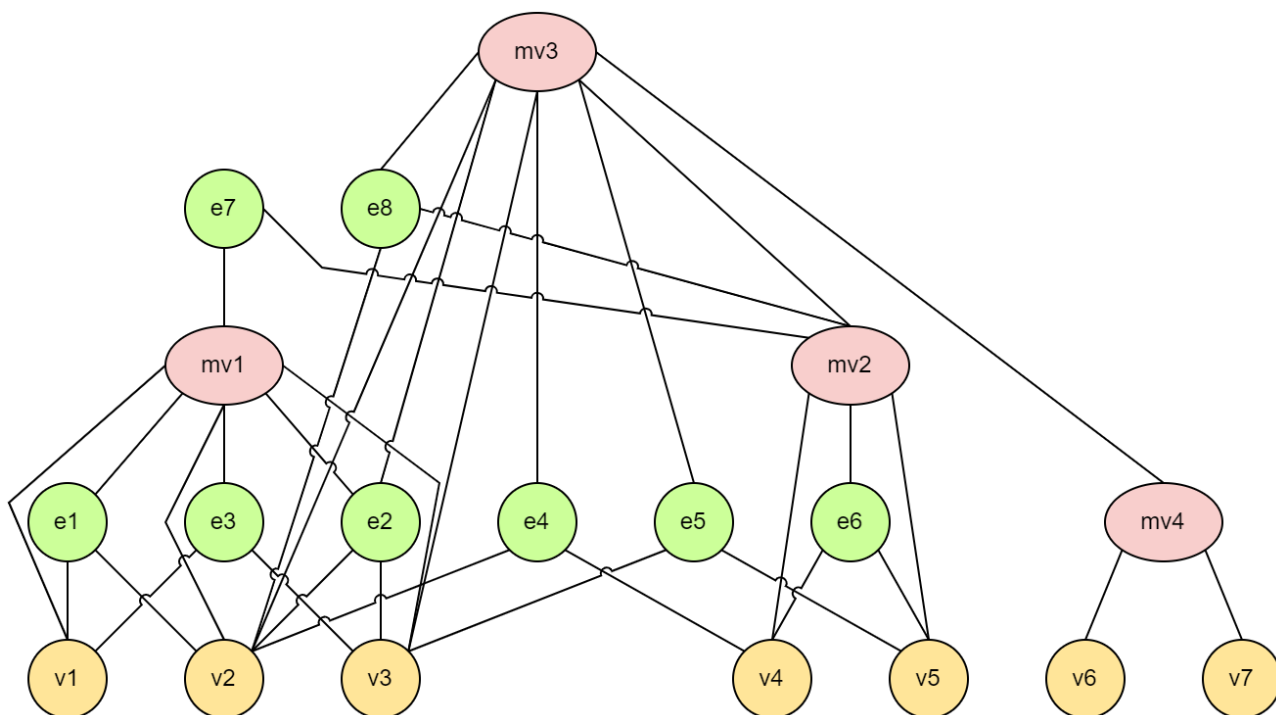


Рисунок 7 – Трёхдольный граф

2.3. Алгоритм преобразования метаграфа в трёхдольный граф

Основная идея для отображения метаграфа – преобразование иерархической модели графа в плоский граф. Однако, невозможно преобразовать иерархическую модель графа в плоский граф напрямую, поэтому ключевой идеей является использование многодольных графов.

Рассмотрим модель плоского графа как:

$$FG = \langle FG^V, FG^E \rangle,$$

где FG^V – набор вершин графа; FG^E – набор рёбер графа.

Модель метаграфа, если учитывать исследование предыдущего раздела, рассматривается так:

$$MG = \langle V, MV, E \rangle,$$

где MG – метаграф; V – множество вершин метаграфа; MV – множество метавершин метаграфа; E – множество рёбер метаграфа.

Тогда плоский граф рассматриваться как трёхдольный граф метаграфа FG :

$$\begin{aligned} G &= \langle G^{VERT}, G^{EDGE} \rangle, \\ G^{VERT} &= \langle FG^V, FG^E, FG^{MV} \rangle, \\ FG^V &\leftrightarrow MG^V, FG^E \leftrightarrow MG^E, FG^{MV} \leftrightarrow MG^{MV}. \end{aligned}$$

Набор вершин графа G^{VERT} может быть разделено на 3 независимых набора FG^V, FG^E, FG^{MV} . Существует 3 изоморфных преобразования между вершинами метаграфа, метавершинами, ребрами и соответствующими множествами:

$$G^{VERT}: FG^V \leftrightarrow MG^V, FG^E \leftrightarrow MG^E, FG^{MV} \leftrightarrow MG^{MV}.$$

Набор G^{EDGE} содержит информацию о связях между вершинами, метавершинами и рёбрами в исходном метаграфе.

С точки зрения n -дольных графов, не важно, является ли метаграф MG ориентированным, поскольку его рёбра представляются в графе G как вершины и знак ориентирования может считаться атрибутом вершины-ребра в FG . В этом графе отношения между вершинами и метавершинами рассматриваются как специальные сущности более высокого уровня, которые включают элементы более низкого уровня.

Учитывая всё вышеизложенное логично представить алгоритм перевода метаграфовой модели в модель трёхдольного графа похожим на алгоритм поиска в глубину для плоского графа. Вложенные элементы метавершины можно представить в виде вершины плоского графа и связями между этой вершиной и вложенными элементами. Результатом работы алгоритма является матрица смежности плоского графа. Для удобства весь метаграф является mg .

Алгоритм преобразования модели метаграфа в трёхдольный граф состоит из двух методов:

- Обход метаграфа в глубину (Metagraph Depth-First Search, MDFS);
- Заполнение матрицы смежности (Adjacency matrix filling, AMF).

Заполнение производится на основе результата на основе результатов работы метода MDFS.

Ниже представлены текстовые краткие версии этих двух алгоритмов. Пояснения к ним приведены ниже.

- MDFS: Проход в цикле по всем элементам $ev \in mg$, где $ev \in MV \cup V \cup E$:

- а. Если ev не находится в списке vet , то есть если $ev \notin vet$:
 - 1) Добавить ev в список vet ;
 - 2) Если ev является метавершиной $ev \in MV$, то вызвать процедуру MDFS для ev .

- AMF: Проход в цикле по всем элементам $ev \in vet$ (vet – список):
 - а. Если ev является метавершиной:
 - 1) Для каждого элемента $ev' \in ev$, заполнить матрицу смежности следующим образом: $mt(i, j) = 1$ и $mt(j, i) = 1$, где i, j – индексы метавершины ev и элемента ev' в списке vet ;
 - б. Если ev является ребром $ev \in E$:
 - 1) Заполнить матрицу $mt(s, e) = 1$, $mt(e, s) = 1$, $mt(d, e) = 1$ и $mt(e, d) = 1$, где s, e, d – индексы элементов списка vet : s (source) – индекс элемента начальной вершины ребра ev , e (elem) – индекс элемента ребра ev , d (destination) – индекс элемента конечной вершины ребра ev .

Ниже, на рисунке 8, изображён простой метаграф. В результате работы алгоритма он будет преобразован в плоский трёхдольный граф, который изображён на рисунке 9.

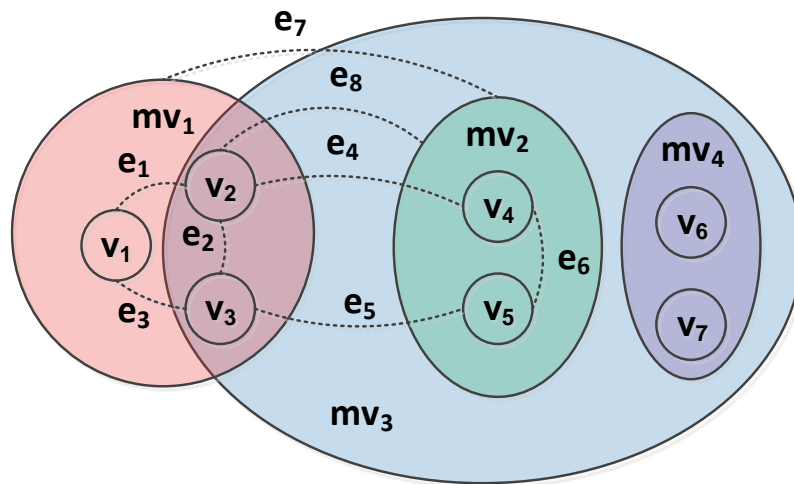


Рисунок 8 – Пример простого метаграфа

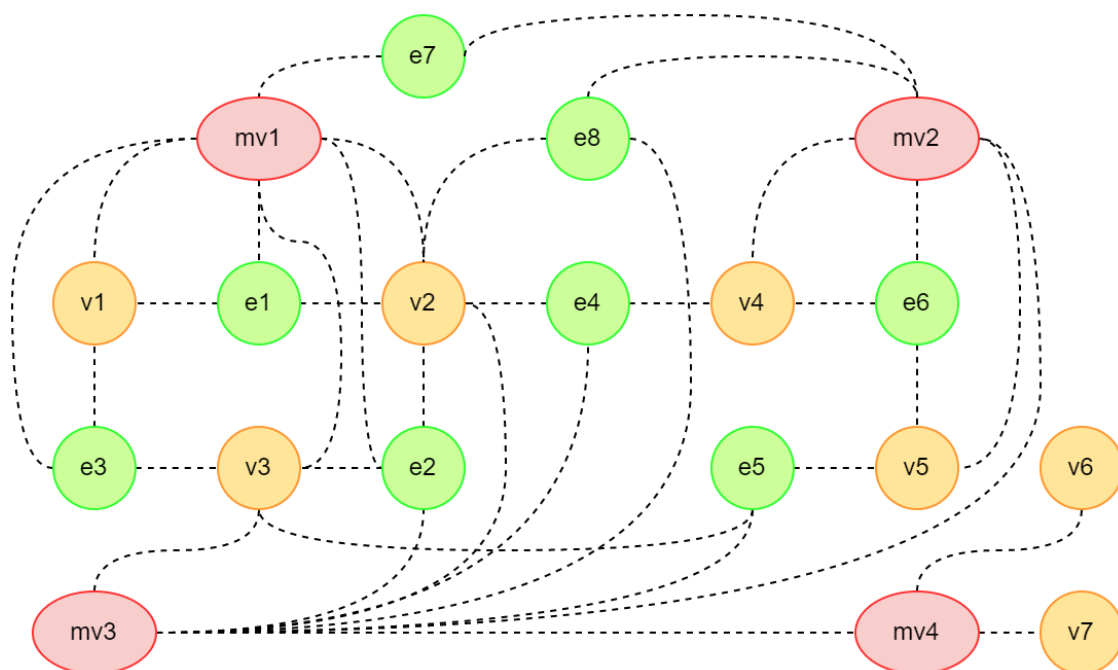


Рисунок 9 – Полученный алгоритмом трёхдольный граф

Как видно из рисунка 9, плоский граф состоит из четырёх типов вершин: вершин-метавершин mv_{1-4} , вершин-вершин v_{1-7} , и вершин-рёбер e_{1-8} .

Все вершины-рёбра соединены с другими вершинами графа рёбрами, для которых не важна направленность исходных рёбер метаграфа, поскольку для вершин-рёбер устанавливается атрибут направленности. Остальные элементы полученного четырёхдольного графа соединены друг с другом в соответствии с топологией исходного метаграфа.

2.4. Подходы к эмбедингу графов

Описание способов эмбединга графов в этом и следующих разделах в основном опираются на обзоры [15-16], а также на статьи, посвящённые конкретным алгоритмам эмбединга.

Графовая аналитика зарекомендовала себя как практичная и важная область знаний, однако сейчас большая часть её методов страдает от высоких затрат памяти компьютера, больших вычислительных и временных затрат. Объём затрат не позволяет сделать графовую аналитику доступной для всех.

Много исследований посвящено эффективному проведению дорогостоящей графовой аналитики. Примеры такие исследований включают в себя инфраструктуры обработки данных распределенного графа (например, GraphX [17], GraphLab [18]), компактное хранилище графов [19], которое ускоряет ввод-вывод и вычислительные затраты и т. д.

Эмбединг графа – это эффективный способ решения задачи аналитики графов. Его эффективное использование позволяет перевести граф в низкоразмерное векторное пространство, в котором сохранится информация о графе. После получения представления можно обрабатывать сами данные так же эффективно, как и другие неграфовые данные.

Эмбединг графа в низкоразмерное пространство является комплексной задачей. Сложности решения задачи вложения графа зависят от постановки задачи предметной области, которая состоит, в частности, из требований к входным и выходным данным.

Результатом эмбединга графа является вектор, представляющий целый граф или какую-то определённую часть графа, например, только вершины или только рёбра графа. На рисунке 10 показан пример эмбединга вершин графа в двухмерное пространство.

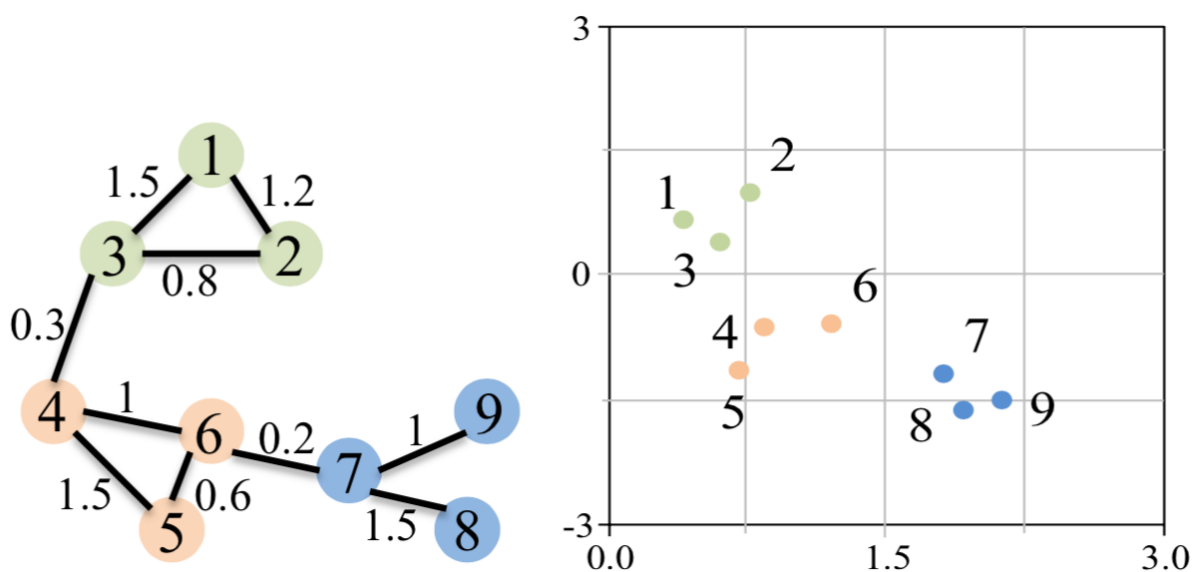


Рисунок 10 – Эмбединг вершин графа в двухмерное векторное пространство

Наиболее распространенный тип эмбединга – это эмбединг узлов. Он может быть полезен при решении задач, связанных с узлами, например, классификация или кластеризация узлов графа. Этот подход неизбежно представляет узлы, схожие по свойствам, близко расположенными друг к другу в векторном пространстве. Основным показателем качества такого эмбединга является видимость различия между векторами таких узлов графа. То есть хороший алгоритм эмбединга любого типа должен сохранять сходство похожих узлов, но не сливать их в один.

В некоторых случаях задачи могут быть более сложными и связанными, например, с какими-то комбинациями, парами, тройками узлов, подграфом или графом целиком. Следовательно, помимо самого эмбединга и его встраивания в общее решение, необходимо решить, какой тип выходных данных необходимо от него получить.

Существуют разные типы эмбединга относительно типов выходных данных: эмбединг узлов, рёбер, гибридный эмбединг и эмбединг целого графа. Логично, что алгоритмы разных типов сталкиваются с разными проблемами по ходу проведения операции эмбединга и имеют разные критерии оптимального встраивания. Например, хорошее вложение узла сохраняет сходство с соседними узлами в векторном пространстве. Напротив, хорошее вложение целого графа представляет целый граф как вектор, так что сходство на уровне графа сохраняется.

Пример вложения рёбер в двухмерное пространство показан на рисунке 11 ниже.

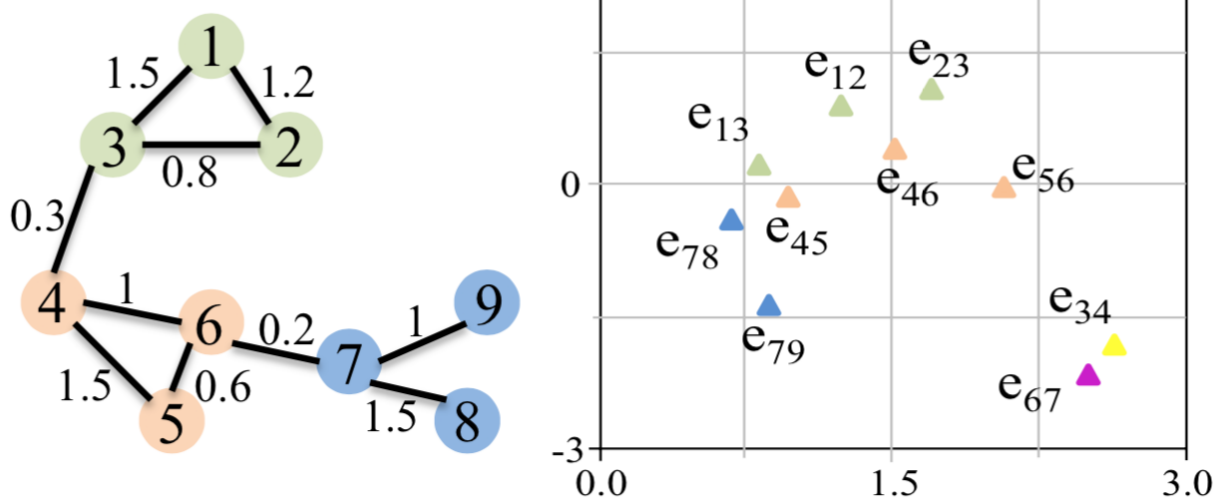


Рисунок 11 – Вложение рёбер в двухмерное пространство

Пример вложения подграфа в двухмерное пространство показан на рисунке 12 ниже.

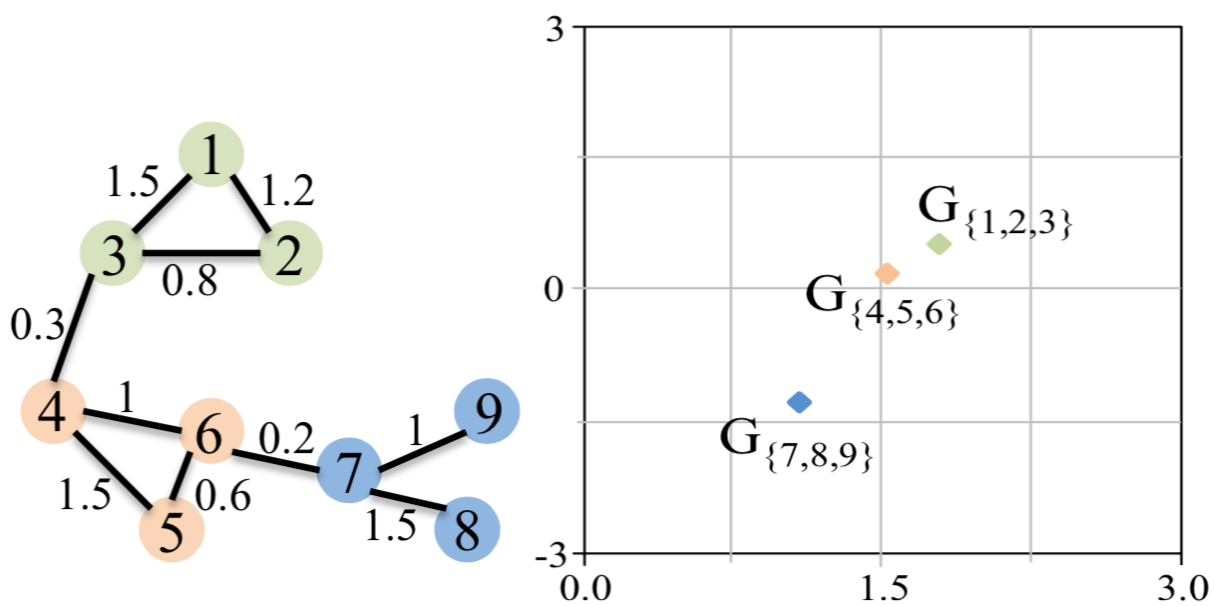


Рисунок 12 – Вложение подграфа в двухмерное пространство

Пример вложения графа в двухмерное пространство показан на рисунке 13 ниже.

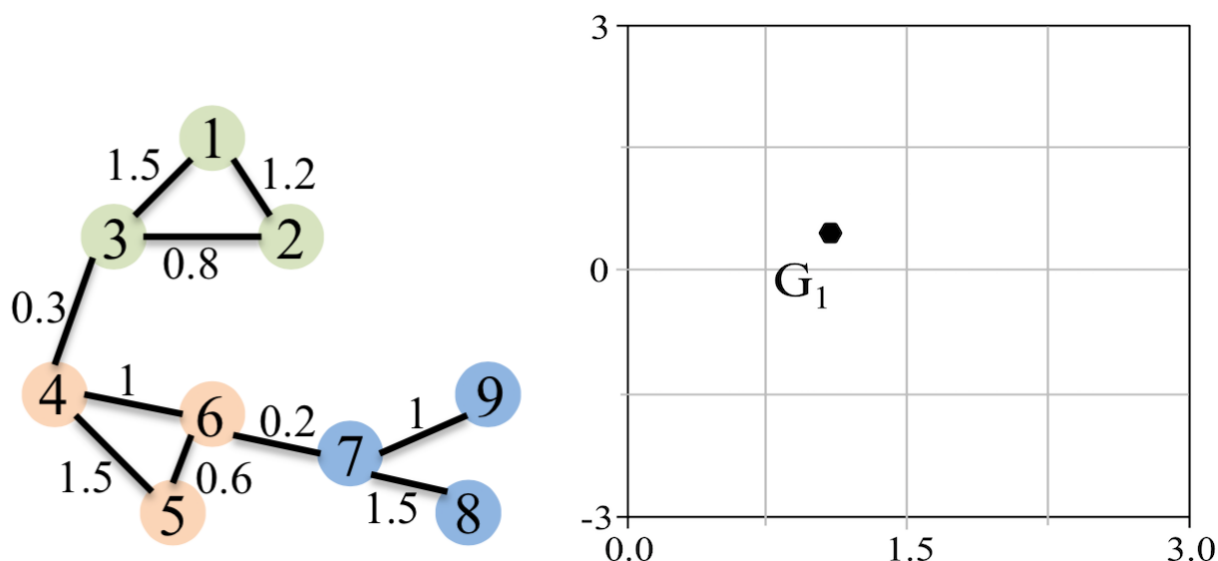


Рисунок 13 – Вложение графа в двухмерное пространство

Важно отметить, что в отличие от реляционной СУБД, где преобразование логической модели в физическую является однозначным отображением, про эмбединг графа и, тем более, метаграфа такого сказать нельзя. На основе одной логической модели с помощью различных методов можно сформировать различные результирующие векторные пространства физической модели. Полученные модели, как правило, оптимизированы для выполнения конкретных алгоритмов над графами.

Значительная часть наиболее простых техник эмбединга основана на том факте, что ребрам графа приписывается определенная числовая метрика, которую можно трактовать как расстояние между вершинами графа или пропускную способность каналов, соединяющих вершины.

2.5. Входные данные эмбединга графов

Входными данными для вложения графов является матрица смежности графа. При осуществлении эмбединга, то есть вложения графа, используются

такие метрики как близость первого порядка и близость второго порядка. Если граф не взвешенный, то матрица смежности имеет одинаковые значения для связей, обычно таким ячейкам присваивается единица, а остальным – нуль. Если же он взвешенный, то в ячейку определённой связи ставится вес этого ребра.

В разделе ниже будет рассмотрен эмбединг многодольных графов. В соответствии с результатами исследования, завершённого в разделе 2.3 данной работы, в экспериментальной части будет произведена работа лишь с невзвешенным трёхдольным графом.

2.6. Особенности многодольных (трёхдольных) графов

Как и говорилось выше, многодольные графы состоят из вершин разного типа (класса), которые составляют доли такого графа. Вершины в многодольных графа обычно несут разный смысл, имеют отличительные свойства или вообще описывают разные объекты. Пример трёхдольного графа представлен на рисунке 14. Вершины этого графа делятся на три класса: «зеленый», «синий» и «красный».

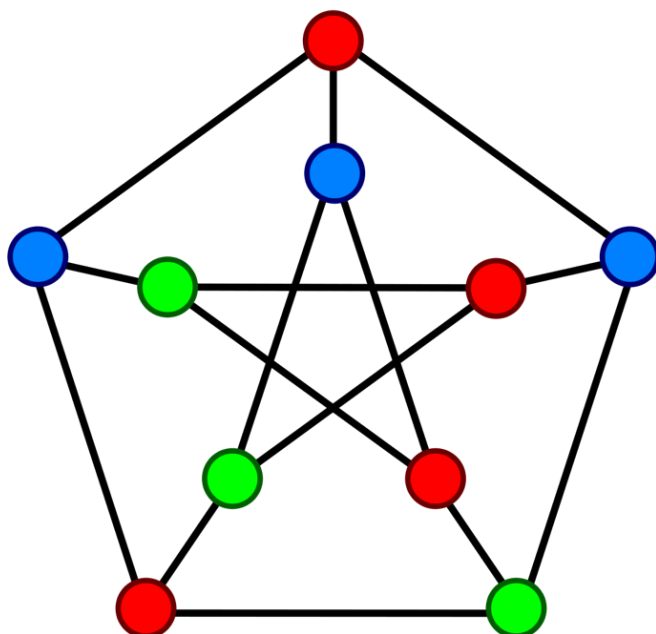


Рисунок 14 – Пример трёхдольного графа

Для многодольных графов существует много сценариев применения. Ниже приведены и кратко описаны три примера.

Сайты сообщества вопросов и ответов (сQA) – это краудсорсинговый сервис в Интернете, который позволяет пользователям размещать на веб-сайте вопросы, на которые отвечают пользователи [20]. Российским пользователям и it-специалистам, в частности, отлично известны такие ресурсы, как «Ответы Mail.ru», «StackOverFlow» или «CyberForum». Множество форумов построены по такой системе и могут действовать в интересах компании-создателя. Если представить граф сQA, то там, очевидно, будут такие типы узлов: вопрос, ответ, пользователь.

Мультимедийные сети – это сети, которые заключают в себе информацию в виде, например, изображения, текста, аудиозаписи и так далее.

В работах [21-22] рассмотрены графы, содержащие два вида узлов: изображение и текст, а также три вида ссылок: совместное появление изображения-изображения, текста-текста и изображения-текста.

В работе [23] рассматривается социальное взаимодействие с пользовательским узлом и узлом изображения. Ссылки на пользовательские изображения используются в данной работе для встраивания пользователей и изображений в одно и то же пространство. Благодаря этому подходу возможно напрямую сравнивать пользователей и изображения для получения рекомендаций по изображениям.

В работе [24] рассматривается граф кликов, который содержит изображения и текстовые запросы. Вес ребра запроса изображения указывает на количество кликов по изображению при данном запросе.

В графах знаний сущности, предстающие в виде узлов, и отношения, предстающие в виде рёбер, бывают разных типов. В работе [25] рассматривается граф знаний о фильмах. В этом графе типами сущностей могут быть «продюсер», «актёр», «фильм» и так далее. Типы отношений могут быть следующими: «производить», «финансировать», «играть в». Работы [26-28], например,

посвящены внедрению графов знаний и описывают сложности, обычно сопровождающие такой процесс.

Главной задачей данной работы является представление метаграфовой модели знаний в виде многодольного графа и проведение эмбединга вершин этого графа. Метаграфовая модель не ограничивает количество элементов того или иного типа в своём составе, поэтому стоит учитывать возможный дисбаланс при эмбединге. Поскольку в рамках данной работы встраивание трёхдольного графа происходит в единое векторное пространство неизбежной сложностью становится сохранение согласованности между различными типами объектов.

2.7. Выходные данные эмбединга графов

Выходными данными эмбединга графов является набор векторов, представляющих граф или часть графа. На предыдущих этапах исследования был получен трёхдольный плоский граф, узлами (вершинами) которого являются вершины, рёбра и метавершины исходного метаграфа. Между новыми вершинами в плоском графе существуют лишь ненаправленные связи, благодаря которым плоский граф остаётся изоморфным с исходным метаграфом. Таким образом в данной работе стоит задача вложения вершин всех трёх типов в единое векторное пространство.

Вложение узла представляет каждый узел как вектор в низкоразмерном пространстве. Узлы, близкие в исходном метаграфе должны быть вложены так, чтобы иметь похожие векторные представления. Различия между методами вложения графов заключаются в том, как они определяют «близость» между двумя узлами.

Ниже вводятся два определения на основе классической математической теории, изложенной, в частности, в работе [29].

Близостью первого порядка между вершиной v_i и вершиной v_j является вес e_{ij} . Вершины из пары тем более «близки», чем больше значение веса e_{ij} ребра, которым они соединены. Пусть близость первого порядка между вершиной v_i и вершиной v_j обозначается как s_{ij}^1 . Тогда $s_i = [s_{i1}, s_{i2}, \dots, s_{in}]$ – близость первого порядка между вершиной i и остальными вершинами. В таком случае для вершины с номером 7 ($i = 7$) плоского графа, который показан на рисунке 15 ниже, $s_7^1 = [0, 0, 0, 0, 0, 0.2, 0, 1.5, 1]$.

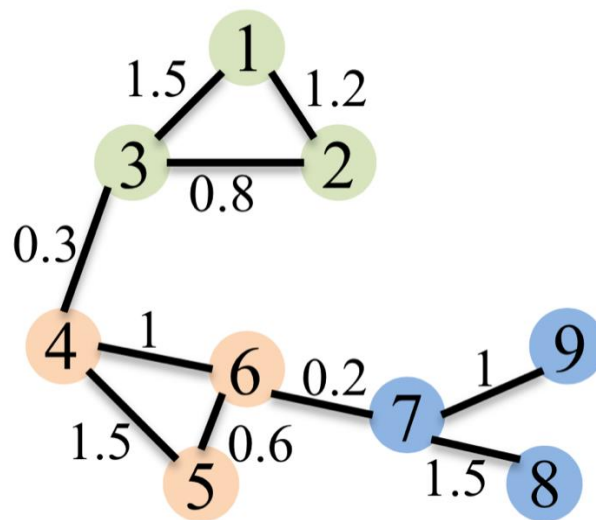


Рисунок 15 – Пример плоского графа

Близость второго порядка сравнивает сходство структур соседних узлов. Чем больше схоже окружение двух узлов, тем больше значение близости второго порядка между ними. Значение близости второго порядка между вершиной v_i и вершиной v_j напрямую зависит от значений близости первого порядка каждой из этих вершин. Для определения такой близости в данной работе предлагается использовать косинусное сходство [29].

Исходя из рисунка 15 для довольно похожих вершин с номерами 6 и 7 вектора близости s_i^1 примут следующие значения: $s_6^1 = [0, 0, 0, 1, 0.6, 0, 0.2, 0, 0]$, $s_7^1 = [0, 0, 0, 0, 0, 0.2, 0, 1.5, 1]$. В таком случае косинусное расстояние s_{ij}^2 определяется по формуле (7):

$s_{ij}^2 = cosine(s_i^1, s_j^1) = \frac{\sum_{k=1}^n s_i^1 * s_j^1}{\sqrt{\sum_{i=1}^n (s_i^1)^2} * \sqrt{\sum_{j=1}^n (s_j^1)^2}},$	(7)
---	-----

В случае вершин 6 и 7 значения метрики близости второго порядка s_{67}^2 будет равно нулю, что будет означать, что они ортогональны друг другу в векторном пространстве.

В случае вершин 5 и 6, для которых $s_5^1 = [0, 0, 0, 1.5, 0, 0.6, 0, 0, 0]$, $s_6^1 = [0, 0, 0, 1, 0.6, 0, 0.2, 0, 0]$, значение метрики близости второго порядка s_{56}^2 будет:

$$s_{56}^2 = \frac{1.5}{\sqrt{1.5^2 + 0.6^2} * \sqrt{1 + 0.6^2 + 0.2^2}} = \frac{1.5}{\sqrt{2.25 + 0.36} * \sqrt{1 + 0.36 + 0.04}} = \frac{1.5}{\sqrt{2.61} * \sqrt{1.4}} = \frac{1.5}{1.62 * 1.18} = 0.78.$$

Такой результат означает, что вершины 5 и 6 довольно схожи.

Итого, две эти метрики близости используются для расчета схожести узлов.

2.8. Алгоритмы эмбединга

Существует множество различных алгоритмов эмбединга графов. Некоторые из них, рассмотренные, в частности, в работе [55], предназначены для анализа текстов.

Как правило, алгоритмы эмбединга графа направлены на представление его в низкоразмерном пространстве и сохранение информации о как можно большем количестве свойств графа. Разница между алгоритмами, в основном, заключается в том, как и какие свойства «хочет» сохранить алгоритм. Алгоритмы используют различные «сходства» узлов графа и представляют их в векторных пространствах тоже по-разному. Ниже будут кратко описаны механизмы работы алгоритмов, их отличительные свойства и преимущества при эмбединге плоских графов. Также будет пояснено, на какие свойства графа они обращают внимание.

Практически все существующие алгоритмы эмбединга производят встраивание графа в евклидово пространство, однако, как известно, существуют и другие типы пространств, неевклидовых. К таким алгоритмам относится и Riemannian TransE [53] или MuRP [54], однако он плохо работает с графами большой размерности, поэтому он и ему подобные не рассматриваются в данной работе.

Эмбединг на основе матричной факторизации представляет связи графа в форме матрицы и разлагает эту матрицу для получения вложения узла [30]. Матрицы, используемые для представления соединений, включают в себя: матрицу смежности узлов, матрицу Лапласа, матрицы сингулярного разложения, матрицу вероятностей перехода узлов, а также другие.

В экспериментальной части данной работы будет рассмотрен результат эмбединга двумя алгоритмами этого типа. Будут использованы «Собственные карты Лапласа» (Graph Laplacian eigenmaps, Laplacian eigenmaps, LE) [31-35] и «Встраивание с сохранением близости высокого порядка» (HOPE – High-Order Proximity preserved Embedding) [36], которые описаны в подразделах 2.8.1 и 2.8.2 ниже.

Глубокое обучение (Deep Learning, DL) показало отличную производительность в широком спектре областей знаний и исследований, таких как компьютерное зрение, языковое моделирование и т. д. Применение моделей DL давно зарекомендовало себя в области обработки естественного языка, и одним из примеров таких алгоритмов являются методы, построенные на основе моделей word2vec [37]. Однако позже исследователи показали на примерах алгоритмов DeepWalk [38] и Node2Vec [39].

В экспериментальной части данной работы будет рассмотрен результат эмбединга метода Node2Vec, который описан ниже в подразделе 2.8.3.

2.8.1. Встраивание с сохранением близости высокого порядка (HOPE – High-Order Proximity preserved Embedding)

Существующие методы вложения графов не могут хорошо сохранить асимметричную транзитивность, которая является критическим свойством ориентированных графов. В случае метаграфа это свойство также важно учитывать. Асимметричная транзитивность показывает корреляцию между направленными ребрами, то есть, если существует направленный путь от v_1 до v_2 , то, вероятно, существует направленное ребро от v_1 до v_2 . Асимметричная транзитивность может помочь в захвате структур графов и восстановлении из частично наблюдаемых графов. Метод HOPE предлагает идею сохранения асимметричной транзитивности путем приближения к близости высокого порядка, основанной на асимметричной транзитивности.

Входными данными алгоритма является матрица смежности графа, размерность K векторного пространства, специальный параметр масштабности выходных векторов, а также информация о направленности рёбер. Следуя работам [36] и [40], сначала определяется разложение матрицы смежности по сингулярным значениям (Singular Value Decomposition, SVD) на матрицы M_g и M_l . Затем масштабируемый алгоритм вычисляет (аппроксимирует) близость высокого порядка на основе базовой матрицы смежности: сначала вычисляются вектора сингулярных значений $\{\sigma_1^g, \dots, \sigma_K^g\}$ и $\{\sigma_1^l, \dots, \sigma_K^l\}$ и соответствующие сингулярные вектора начальных и конечных узлов графа $\{v_1^s, \dots, v_K^s\}$ и $\{v_1^t, \dots, v_K^t\}$. Затем вычисляется обобщённый вектор по формуле (8) ниже:

$$\sigma_i = \frac{\sigma_i^l}{\sigma_i^g}, i = \{1, \dots, K\} \quad (8)$$

Затем вычисляются набор векторов для стартовых вершин и конечных вершин в евклидовом векторном пространстве размерности K по формулам (9) – (10) ниже:

$$U_s = [\sqrt{\sigma_1} * v_s^1, \dots, \sqrt{\sigma_K} * v_s^K] \quad (9)$$

$$U_t = [\sqrt{\sigma_1} * v_t^1, \dots, \sqrt{\sigma_K} * v_t^K] \quad (10)$$

Таким образом метод NOPE аппроксимирует близости высокого порядка значительно лучше, чем другие алгоритмы, и по результатам экспериментов, проведённых авторами метода [36], превосходит другие современные алгоритмы в задачах реконструкции, прогнозирования и рекомендации связей.

2.8.2. Собственные карты Лапласа (Laplacian Eigenmaps)

Свойства графа можно интерпретировать как сходство парных узлов, которые оформляются в виде матрицы смежности W , являющейся входными данными алгоритма вместе с самим графом G . Цель вложения графа - представить каждую вершину графа как низкоразмерный вектор, сохраняющий сходства между парами вершин. Чем больше вес ребра, тем больше «схожесть» двух вершин графа.

Далее последует краткое описание сути алгоритма.

Пусть $y^* = [y_1, y_2, \dots, y_n]^T$ – набор векторов который представляет каждый из вершин графа в низкоразмерном пространстве. Оптимальный набор y^* пытается оптимизировать функцию (11):

$$y^* = \underset{y}{\operatorname{argmin}} \sum_{i \neq j} (y_i - y_j)^2 W_{ij}, \quad (11)$$

где W_{ij} – элемент матрицы смежности, вес ребра между вершинами v_i и v_j .

Оптимизация функции происходит под соответствующим ограничением. Эта целевая функция (2) не допускает большое расстояние между соседними вершинами. Поэтому сведение y^* к минимуму является попыткой убедиться, что если вершины v_i и v_j «близки», то разница между y_i и y_j тоже маленькая [41]. Поэтому мы имеем (12):

$$y^* = \underset{i \neq j}{\operatorname{argmin}} \sum (y_i - y_j)^2 W_{ij} = 2y^T L y, \quad (12)$$

где $L = D - W$ – Лапласиан графа [42]. D – диагональная матрица где $D_{ii} = \sum_{i \neq j} W_{ij}$. Чем больше значение D_{ii} , тем больше значение y_i .

В итоге задача состоит в минимизации (13) функции:

$$y^* = \underset{y^T D y = 1}{\operatorname{argmin}} (y^T L y) = \underset{y^T D y = 1}{\operatorname{argmin}} \left(\frac{y^T L y}{y^T D y} \right) = \underset{y^T D y = 1}{\operatorname{argmax}} \left(\frac{y^T W y}{y^T D y} \right), \quad (13)$$

где ограничение $y^T D y = 1$ нивелирует возможность произвольного масштабирования представления.

Существуют разные алгоритмы, основанные на этом методе: MDS [43], Isomap [44], LE [45], LLP [46], AgLLP [47], ARE [48], SR [31]. Эти алгоритмы используют матрицу W с другими значениями. Выбор матриц W и D может быть очень разным, но играет ключевую роль в этих алгоритмах вложения графа. Алгоритм LGRM [51] использует модель локальной регрессии для понимания структуры графа. Алгоритм LSE [52] использует локальную сплайн-регрессию для сохранения глобальной геометрии.

Также помимо создания подобных вариаций были предприняты попытки, описанные в работах [49-50], обобщить существующие методы, однако в экспериментальной части данной работы используется именно описанная выше базовая версия алгоритма.

2.8.3. Node2Vec

Алгоритмы эмбединга графов, основанные на методах глубокого обучения, сначала формируют набор случайных путей в графе, а потом с помощью методов глубокого обучения производят эмбединг на основе путей. Таким образом подобные алгоритмы, и Node2Vec в частности, сохраняют свойства путей графа.

В рамках этой работы в первую очередь необходимо сопоставить каждый узел метаграфа с точкой в низкоразмерном векторном пространстве. Совокупность точек должна максимально полно описывать структуру метаграфа, а точнее говоря, совокупность свойств его узлов и метаграфа в целом.

Метод Node2Vec основан на механизме случайных блужданий.

Ниже, на рисунке 16, приведена иллюстрация процедуры случайного блуждания в Node2Vec.

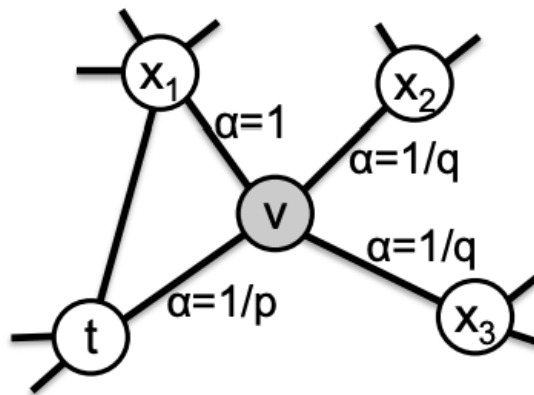


Рисунок 16 – случайное блуждание в Node2Vec

В данном алгоритме при переходе к следующему узлу учитывается не только текущее состояние блуждания, но и предыдущее состояние. Таким образом блуждание в алгоритме Node2Vec представляет собой Марковский случайный процесс второго порядка. Вводится два параметра p и q , характеризующих вероятность перехода от одних вершин к другим.

На рисунке 16 рассмотрен пример блуждания, которое прошло по пути от вершины t к вершине v и сейчас находится в вершине v . Вероятность π_{vx} перехода от вершины v к вершинам t и x_i равна $\pi_{vx} = \alpha_{pq}(t, x) * w_{vx}$, где

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p}, & \text{если } d_{tx} = 0 \\ 1, & \text{если } d_{tx} = 1 \\ \frac{1}{q}, & \text{если } d_{tx} = 2 \end{cases}$$

и d_{tx} обозначает кратчайшее расстояние между узлами t и x . Параметр p отвечает за немедленный возврат назад, а q за продвижение вперёд. Очевидно, что значение 0 для p и q недопустимо, но любое другое значение не гарантирует прямой проход без возврата назад и наоборот, не может дать гарантировать заикленность блуждания. В алгоритме Node2Vec, в отличие от базового DeepWalk, вероятность перехода в каждый из соседних узлов вычисляется при попадании в очередной узел.

3 Экспериментальная часть

3.1. Используемый метаграф

В качестве тестового метаграфа для экспериментов будет использоваться неориентированный метаграф, изображенный на рисунке 17.

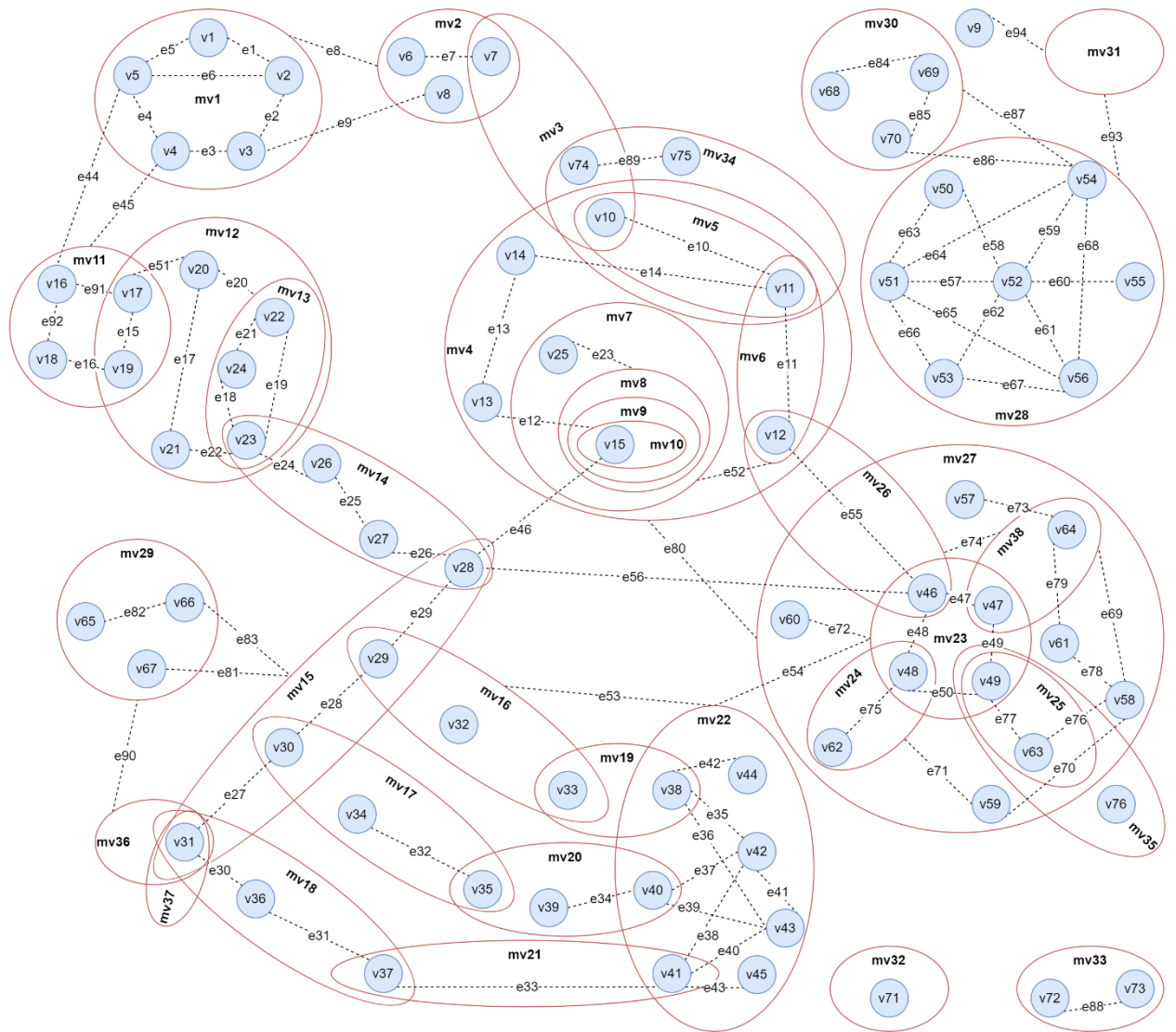


Рисунок 17 – Пример большого метаграфа

Этот метаграф состоит из 208 элементов, в число которых входят метавершины, вершины и ребра в соответствии с метаграфовой моделью, представленной в разделе 1.3 данной работы. В метаграфе, представленном на рисунке 17, имеется 38 метавершин, 76 вершин и 94 ребра.

Данный метаграф был составлен так, чтобы отразить возможные уровни вложенности и отношения между объектами метаграфа, сохраняя максимальную глубину вложенности в 6 единиц. Многие участки были составлены довольно похожими друг на друга для будущей подробной проверки корректности работы алгоритмов эмбединга. Ошибки могут возникнуть в том случае, если объекты имеют одинаковую структуру связей с другими объектами, то есть в случае, если близость второго порядка для них одинакова. Подробнее об этом описано в пункте 2.7 данной работы.

В данном тестовом метаграфе практически нет объектов, которые являются одинаковыми вследствие таких однотипных связей, однако, если разные алгоритмы эмбединга обнаружат такие узлы, это будет интересно разобрать.

Стоит отметить, что вершины v_{11} и v_{49} вложены не только в свои непосредственно родительские метавершины (пара mv_5 , mv_6 и пара mv_{23} , mv_{25} соответственно), но и в те метавершины, которые являются родительскими для одной из их непосредственных родительских метавершин. Таким образом вершина v_{11} включена в mv_6 , в mv_4 и в mv_5 , которая включена в mv_4 , а вершина v_{49} – в mv_{23} , mv_{35} и mv_{25} , которая включена в mv_{35} .

В таблицах А.1, А.2 и А.3 Приложения А приведены списки всех элементов метаграфа с рисунка 17.

3.2. Преобразование метаграфа в трёхдольный плоский граф

Для использования алгоритмов вложения, нужно представить исходный метаграф в виде плоского графа. Необходимо реализовать алгоритмы, описанный в разделе 2.3 данной работы.

Ниже приведу повтор краткого описания алгоритмов ниже.

– MDFS: Проход в цикле по всем элементам $ev \in mg$, где $ev \in MV \cup V \cup E$:

- а. Если ev не находится в списке vet , то есть если $ev \notin vet$:
 - 1) Добавить ev в список vet ;
 - 2) Если ev является метавершиной $ev \in MV$, то вызвать процедуру MDfS для ev .
- AMF: Проход в цикле по всем элементам $ev \in vet$ (vet – список):
 - а. Если ev является метавершиной:
 - 1) Для каждого элемента $ev' \in ev$, заполнить матрицу смежности следующим образом: $mt(i, j) = 1$ и $mt(j, i) = 1$, где i, j – индексы метавершины ev и элемента ev' в списке vet ;
 - б. Если ev является ребром $ev \in E$:
 - 2) Заполнить матрицу $mt(s, e) = 1$, $mt(e, s) = 1$, $mt(d, e) = 1$ и $mt(e, d) = 1$, где s, e, d – индексы элементов списка vet : s (source) – индекс элемента начальной вершины ребра ev , e (elem) – индекс элемента ребра ev , d (destination) – индекс элемента конечной вершины ребра ev .

На рисунке 18 изображены блок схемы алгоритмов преобразования метаграфа в трёхдольный граф.

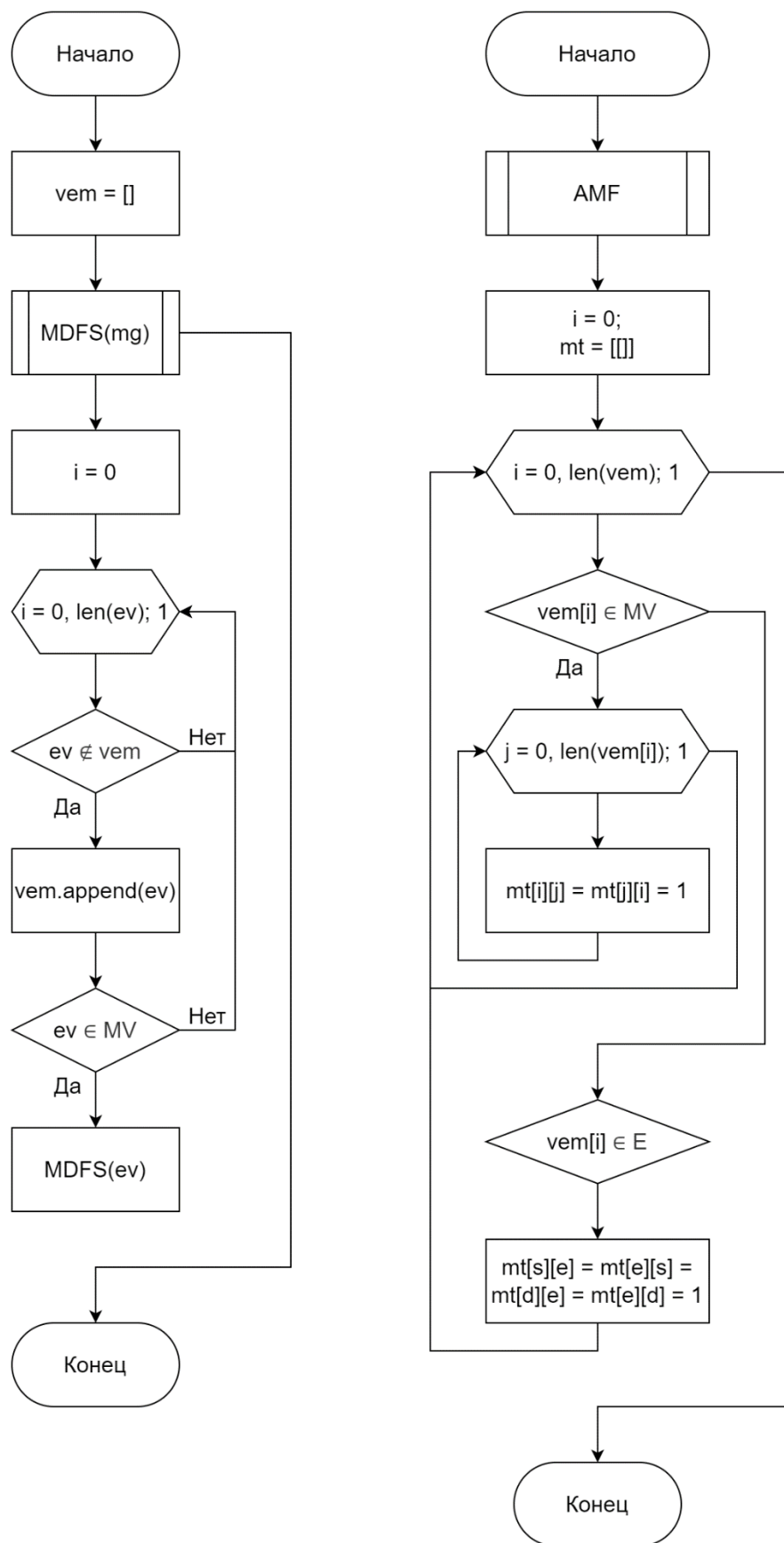


Рисунок 18 – Подпрограммы MDFS и AMF

С помощью подпрограммы MDFS все узлы метаграфа записываются в список *vet*. Если узел при рассмотрении оказывается метавершиной, то для этого узла сразу после записи информации в *vet* происходит рекурсивный вызов подпрограммы MDFS для сбора информации обо всех вложенных узлах. Таким образом алгоритм напоминает алгоритм рекурсивного обхода обычного графа в глубину.

Вторая подпрограмма AMF формирует матрицу смежности *mt*, которая отражает все связи в новом трёхдольном графе. Рёбра, метавершины и вершины все преобразуются в вершины своего класса в трёхдольном графе. При записи файла связей, который и использовался алгоритмами эмбединга, различным связям назначались уникальные веса: Метавершина-узел – 10, Вершина-узел – 2, Ребро-узел – 1.

3.3. Эксперименты

В подразделах ниже будут показаны результаты работы различных популярных современных алгоритмов эмбединга плоских графов, которые будут работать исключительно на основе матрицы смежности и информации о классах узлов трёхдольного графа. В качестве результатов экспериментов будут визуально представлены получившиеся эмбединги и небольшие разборы качества каждого из них.

В эксперименте использовались следующие алгоритмы:

1. Встраивание с сохранением близости высокого порядка (HOPE – High-Order Proximity preserved Embedding);
2. Собственные карты Лапласа (LE – Laplacian Eigenmaps);
3. Node2Vec.

3.3.1. Эксперимент с алгоритмом Встраивание с сохранением близости высокого порядка (High-Order Proximity preserved Embedding, HOPE)

При проведении операции вложения был получен результат, представленный на рисунке 19. Здесь и в дальнейшем метавершина обозначается красным кружком, вершина – синим квадратом поменьше, ребро – зелёным полным ромбом одного с вершиной размера.

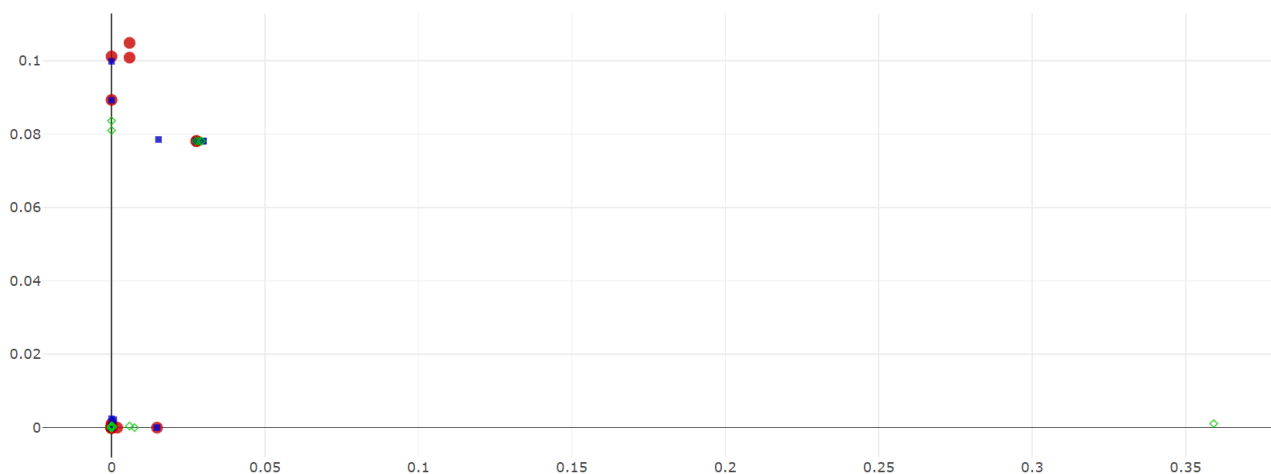


Рисунок 19 – двухмерный график эмбединга метаграфа алгоритмом HOPE

Как ясно из рисунка 19, результаты эмбединга неоднозначны, поскольку невозможно отличить друг от друга большую часть узлов. Ниже, на рисунке 20, приведён укрупнённый вид левого нижнего участка.

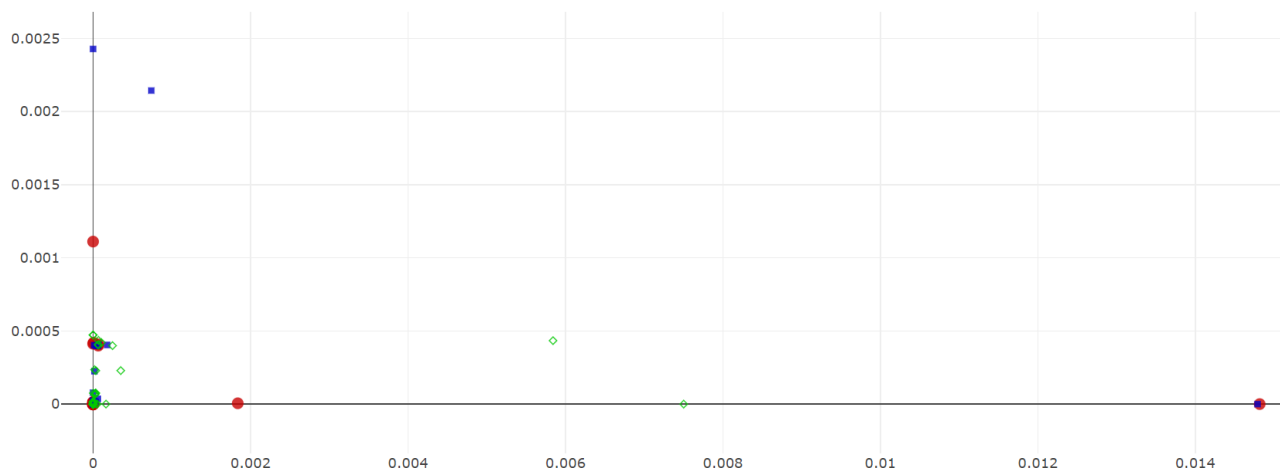


Рисунок 20 – Левый нижний участок графика с рисунка 19

Как ясно из рисунка 20, узлы находятся невероятно близко друг к другу, но не сливаются воедино даже на двухмерной плоскости. Если рассмотреть ещё ближе различные части этого графика, то можно заметить рёбра e_{64} , e_{65} и e_{66} . Они показаны на рисунке 21 ниже.



Рисунок 21 – Эмбединг рёбер e_{64} , e_{65} и e_{66} по методу NOPE

На рисунке 22 ниже показаны рёбра на исходном графе. Также на рисунке обозначены некоторые ключевые закономерности, которые делают похожими эти рёбра.

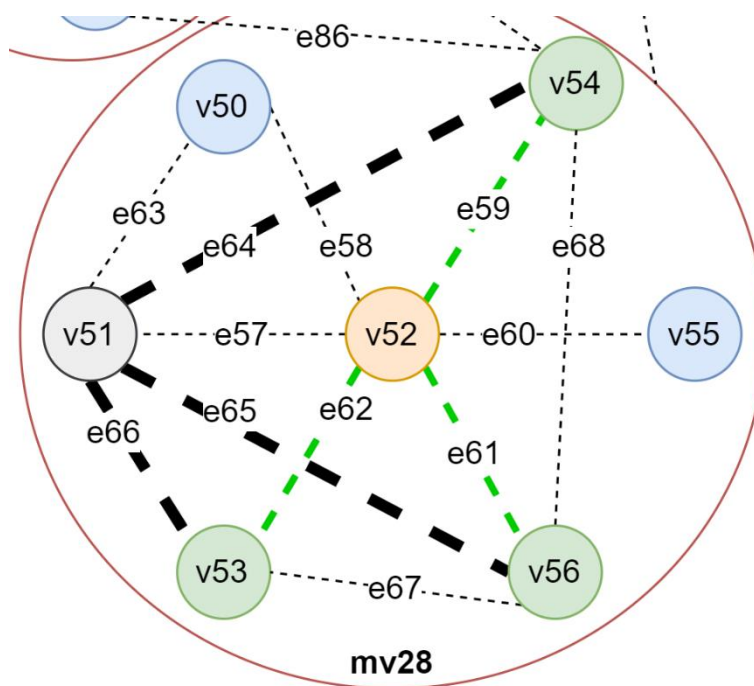


Рисунок 22 – Сходство рёбер e_{64} - e_{66} в исходном метаграфе

Как видно из рисунка 22 и таблицы 3, все три ребра исходят из одной вершины v_{51} . Их цели, вершины v_{54} , v_{53} и v_{56} соответственно вместе с исходной вершиной v_{51} вложены в метавершину m_{28} и не вложены ни в какие другие метавершины. Если рассматривать связь рёбер по степени близости более высокого порядка, то уже видны различия между рёбрами. Вершина v_{54} соединена с другими объектами за пределами m_{28} , что нельзя сказать о v_{53} и v_{56} . К тому же, все три вершины в целом связаны с другими объектами разным количеством и направлением рёбер. Поэтому результат эмбединга для данных рёбер можно считать удовлетворительным.

Ниже приведены некоторые основные статистические данные по выборке, которую представляют собой результаты эмбединга методом NOPE в двухмерном евклидовом пространстве.

Таблица 5 – Статистические данные

Статистика	X	Y
Максимум	0.3591723	0.1048563
Минимум	$-9.60831 \cdot 10^{-18}$	$-1.2197 \cdot 10^{-18}$
Среднее значение	0.003461	0.007815
Медиана	$8.142563 \cdot 10^{-10}$	$1.961766 \cdot 10^{-08}$
Стандартное отклонение	0.025589	0.0246059

График плотности распределения значений по осям x и y представлен на рисунках 23 и 24.

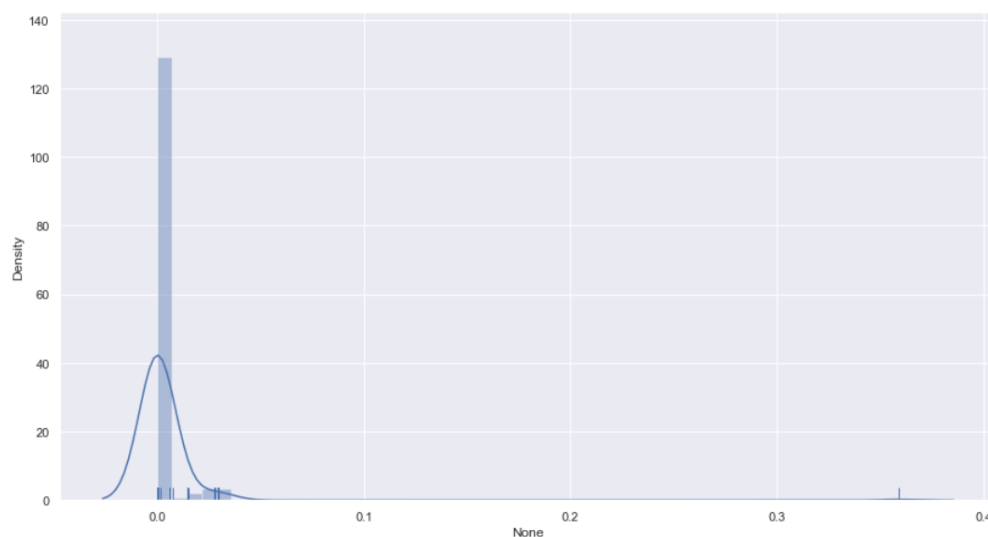


Рисунок 23 – Плотность распределения по оси X для NOPE

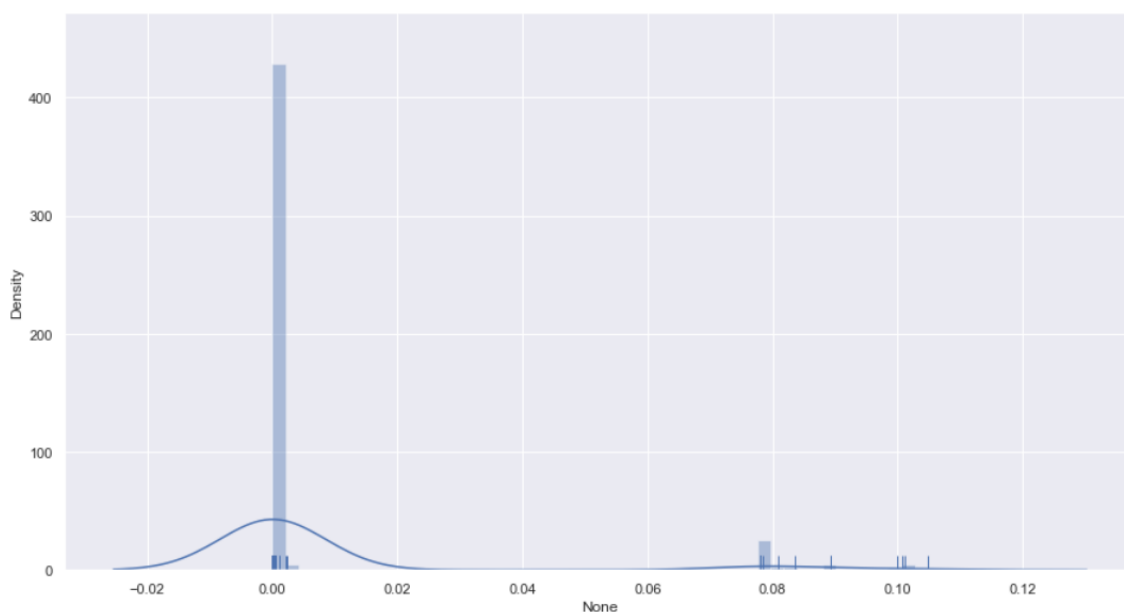


Рисунок 24 –Плотность распределения по оси Y для NOPE

Графики плотностей распределения показывают очень нестабильную плотность данных, имеющую подавляющую концентрацию на практически нулевой отметке. Данные картины плотностей распределения обусловлены алгоритмом вложения.

Ниже, на рисунке 25, приведён график плотности распределения для обеих плоскостей, что подтверждает изложенное выше.

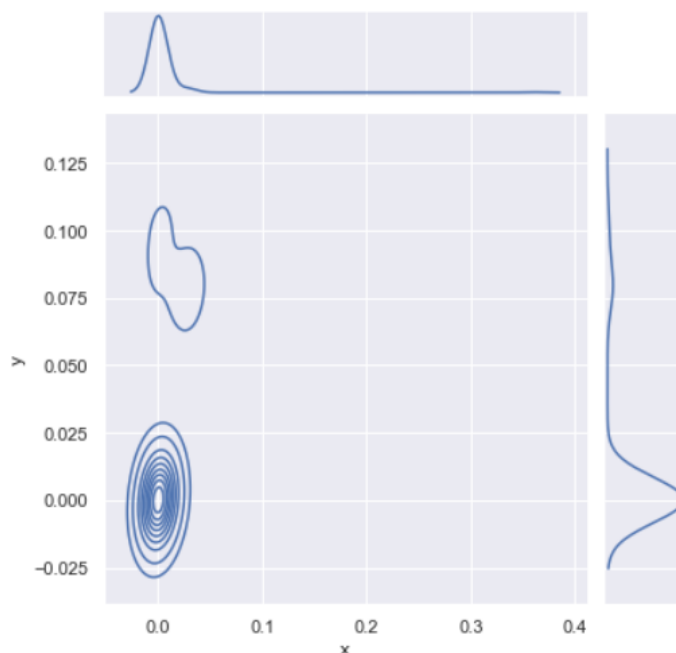


Рисунок 25 – Плотность распределения NOPE-эмбединга на плоскости

Такое плотное распределение, на самом деле, говорит лишь о том, насколько сильно схожи между собой узлы метаграфа. Чтобы проводить успешное обучение различных моделей машинного обучения, используя получившийся эмбединг, необходимо будет учитывать возможную невероятную плотность распределения исходных данных.

3.3.2. Эксперимент с алгоритмом Собственные карты Лапласа (Laplacian Eigenmaps, LE)

При проведении операции вложения был получен результат, представленный на рисунке 26.

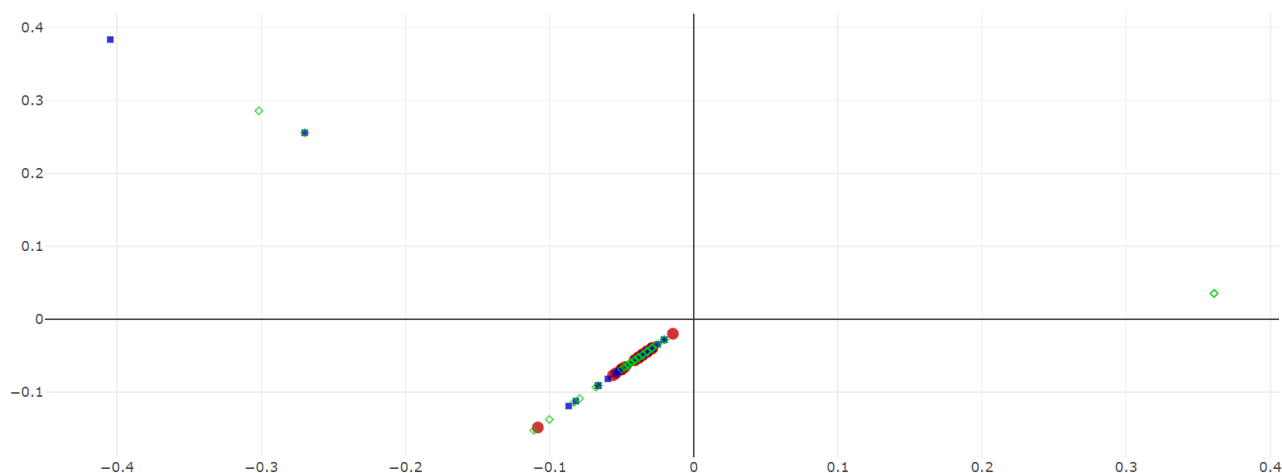


Рисунок 26 – двухмерный график эмбединга метаграфа алгоритмом Laplacian Eigenmaps

Как ясно из рисунка 26, результаты эмбединга получились более различимыми, из чего можно сразу сделать вывод, что данный алгоритм более чувствителен к различиям между близкими узлами метаграфа, а точнее трёхдольного графа, полученного из него.

Однако, сразу же видно, что данный алгоритм допускает ошибку и сливает воедино два узла e37 и v43, что представлено на рисунке 27 ниже.

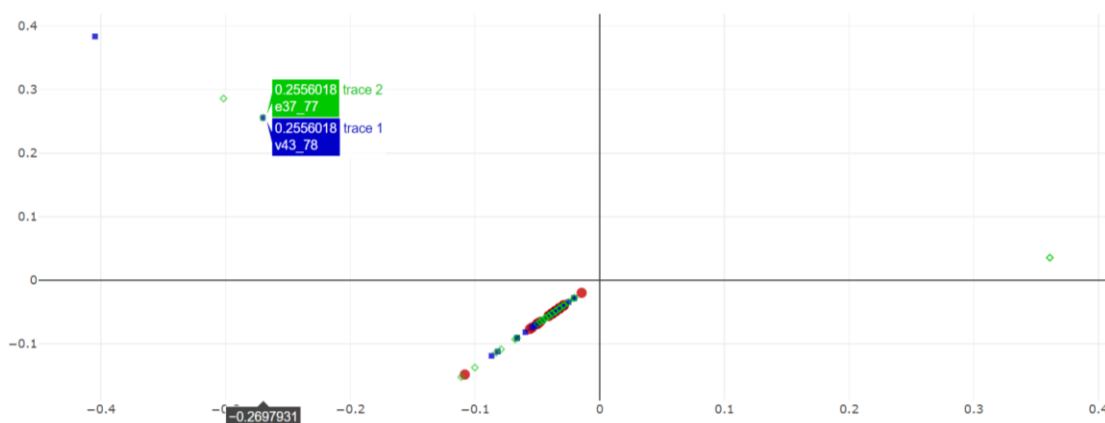


Рисунок 27 – Идентичность эмбединга узлов v43 и e37

Ниже на рисунке 28 ниже показан этот фрагмент в исходном метаграфе. Также на рисунке обозначены некоторые ключевые закономерности, которые делают похожими эти узлы.

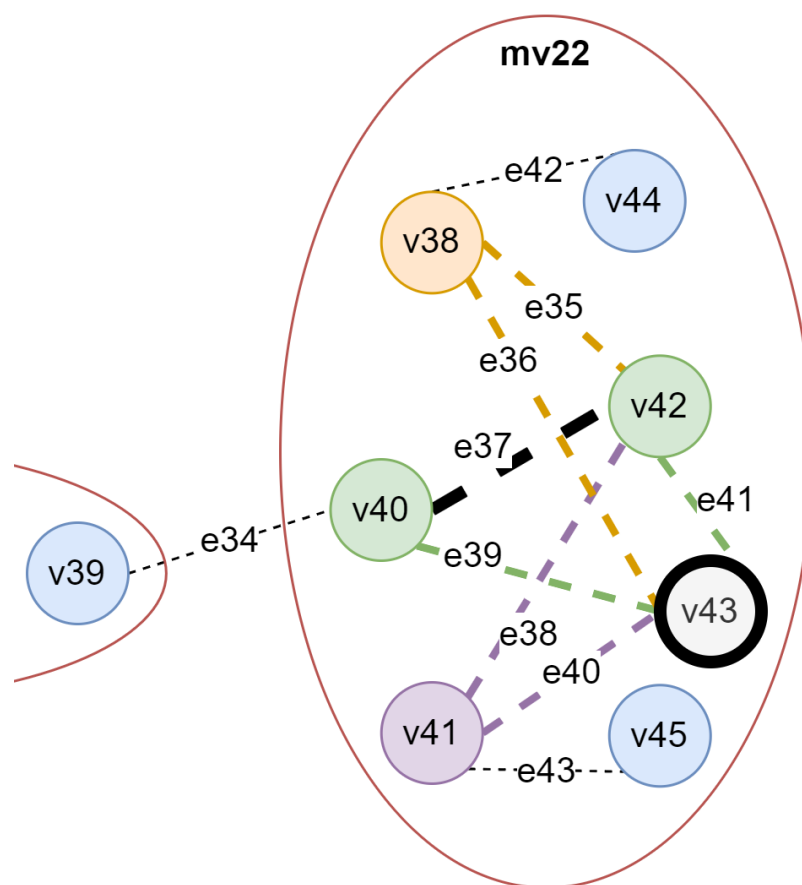


Рисунок 28 – Сходство ребра e37 и вершины v43 в исходном метаграфе

Очень сложно понять, чем схожи ребро e37 и вершина v43, кроме симметричной связи друг с другом соответственно через пары узлов e39 с v40 и v42 с e41, обозначенные на рисунке выше зелёным цветом. Кажущаяся симметрия относительно связей с другими вершинами показывает, что для связи с вершинами v38 и v41 ребру e37 необходимо пройти через соответствующие узлы вершин и рёбер, а вершине v43 только через рёбра (e36 и e40 соответственно).

Аналогично произошло и с другой парой вершины и ребра v20 и e29, не похожими друг на друга, с парой вершины и метавершины v6, mv10 и так далее. На рисунке 29 ниже показаны другие подобные, часто встречающиеся примеры одинаковых эмбедингов неизбежно разноклассовых узлов.

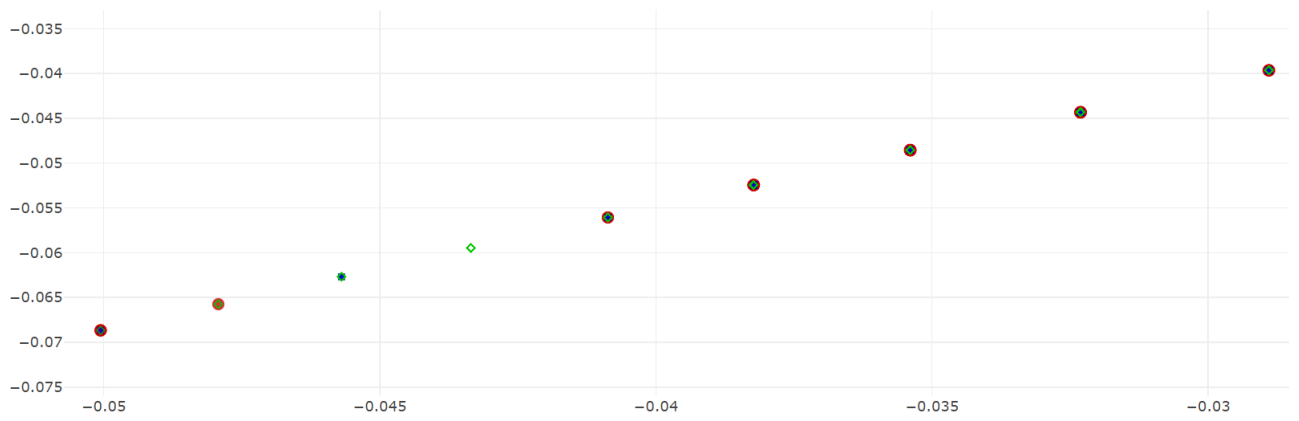


Рисунок 29 – Сливающиеся пары (тройки) разноклассовых узлов

На первый взгляд кажется, что эта ошибка фатальна, однако стоит заметить, что сливаются объекты разных классов, что лишь означает, что данный алгоритм специфическим образом учитывает, к какому классу принадлежит узел. Самое главное, что данный алгоритм не делает одинаковыми эмбединги узлов из разных классов. Это позволяет эффективно использовать алгоритм для получения входных данных моделей машинного обучения.

Ниже, на рисунке 30, приведён график трёхмерного эмбединга этим же алгоритмом этого же трёхдольного графа.

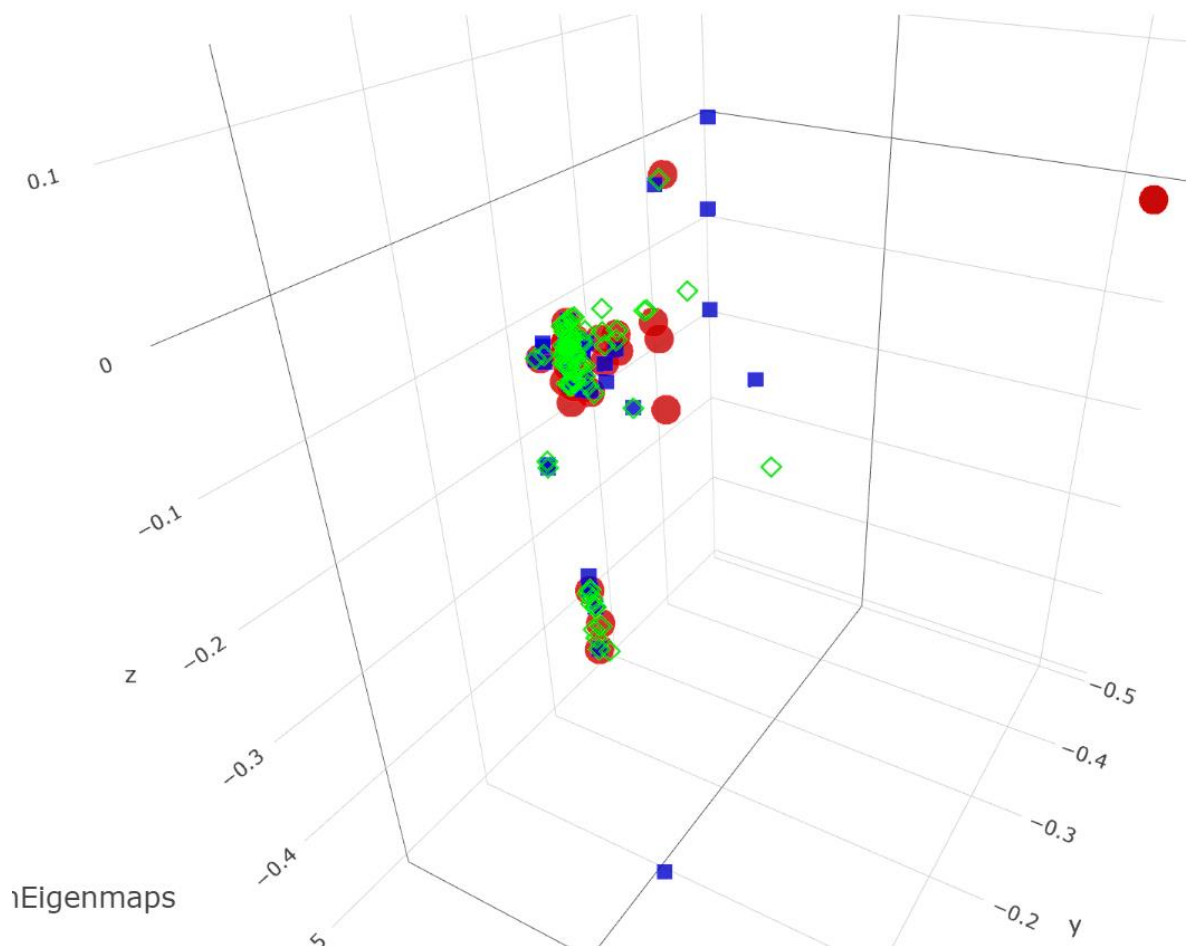


Рисунок 30 – Трёхмерный LE-эмбединг

Поскольку модель, производящая операцию вложения, была перезапущена, то эмбединг получился несколько другим, чем во время эксперимента с плоским вложением. Однако и здесь наблюдается сосредоточение большей части узлов на одной плоскости (предыдущий алгоритм собирал узлы на одной прямой), что отлично видно на рисунке 31 ниже.

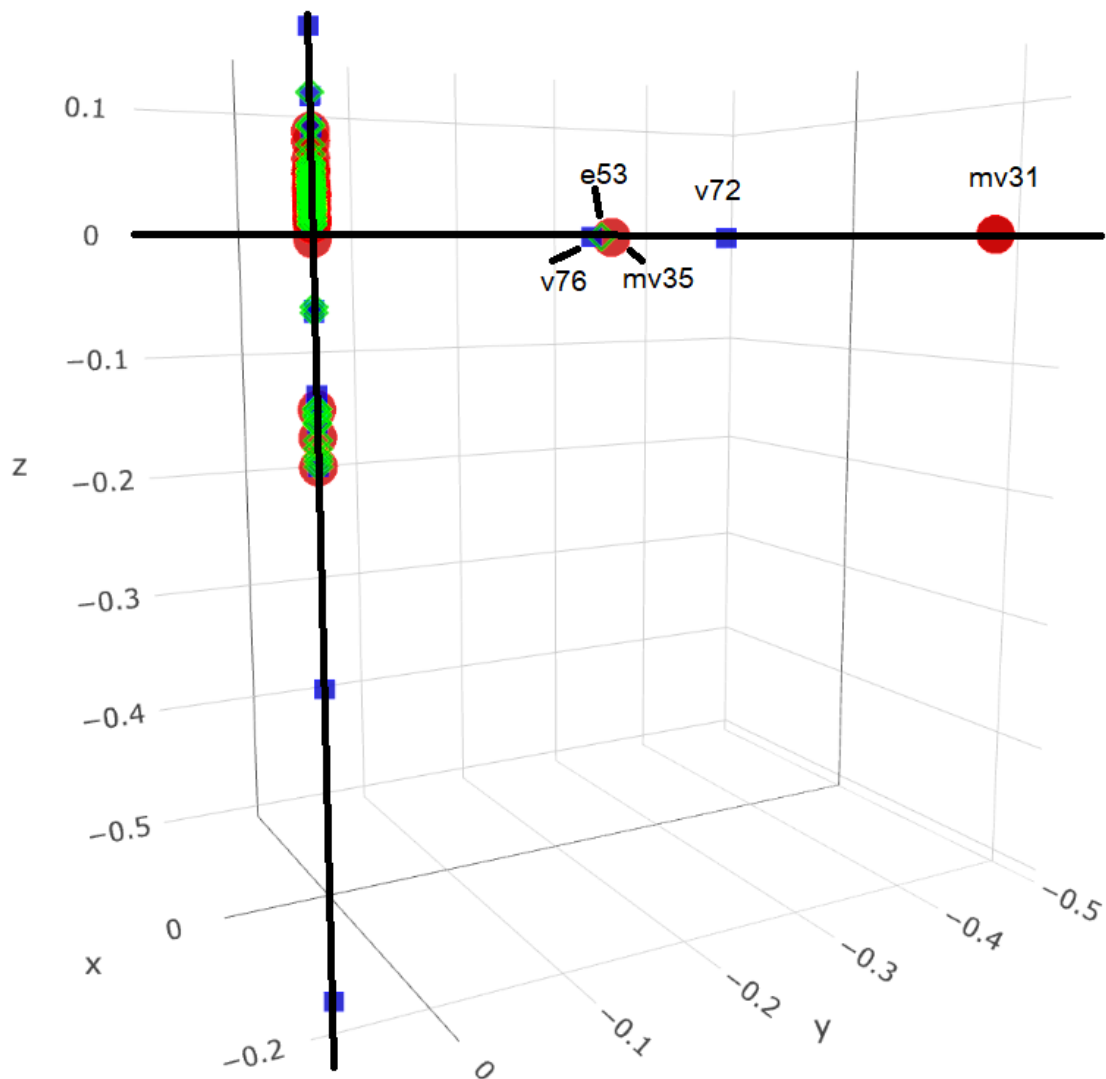


Рисунок 31 – Трёхмерный LE-эмбединг под другим углом

Отдельно стоит отметить, что узлы $v76$, $v72$, $mv31$, $mv35$ и $e53$ находятся не на одной плоскости, а лишь имеют значение координаты Y близкое к нулю. Если подробно рассматривать получившийся график, то становятся видны совпадения координат в векторном пространстве для узлов, принадлежащих разным классам, что не является помехой для алгоритмов машинного обучения.

Ниже, на рисунке 32, близко рассмотрен один из участков «плоского» скопления, показанного на рисунке 31 почти вертикальной чёрной линией. На этом участке очень близко друг к другу располагаются две метавершины: $mv8$ и $mv9$. Интересны они тем, что в исходном метаграфе они идентичны по сути состава вложенного фрагмента, но не по связям с другими узлами метаграфа.

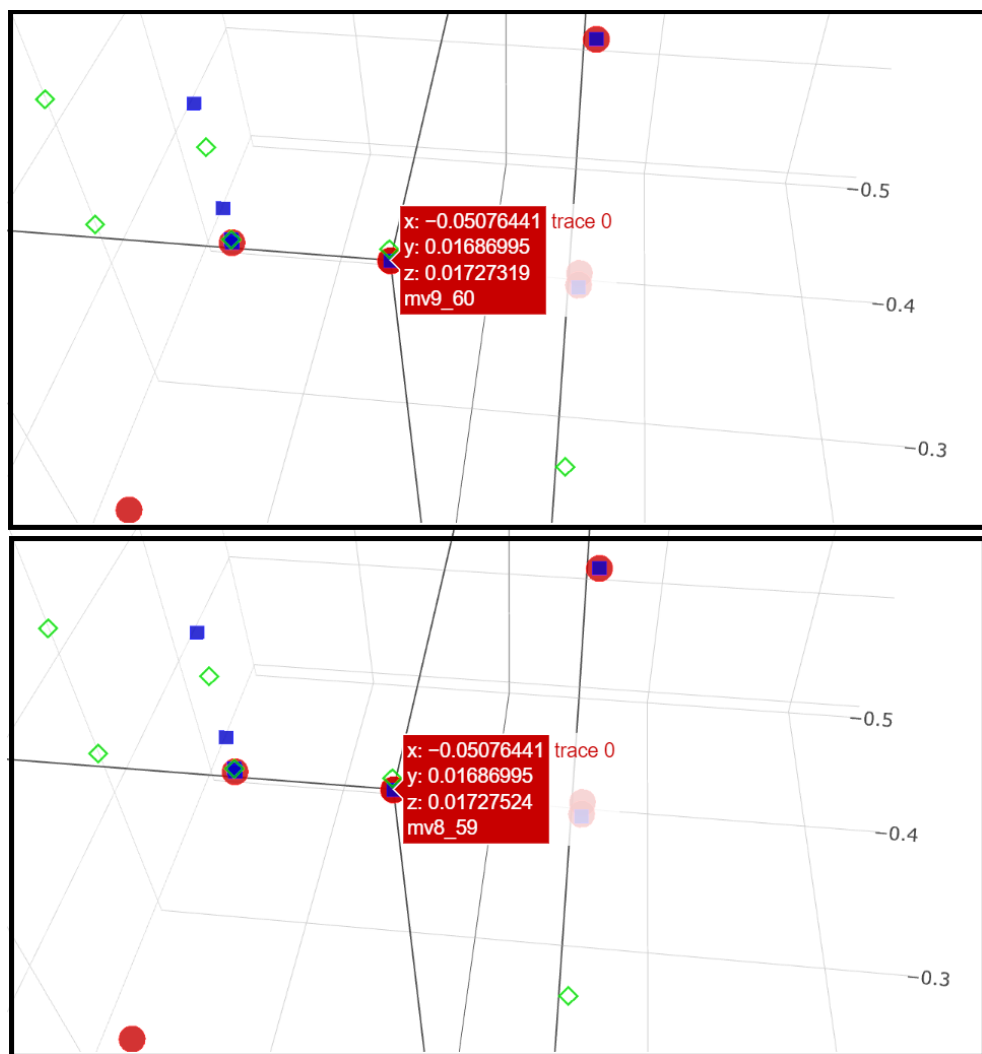


Рисунок 32 – Практически слившиеся метавершины mv8 и mv9

Ниже, на рисунке 33, приведён участок с этими метавершинами в исходном метаграфе.

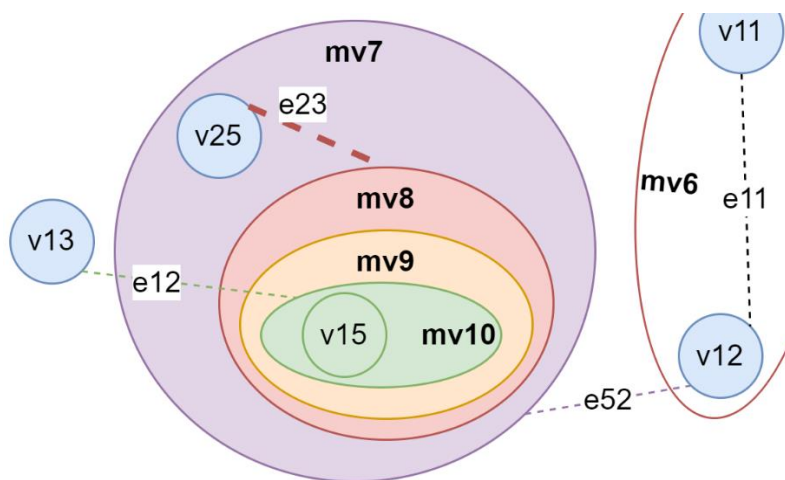


Рисунок 33 – Выделенные метавершины mv8 и mv9

Стоит отметить, что между метавершинами mv8 и mv9 достаточно много различий и главными из них являются вложенность mv9 в mv8 и непосредственная связь mv8 с ребром e23. Однако, стоит заметить, что близость второго порядка не учитывает непосредственную связь между узлами графа, следовательно, и алгоритм не учёл этой вложенности, как разницу между узлами. mv9 не связана с другими узлами посредством рёбер, в отличие от mv8 - это их явное отличие друг от друга. Также нельзя забывать, что трёхдольный граф не позволяет оценить, какая метавершина является вложенной, а какая родительской, следовательно, связи с вершинами v12 и v13 для mv8 (через mv7 и e52) и mv9 (через mv10 и e12) соответственно, являются по сути одинаковыми.

Ниже приведены некоторые основные статистические данные по выборке, которую представляют собой результаты эмбединга методом Laplacian Eigenmaps в двухмерном евклидовом пространстве.

Таблица 6 – Статистические данные

Статистика	X	Y	Z
Максимум	0.17502	0.06466	0.12427
Минимум	-0.234463	-0.48418	-0.4906
Среднее значение	-0.05667438	0.0078882	$-7.94877 \cdot 10^{-6}$
Медиана	-0.05268	0.017507	0.0219
Стандартное отклонение	0.0400427	0.06905	0.069505

График плотности распределения значений по осям x, y и z представлен на рисунках 34, 35 и 36.

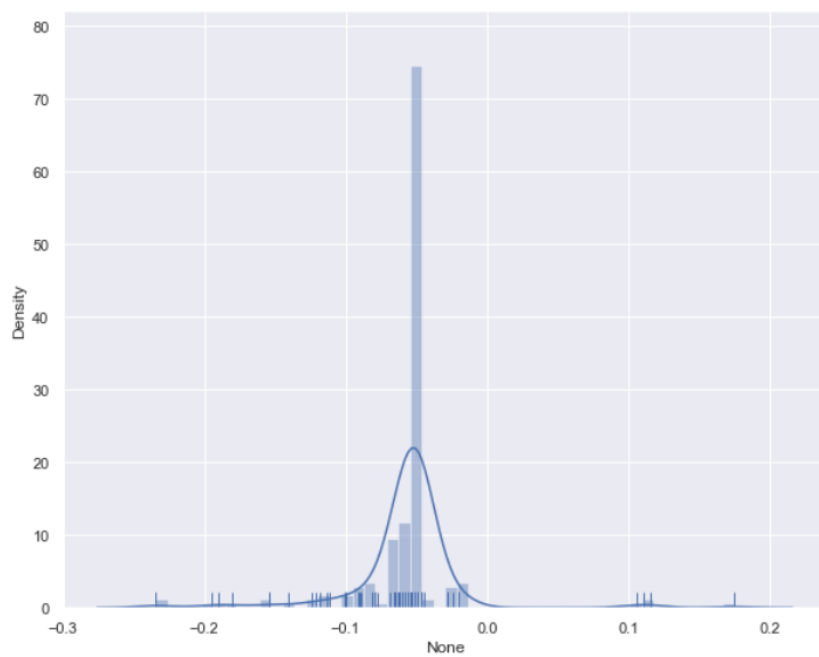


Рисунок 34 – Плотность распределения по оси X для LE

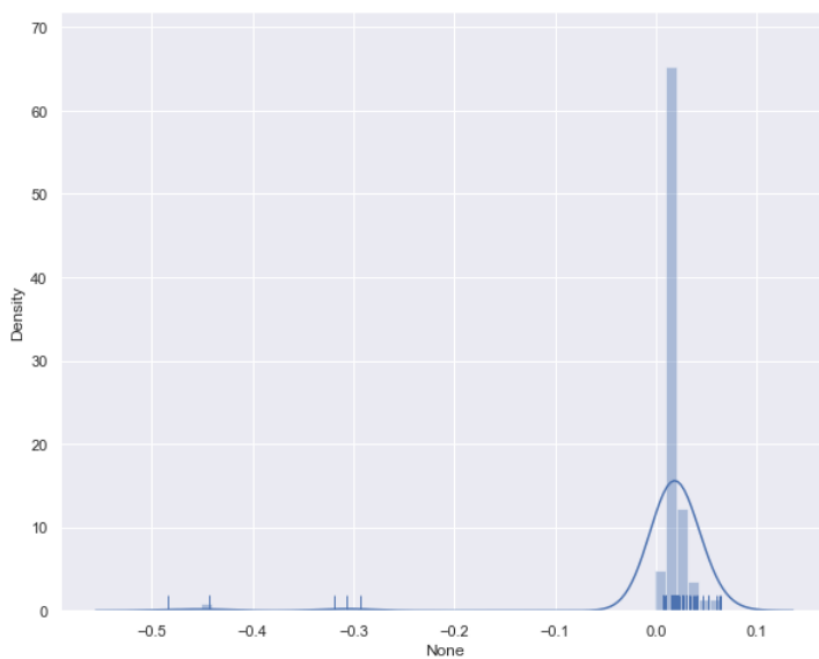


Рисунок 35 – Плотность распределения по оси Y для LE

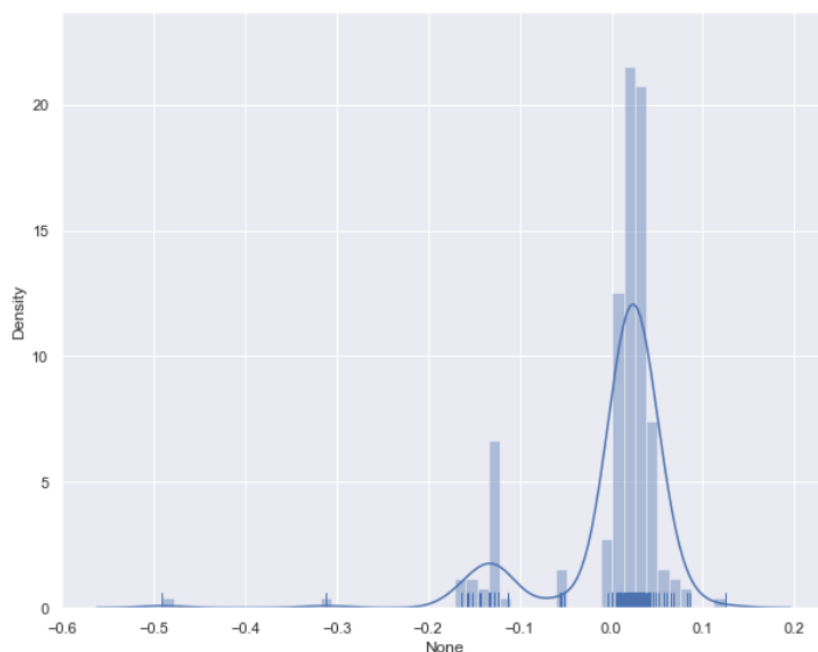


Рисунок 36 – Плотность распределения по оси Z для LE

Графики плотностей распределения показывают высокую плотность данных, более подходящую для успешного применения методов машинного обучения, чем та, что показана на соответствующих графиках по методу НОРЕ. Ниже, на рисунках 37.1-3, приведён график плотностей распределения для трёх пар плоскостей, что подтверждает достаточно высокий уровень плотности данных. Это необходимо учитывать при создании моделей машинного обучения, которые будут использовать этот эмбединг.

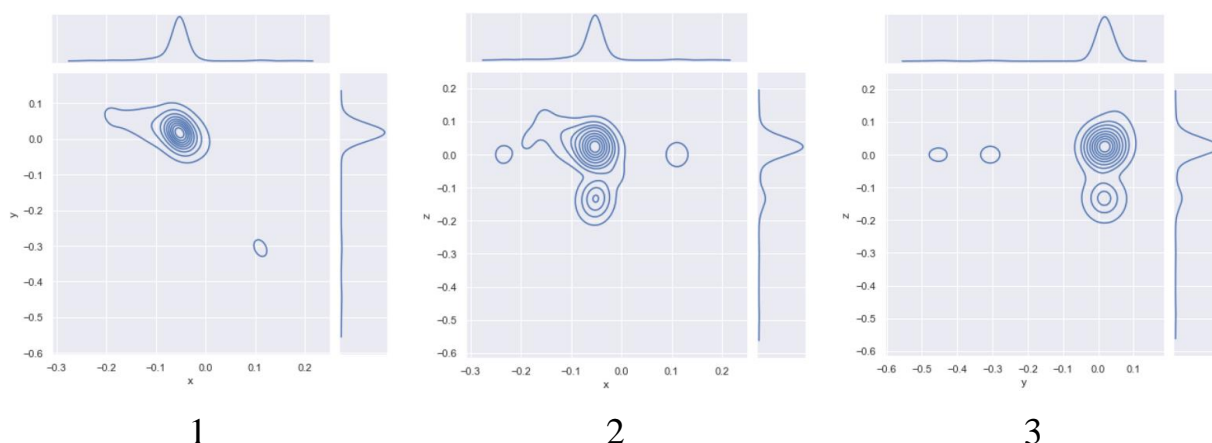


Рисунок 37 – Плотность распределения LE-эмбединга на плоскости

3.3.3. Эксперимент с алгоритмом Node2Vec

В результате работы алгоритма Node2Vec получен многообещающий трёхмерный график, представленный ниже на рисунке 38. По сравнению с предыдущими экспериментами бросается в глаза, насколько хорошо это векторное представление передаёт топологию исходного графа. И при этом, благодаря очевидно равномерному распределению узлов в пространстве, отсутствуют узлы с одинаковыми координатами.

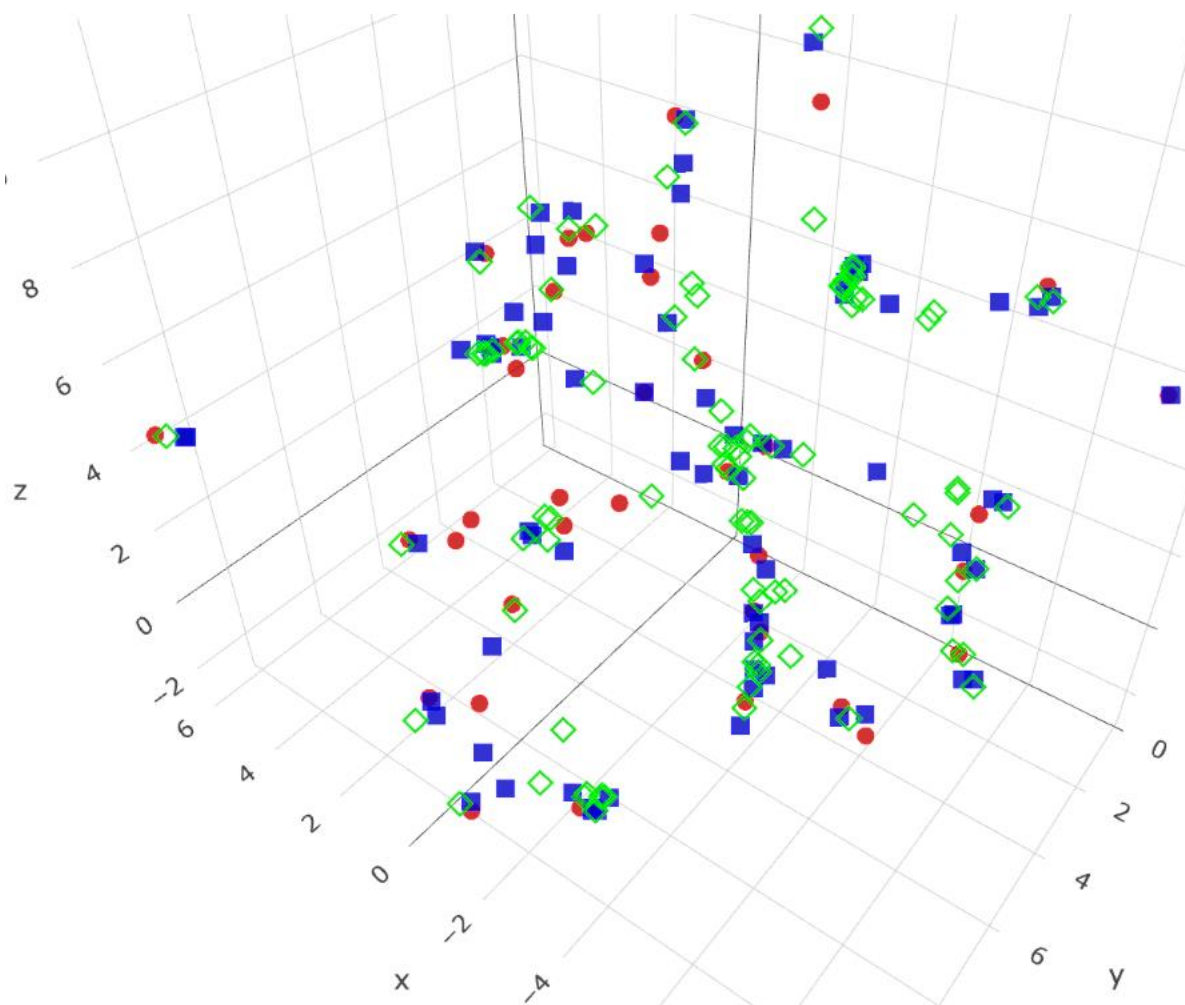


Рисунок 38 – трёхмерный график эмбединга алгоритмом Node2Vec

Ниже, на рисунках 39.1-2, представлен эмбединг фрагмента метаграфа и сам фрагмент в оригинальном метаграфе.

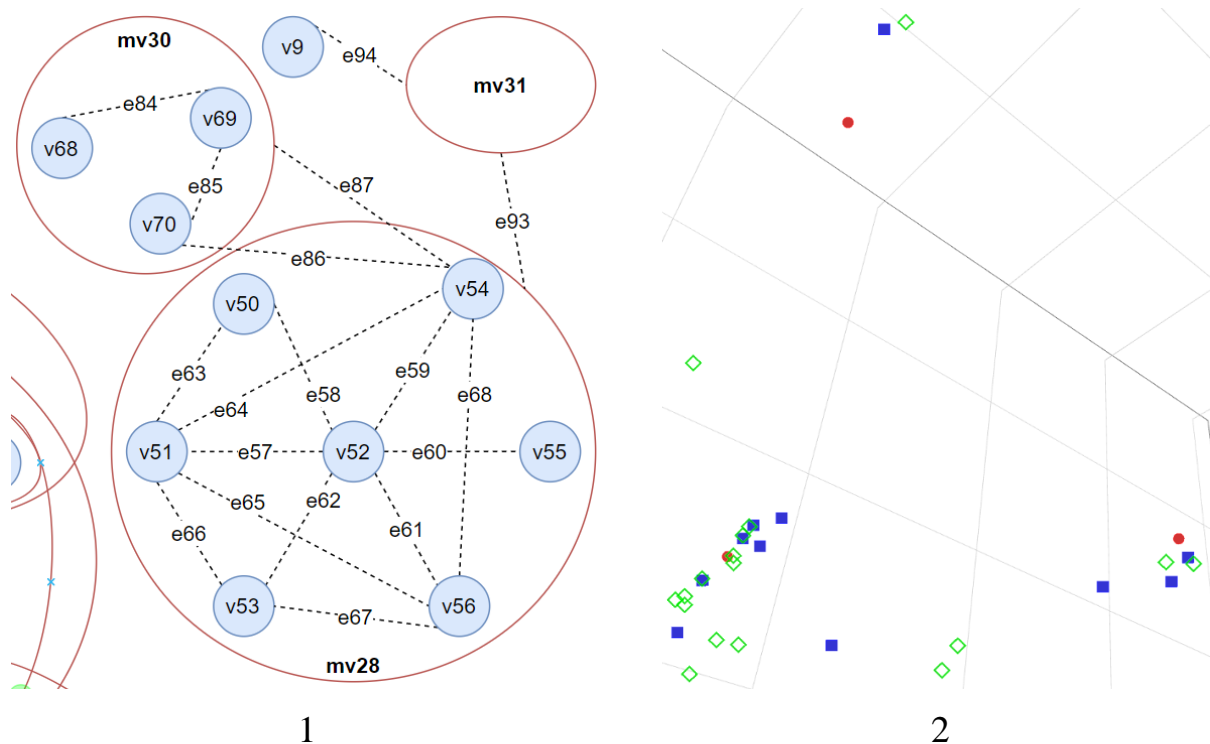


Рисунок 39 – Фрагмент метаграфа и его Node2Vec-эмбединг

В верхней части эмбединга находится вершина v_9 , рядом ребро e_{94} , далее вниз и налево (см. рисунок 39.2) mv_{31} и ребро e_{93} . Около метавершины mv_{30} (справа на рисунке 39.2) собраны соответствующие вложенные в неё узлы, около mv_{28} – аналогично. Между этими двумя скоплениями узлов видны две ребра e_{86} и e_{87} , которые связывают одно скопление с другим в оригинальном метаграфе. Такая же кучность сохраняется и для других узлов, благодаря чему положение узла в векторном пространстве уникально для каждого узла и легко угадывается при сравнении графика и метаграфа.

Ниже, на рисунках 40.1-2 показаны эмбединги метавершин mv_{32} и mv_{33} (и включенных в них рёбер и вершин) и фрагменты в оригинальном метаграфе. Стоит подчеркнуть, что данные узлы и в исходном метаграфе, и в получившемся векторном представлении удалены/отделены от остальных объектов, что видно и на рисунке 38, и на рисунке 17.

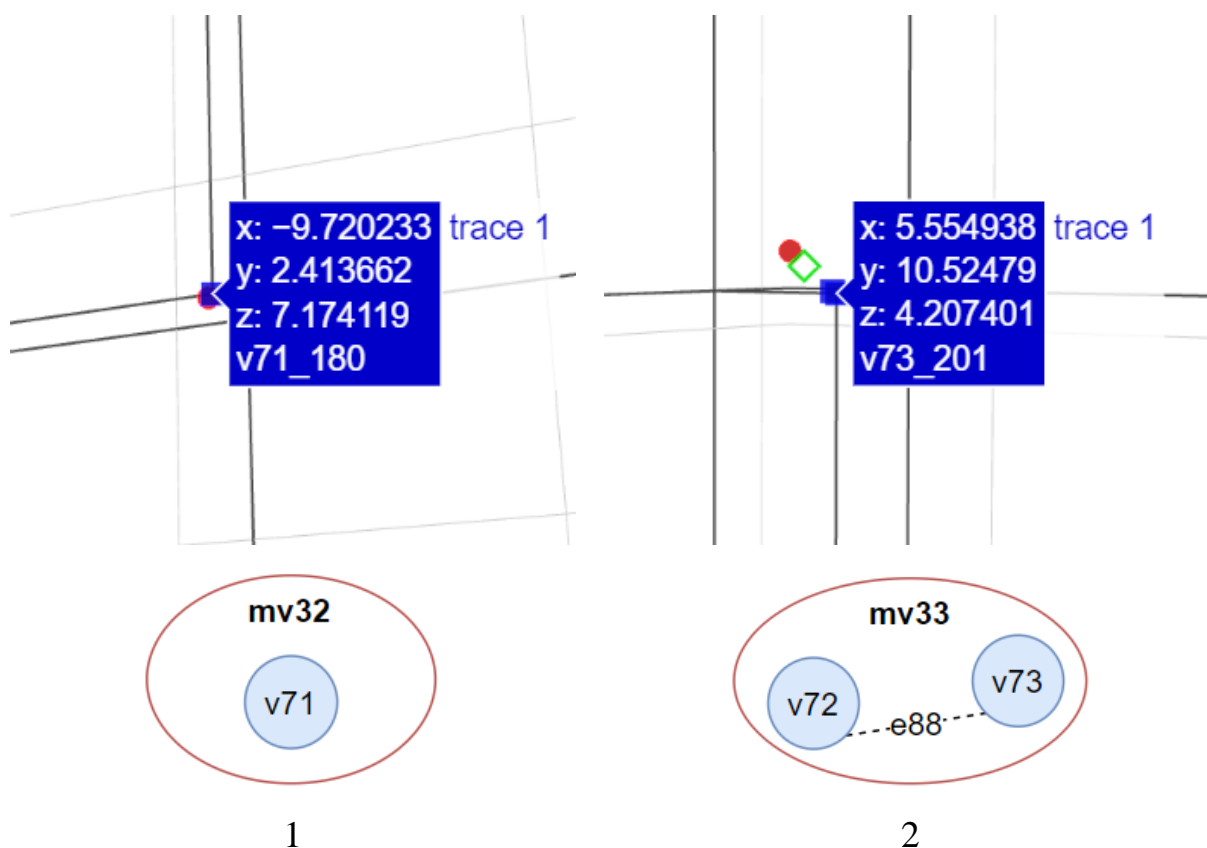


Рисунок 40 – Отдельные фрагменты трёхмерного Node2Vec-эмбединга

Ниже, на рисунках 41.X-Z, представлены графики плотности распределения по осям x, y и z.

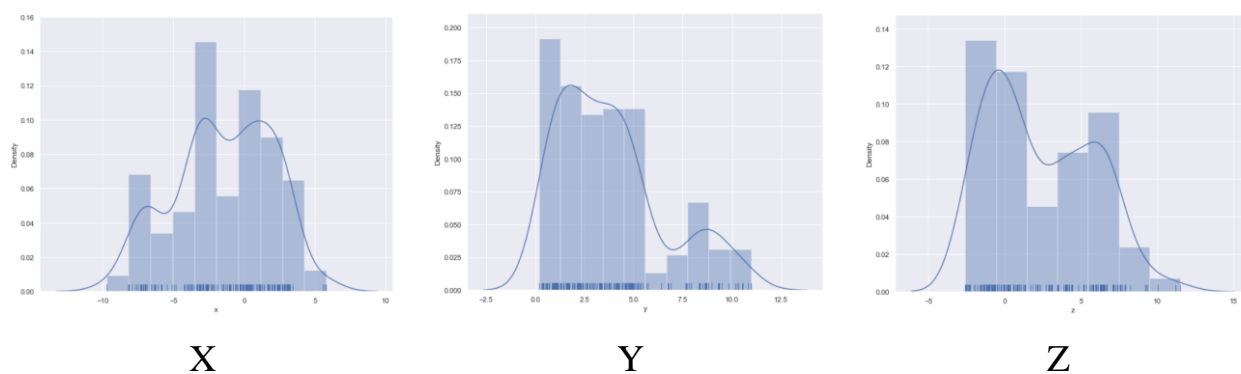


Рисунок 41 – Плотность распределения Node2Vec эмбединга по осям

Ниже приведены рисунки 42.XY, 42.XZ, 42.YZ, которые показывают плотности распределения по соответствующим плоскостям.

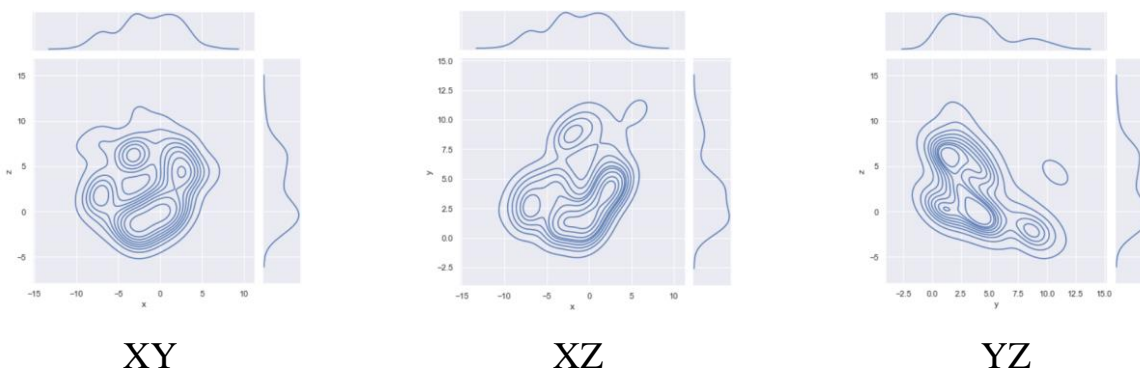


Рисунок 42 – Плотность распределения Node2Vec эмбединга по плоскостям

3.4. Обобщение результатов экспериментов

Все алгоритмы эмбединга дают результаты откровенно разного качества для данного примера.

Алгоритм NOPE сохраняет близость узлов метаграфа, похожих друг на друга. Этот метод направлен на сохранение близости высоких порядков между узлами графа, и он справляется со своей задачей: похожие узлы, расположенные рядом в исходном метаграфе остаются рядом и в векторном представлении. Однако, несмотря на возможность масштабировать полученные значения координат векторов, они получились очень близкими к нулю. Около 10% узлов отдалены от нуля на 0.35-1.1, остальные 90% скопились в области нуля плоскости XY, а разница между значениями их координат составляет от 0.05 до 10^{-9} , что категорически неудобно для большинства алгоритмов машинного обучения. Следовательно, полученное векторное представление будет нуждаться в предобработке, в масштабировании значений координат перед началом обучения. Из-за значений близости порядка 10^{-9} получается так, что многие узлы всё равно практически одинаковые, что повышает риск недообучения или переобучения моделей, использующих такое вложение.

Алгоритм Laplacian Eigenmaps лучше сохраняет структуру графа, хотя и не ориентируется на сохранение близости высокого порядка. Этот метод ориентируется на сохранение прямой связи (близости) между объектами, поэтому для метаграфа, узлы которого достаточно похожи между собой,

результаты получились довольно «плоскими». 95% узлов скопились практически на одной плоскости в трёхмерном пространстве, скопились достаточно близко друг к другу. При этом наблюдалось множество ситуаций (при встраивании и в двухмерное, и в трёхмерное пространство), когда узлы разного типа получали одинаковые координаты в векторном пространстве. Из-за тех 5% узлов, что не встраиваются в эту «плоскость», результаты могут оказаться не очень хорошими для методов машинного обучения. Также важным недостатком является одинаковость координат узлов разного класса. Несмотря на то, что есть возможность задать класс узла как признак в наборе данных для моделей машинного обучения, это может вызвать проблемы в процессе обучения.

Алгоритм Node2Vec показал себя наилучшим образом. координаты узлов в векторном пространстве чётко различаются между собой, не скапливаются в одной точке или на одной плоскости, а достаточно равномерно распределяются в векторном пространстве. При этом алгоритм не допускает излишнего масштабирования, слишком сильного отдаления, в частности, узлов-метавершин от остальной части метаграфа, если они не связаны с ним рёбрами или вложенностью. Наоборот, чем сильнее связь между объектами, тем узлы ближе друг к другу, а чем связь слабее - тем они дальше друг от друга. И речь идёт не только о соседних узлах, но и о узлах соединённых через множество разных связей с узлами разного класса. В частности, узлы, включённые в метавершину, располагаются рядом с ней в векторном пространстве. Таким образом сохраняется не только соседство по принципу соседства, но и по принципу вложенности. Благодаря этому алгоритм сохраняет близость высокого порядка, а также равномерную структуру метаграфа.

ЗАКЛЮЧЕНИЕ

В процессе исследования была поставлена задача найти пути к решению задачи вложения метаграфа в векторные пространства для дальнейшего успешного использования различными методами машинного обучения. Было выдвинуто предположение, что если преобразовать метаграф в изоморфный исходному метаграфу четырёхдольный граф, то получится выполнить операцию встраивания этого графа в какое-либо векторное пространство существующими алгоритмами эмбединга. Однако, в результате этого этапа исследования был сделан вывод, что трёхмерного пространства достаточно, чтобы передать классическую структуру метаграфа.

В процессе исследования было решено, что существующие алгоритмы эмбединга в неевклидовы пространства не оправдывают себя при вложении графов большой размерности, поэтому в основной части работы рассматривались только те алгоритмы, которые встраивают графы в евклидово векторное пространство.

Эмбединг производился с использованием лишь данных о связях между узлами трёхдольного графа (матрица смежности). Также связям были назначены веса с учётом класса того или иного узла и связи, которая идёт от него или к нему.

Из существующего множества алгоритмов эмбединга плоских графов в евклидово векторное пространство были выбраны три алгоритма, действующих по-разному принципам и делающих упор на сохранение разных свойств графа: NOPE, Laplacian Eigenmaps и Node2Vec. Лучше всего структуру метаграфа и близость между узлами графа сохранил алгоритм Node2Vec, основанный на методах глубокого обучения.

Данное направление исследований метаграфовой модели данных требует дальнейших исследований. Для создания полноценного и проверенного метода эмбединга метаграфов, необходимо применение и других алгоритмов эмбединга, использование множества самых разных примеров метаграфов,

разработка прямого алгоритма эмбединга метаграфов или хотя бы иная манипуляция с весами рёбер трёхдольного графа. Также в будущем возможно усовершенствование самой метаграфовой модели, уточнение модели метаребра. Однако всё это, как и поиски наиболее оптимальных путей для решения тех или иных задач со сложными сетями различных предметных областей, уже темы для других будущих исследований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wang X., Cui P., Wang J., Pei J., Zhu W., and Yang S., “Community preserving network embedding,” in AAAI, 2017, pp. 203–209.
2. Nie F., Zhu W., and Li X., “Unsupervised large graph embedding,” in AAAI, 2017, pp. 2422–2428.
3. Zhou C., Liu Y., Liu X., Liu Z., and Gao J., “Scalable graph embedding for asymmetric proximity,” in AAAI, 2017, pp. 2942– 2948.
4. Wei X., Xu L., Cao B., and Yu P.S., “Cross view link prediction by learning noise-resilient representation consensus,” in WWW, 2017, pp. 1611–1619.
5. Basu A., Blanning R. Metagraphs and their applications. Springer, 2007. 174 p.
6. Черненький В.М., Терехов В.И., Гапанюк Ю.Е. Структура гибридной интеллектуальной информационной системы на основе метаграфов. Нейрокомпьютеры: разработка, применение. 2016. Выпуск №9. С. 3-14.
7. Черненький В.М., Гапанюк Ю.Е., Ревунков Г.И., Терехов В.И., Каганов Ю.Т. Метаграфовый подход для описания гибридных интеллектуальных информационных систем. Прикладная информатика. 2017. № 3 (69). Том 12. С. 57–79.
8. Черненький В.М., Терехов В.И., Гапанюк Ю.Е. Представление сложных сетей на основе метаграфов // Нейроинформатика-2016. XVIII Всероссийская научно-техническая конференция. Сборник научных трудов. Ч. 1. М.: НИЯУ МИФИ, 2016. С. 173-178
9. Самохвалов Э.Н., Ревунков Г.И., Гапанюк Ю.Е. Использование метаграфов для описания семантики и прагматики информационных систем. Вестник МГТУ им. Н.Э. Баумана. Сер. «Приборостроение». 2015. Выпуск №1. С. 83-99.
10. Chernenkiy V.M., Terekhov V.I., Gapanyuk Yu.E. Predstavleniye slozhnikh setey na osnove metagrafov [Metagraph representation of complex

networks]. Trudi XVIII vserossiyskoy konferencii “Neuroinformatics-2016” [Proc. XVIII all-russian conference “Neuroinformatics-2016”], Moscow, 2016, pp. 173-178.

11. Samokhvalov E.N., Revunkov G.I. Gapanyuk Yu.E. Ispolzovaniye metagrazfov dlya opisaniya semantiki i pragmatiki informatsionnykh sistem [Metagraphs for information systems semantics and pragmatics definition]. Vestnik MGTU im. N.E. Baumana, seriya “Priborostroeniye” [Herald of the Bauman Moscow State Technical University, “Instrument Engineering”], 2015, no. 1, pp. 83-99.

12. Voloshin Vitaly I. Introduction to Graph and Hypergraph Theory. Nova Science Publishers, Inc., 2009, 287 p.

13. Попков В.К. Математические модели связности. Новосибирск: ИВМиМГ СО РАН, 2006. – 490 с.

14. Гапанюк Ю.Е., Ревунков Г.И., Федоренко Ю.С. Предикатное описание метаграфовой модели данных. Информационно-измерительные и управляющие системы. 2016. Выпуск № 12. С. 122-131.

15. Wang Q., Mao Z., Wang B., Guo L. Knowledge Graph Embedding: A Survey of Approaches and Applications. IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 12, 2017, pp. 2724-2743.

16. Hongyun C., Zheng W.V., Chang K.C.-C. (2017). A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. IEEE Transactions on Knowledge and Data Engineering. 10.1109/TKDE.2018.2807452.

17. Gonzalez J. E., Xin R. S., Dave A., Crankshaw D., Franklin M. J., and Stoica I., “Graphx: Graph processing in a distributed dataflow framework,” in OSDI, 2014, pp. 599–613.

18. Low Y., Bickson D., Gonzalez J., Guestrin C., Kyrola A., and Hellerstein J. M., “Distributed graphlab: A framework for machine learning and data mining in the cloud,” Proc. VLDB Endow., vol. 5, no. 8, 2012, pp. 716–727.

19. Kumar P. and Huang H. H., “G-store: High-performance graph store for trillion-edge processing,” in SC, 2016, pp. 1–12.

20. Fang H., Wu F., Zhao Z., Duan X., Zhuang Y., and Ester M., “Community-based question answering via heterogeneous social network learning,” in AAAI, 2016, pp. 122–128.
21. Chang S., Han W., Tang J., Qi G.-J., Aggarwal C.C., and Huang T.S., “Heterogeneous network embedding via deep architectures,” in KDD, 2015, pp. 119–128.
22. Zhang H., Shang X., Luan H., Wang M., and Chua T., “Learning from collective intelligence: Feature learning using social images and tags,” TOMCCAP, vol. 13, no. 1, 2016, pp. 1–23.
23. Geng X., Zhang H., Bian J., and Chua T., “Learning image and user features for recommendation in social networks,” in ICCV, 2015, pp. 4274–4282.
24. Wu F., Lu X., Song J., Yan S., Zhang Z.M., Rui Y., and Zhuang Y., “Learning of multimodal representations with random walks on the click graph,” IEEE Trans. Image Processing, vol. 25, no. 2, 2016, pp. 630–642.
25. Bollacker K., Evans C., Paritosh P., Sturge T., and Taylor J., “Freebase: A collaboratively created graph database for structuring human knowledge,” in SIGMOD, 2008, pp. 1247–1250.
26. Nickel M., Murphy K., Tresp V., Gabrilovich E., “A Review of Relational Machine Learning for knowledge Graphs” [Электронный ресурс]. 2015. Дата обновления: 02.3.2015. URL: <https://arxiv.org/abs/1503.00759> (дата обращения: 1.06.2022).
27. Wu F., Song J., Yang Y., Li X., Zhang Z.M., and Zhuang Y., “Structured embedding via pairwise relations and long-range interactions in knowledge base,” in AAAI, 2015, pp. 1663–1670.
28. Shi B. and Weninger T., “Proje: Embedding projection for knowledge graph completion,” in AAAI, 2017, pp. 1236–1242.
29. Ashby F.G. and Ennis D.M., Similarity measures. Scholarpedia, 2(12):4116, 2007.
30. Goyal P. and Ferrara E., “Graph embedding techniques, applications, and performance: A survey,” CoRR, vol. abs/1705.02801, 2017.

31. Cai D., He X., and Han J.. “Spectral regression: a unified subspace learning framework for content-based image retrieval” in MM, 2007, pp. 403–412.
32. Guattery S. and Miller G.L., Graph embeddings and laplacian eigenvalues. SIAM Journal on Matrix Analysis and Applications, 21(3), 2000, pp. 703–723.
33. Chung F.R.K., Spectral Graph Theory, Regional Conference Series in Mathematics, vol.92, AMS, 1997.
34. He X., Yan S., Hu Y., Niyogi P., and Zhang H.-J., Face recognition using laplacian faces. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(3), 2005, pp. 328–340.
35. Belkin, M., & Niyogi, P., Laplacian eigenmaps and spectral techniques for embedding and clustering. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), Advances in neural information processing systems, 14. Cambridge, MA: MIT Press, 2002.
36. Ou M., Cui P., Pei J., Zhang Z., Zhu W., KDD'16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1105–1114.
37. Mikolov T., Chen K., Corrado G., and Dean J., Efficient estimation of word representations in vector space. ICLR Workshop, 2013.
38. Perozzi B., Al-Rfou R., and Skiena S., Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, ACM, New York, USA, 2014, pp. 701–710.
39. Grover A. and Leskovec J., node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
40. Hopcroft J. and Kannan R., Foundations of data science. 2014.
41. Guattery S. and Miller G.L., Graph embeddings and laplacian eigenvalues. SIAM Journal on Matrix Analysis and Applications, 21(3), 2000, pp. 703–723.

42. Chung F.R.K., Spectral Graph Theory, volume 92 of Regional Conference Series in Mathematics. AMS, 1997.
43. Hofmann T. and Buhmann J.M., “Multidimensional scaling and data clustering,” in NIPS, 1994, pp. 459–466.
44. Balasubramanian M. and Schwartz E.L., “The isomap algorithm and topological stability,” Science, vol. 295, no. 5552, 2002, pp. 7–7.
45. Anderson Jr. W.N. and Morley T. D., “Eigenvalues of the laplacian of a graph,” Linear and Multilinear Algebra, vol. 18, no. 2, 1985, pp. 141–145.
46. Yang Y., Nie F., Xiang S., Zhuang Y., and Wang W., “Local and global regressive mapping for manifold learning with out-of- sample extrapolation,” in AAAI, 2010.
47. Jiang R., Fu W., Wen L., Hao S., and Hong R., “Dimensionality reduction on anchorgraph with an efficient locality preserving projection,” Neurocomputing, vol. 187, 2016, pp. 109–118.
48. Lin Y.-Y., Liu T.-L., and Chen H.-T., “Semantic manifold learning for image retrieval,” in MM, 2005, pp. 249–258.
49. Chen M., Tsang I. W., Tan M., and Jen C. T., “A unified feature selection framework for graph embedding on high dimensional data,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 6, 2015, pp. 1465–1477.
50. Yan S., Xu D., Zhang B., Zhang H., Yang Q., and Lin S., “Graph embedding and extensions: A general framework for dimensionality reduction,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 1, 2007, pp. 40–51.
51. Yang Y., Nie F., Xiang S., Zhuang Y., and Wang W., “Local and global regressive mapping for manifold learning without-of sample extrapolation,” in AAAI, 2010.
52. Xiang S., Nie F., Zhang C., and Zhang C., “Nonlinear dimensionality reduction with local spline embedding,” IEEE Trans. Knowl. Data Eng., vol. 21, no. 9, 2009, pp. 1285–1298.
53. Suzuki A., Enokida Y., and Yamanishi K., Riemannian TransE: Multi-relational Graph Embedding in Non-Euclidean Space, ICLR, 2019.

54. Balažević I., Allen C., Hospedales T., Multi-relational Poincaré Graph Embeddings, Advances in Neural Information Processing Systems 32 (NeurIPS 32), 2019.

55. Гапанюк Ю.Е., Ревунков Г.И., Злобина С.В., Кадиев З.Д. Обзор подходов к векторному представлению графов знаний и метаграфов. Динамика сложных систем - XXI век. 2019. Т. 13. № 2. С. 67-74.

56. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. Силинкина А.А., научный редактор Борисов А.В. М.: ДМК Пресс, 2018. – 232 с.: ил. ISBN 978-5-97060-581-3.

ПРИЛОЖЕНИЕ А

ГРАФИЧЕСКАЯ ЧАСТЬ

В графическую часть дипломного проекта входят:

- 1) Вершины тестового метаграфа (Таблица А.1);
- 2) Рёбра тестового метаграфа (Таблица А.2);
- 3) Метавершины тестового метаграфа (Таблица А.3).

Таблица А.1 – Вершины тестового метаграфа

№ узла	Название	Список родительских метавершин
1	v1	mv1
2	v2	mv1
3	v3	mv1
4	v4	mv1
5	v5	mv1
6	v6	mv2
7	v7	mv2, mv3
8	v8	mv2
9	v9	-
10	v10	mv3, mv5
11	v11	mv4, mv5, mv6
12	v12	mv6, mv26
13	v13	mv4
14	v14	mv4
15	v15	mv10
16	v16	mv11
17	v17	mv11, mv12
18	v18	mv11
19	v19	mv11, mv12
20	v20	mv12
21	v21	mv12
22	v22	mv13
23	v23	mv13, mv14
24	v24	mv13
25	v25	mv7
26	v26	mv14

№ узла	Название	Список родительских метавершин
27	v27	mv14
28	v28	mv14, mv15
29	v29	mv15, mv16
30	v30	mv15, mv17
31	v31	mv15, mv18, mv36, mv37
32	v32	mv16
33	v33	mv16, mv19
34	v34	mv17
35	v35	mv17, mv20
36	v36	mv18
37	v37	mv18, mv21
38	v38	mv19, mv22
39	v39	mv20
40	v40	mv20, mv22
41	v41	mv21, mv22
42	v42	mv22
43	v43	mv22
44	v44	mv22
45	v45	mv22
46	v46	mv23, mv26
47	v47	mv23, mv38
48	v48	mv23, mv24
49	v49	mv23, mv25, mv35
50	v50	mv28
51	v51	mv28
52	v52	mv28
53	v53	mv28

№ узла	Название	Список родительских метавершин
54	v54	mv28
55	v55	mv28
56	v56	mv28
57	v57	mv27
58	v58	mv27
59	v59	mv27
60	v60	mv27
61	v61	mv27
62	v62	mv24
63	v63	mv25
64	v64	mv38
65	v65	mv29
66	v66	mv29
67	v67	mv29
68	v68	mv30
69	v69	mv30
70	v70	mv30
71	v71	mv32
72	v72	mv33
73	v73	mv33
74	v74	mv3, mv34
75	v75	mv34
76	v76	mv35

Таблица А.2 – Рёбра тестового метаграфа

№ узла	Название	Стартовая вершина (метавершина)	Конечная вершина (метавершина)	Список родительских метавершин
77	e1	v1	v2	mv1
78	e2	v2	v3	mv1
79	e3	v3	v4	mv1
80	e4	v4	v5	mv1
81	e5	v5	v1	mv1
82	e6	v5	v2	mv1
83	e7	v6	v7	mv2
84	e8	mv1	mv2	-
85	e9	v3	v8	-
86	e10	v10	v11	mv5
87	e11	v11	v12	mv6
88	e12	v13	mv10	mv4
89	e13	v13	v14	mv4
90	e14	v14	v11	mv4
91	e15	v17	v19	mv11, mv12
92	e16	v18	v19	mv11
93	e17	v21	v20	mv12
94	e18	v24	v23	mv13
95	e19	v23	v22	mv13
96	e20	v20	mv13	mv12
97	e21	v24	v22	mv13
98	e22	v21	v23	mv12
99	e23	v25	mv8	mv7
100	e24	v23	v26	mv14

№ узла	Название	Стартовая вершина (метавершина)	Конечная вершина (метавершина)	Список родительских метавершин
101	e25	v26	v27	mv14
102	e26	v27	v28	mv14
103	e27	v31	v30	mv15
104	e28	v30	v29	mv15
105	e29	v29	v28	mv15
106	e30	v31	v36	mv18
107	e31	v36	v37	mv18
108	e32	v34	v35	mv17
109	e33	v37	v41	mv21
110	e34	v39	v40	mv20
111	e35	v38	v42	mv22
112	e36	v38	v43	mv22
113	e37	v40	v42	mv22
114	e38	v41	v42	mv22
115	e39	v40	v43	mv22
116	e40	v41	v43	mv22
117	e41	v42	v43	mv22
118	e42	v38	v44	mv22
119	e43	v41	v45	mv22
120	e44	v5	v16	-
121	e45	mv4	mv11	-
122	e46	v28	v15	-
123	e47	v46	v47	mv23
124	e48	v46	v48	mv23
125	e49	v47	v49	mv23

№ узла	Название	Стартовая вершина (метавершина)	Конечная вершина (метавершина)	Список родительских метавершин
126	e50	v48	v49	mv23
127	e51	v17	v20	mv12
128	e52	mv6	mv7	mv4
129	e53	mv16	mv22	-
130	e54	mv22	mv23	-
131	e55	v12	v46	mv26
132	e56	v28	v46	-
133	e57	v52	v51	mv28
134	e58	v52	v50	mv28
135	e59	v52	v54	mv28
136	e60	v52	v55	mv28
137	e61	v52	v56	mv28
138	e62	v52	v57	mv28
139	e63	v51	v50	mv28
140	e64	v51	v54	mv28
141	e65	v51	v56	mv28
142	e66	v51	v53	mv28
143	e67	v53	v56	mv28
144	e68	v54	v56	mv28
145	e69	mv38	v58	mv27
146	e70	v58	v59	mv27
147	e71	v59	mv24	mv27
148	e72	v60	mv23	mv27
149	e73	v57	v64	mv27
150	e74	mv26	mv38	mv27

№ узла	Название	Стартовая вершина (метавершина)	Конечная вершина (метавершина)	Список родительских метавершин
151	e75	v62	v48	mv24
152	e76	v58	v63	mv27
153	e77	v49	v63	mv25
154	e78	v61	v58	mv27
155	e79	v61	v64	mv27
156	e80	mv27	mv4	-
157	e81	v67	mv15	-
158	e82	v65	v66	mv29
159	e83	v66	mv15	-
160	e84	v68	v69	mv30
161	e85	v69	v70	mv30
162	e86	mv30	v54	-
163	e87	v70	v54	-
164	e88	v72	v73	mv33
165	e89	v74	v75	mv34
166	e90	mv29	mv36	-
167	e91	v16	v17	mv12
168	e92	v18	v16	mv11
169	e93	mv31	mv28	-

Таблица А.3 – Метавершины тестового метаграфа

№ узла	Название	Список вложенных узлов	Список родительских метавершин
169	mv1	v1, v2, v3, v4, v5, e1, e2, e3, e4, e5, e6	-
170	mv2	v6, v7, v8, e7	-
171	mv3	v7, v10, v74	-
172	mv4	mv5, mv6, mv7, v11, v13, v14, e12, e13, e14, e52	-
173	mv5	v11, v10, e10	mv4, mv34
174	mv6	v11, v12, e11	mv4
175	mv7	mv8, v25, e23	mv4
176	mv8	mv9	mv7
177	mv9	mv10	mv8
178	mv10	v15	mv9
179	mv11	v16, v17, v18, v19, e15, e16, e91, e92	-
180	mv12	v17, v19, e15, e51, v20, v21, e20, e22, mv13, e17	-
181	mv13	v22, v23, v24, e18, e19, e21	mv12
182	mv14	v23, v26, v27, v28, e24, e25, e26	-
183	mv15	v28, v29, v30, v31, e27, e28, e29	-
184	mv16	v29, v32, v33	-
185	mv17	v30, v34, v35, e32	-
186	mv18	v31, v36, v37, e30, e31	-
187	mv19	v33, v38	-
188	mv20	v35, v39, v40, e34	-
189	mv21	v37, v41, e33	-

№ узла	Название	Список вложенных узлов	Список родительских метавершин
190	mv22	v38, v40, v41, v42, v43, v44, v45, e35, e36, e37, e38, e39, e40, e41, e42, e43	-
191	mv23	v46, v47, v48, v49, e47, e48, e49, e50	mv27
192	mv24	v48, v62, e75	mv27
193	mv25	v49, v63, e77	mv27, mv35
194	mv26	v12, v46, e55	-
195	mv27	v57, v58, v59, v60, v61, mv24, mv25, mv38, mv23, e69, e70, e71, e72, e73, e74, e76, e78, e79	-
196	mv28	v50, v51, v52, v53, v54, v55, v56, e57, e58, e59, e60, e61, e62, e63, e64, e65, e66, e67, e68	-
197	mv29	v65, v66, v67, e82	-
198	mv30	v68, v69, v70, e84, e85	-
199	mv31	-	-
200	mv32	v71	-
201	mv33	v72, v73, e88	-
202	mv34	mv5, v74, v75, e89	-
203	mv35	v49, mv25, v76	-
204	mv36	v31	-
205	mv37	v31	-
206	mv38	v47, v64	mv27

ПРИЛОЖЕНИЕ Б

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

Утверждаю: Гапанюк Ю.Е. _____

"15" ноября 2021 г.

Исследование векторного представления метаграфов

Техническое задание
(вид документа)

писчая бумага
(вид носителя)

5
(количество листов)

ИСПОЛНИТЕЛЬ:

Фадеев А.А. _____

"15" ноября 2021 г.

Москва - 2021

1. Наименование

Исследование векторного представления метаграфов

2. Основание для разработки

Основанием для разработки является задание на выпускную работу, подписанное руководителем выпускной работы и утверждённое заведующим кафедрой. Задание утверждено кафедрой ИУ5 МГТУ им. Н.Э. Баумана.

3. Исполнитель

Студент второго курса группы ИУ5-31М Фадеев А.А.

4. Назначение и цель работы

Исследовать возможности и определить подходы для решения задачи эмбединга метаграфа, используя существующие алгоритмы эмбединга плоских графов.

5. Содержание работы

5.1. Задачи

В процессе выполнения работы следующие задачи подлежат решению:

- 5.1.1. Исследование предметной области, графовых моделей данных, метаграфовой модели данных, предикатного представления метаграфа;
- 5.1.2. Подготовка алгоритма преобразования из метаграфа в плоский граф;
- 5.1.3. Анализ существующих алгоритмов эмбединга и выбор подходящих для экспериментов вариантов;
- 5.1.4. Разработка алгоритма эмбединга метаграфа;
- 5.1.5. Тестирование;
- 5.1.6. Оформление технической документации.

5.2. Требования к функциональным характеристикам

Разрабатываемый алгоритм должен выполнять следующие функции:

- 5.2.1. Производить однозначный перевод метаграфовой модели данных в модель плоского графа за счёт составления матрицы смежности;
- 5.2.2. Позволять проводить цельные эксперименты с различными методами эмбединга, выбранными для экспериментальной части работы, с учётом возможных различных форматов входных данных конкретных алгоритмов;
- 5.2.3. Производить эмбединг полученного графа и визуализировать полученные векторные представления в форме, удобной для ручного анализа результатов.

5.3. Требования к архитектуре программного изделия

Разрабатываемая архитектура системы должна:

- 5.3.1. Обеспечивать требуемый уровень отказоустойчивости для невырожденных входных данных.

5.4. Требования к интерфейсу программного изделия

Интерфейс, поддерживаемый Jupyter notebook.

5.5. Требования к надежности

Система не должна выдавать ошибок, не предусмотренных работой системы; система должна функционировать надежно и устойчиво.

5.6. Требования к языкам программирования

В качестве языков программирования используется Python.

5.7. Требования к составу технических средств

Минимальные системные требования для работы системы:

Процессор с частотой 1 ГГц

1 ГБ оперативной памяти

Видеоадаптер и монитор, способные обеспечить графический режим 1024*768 точек с 32-ти битной цветопередачей

Манипулятор «мышь»

Клавиатура

Установленный браузер (Google Chrome, Yandex.Browser, Firefox или Атом)

Установленный интерпретатор Python с библиотеками, используемыми в программе, а также сам фреймворк Jupyter notebook.

6. Этапы работы

График выполнения отдельных этапов работ приведен в соответствии с приказом об организации учебного процесса в 2021/2022 учебном году.

Таблица 1 - Этапы разработки

№ п/п	Наименование этапа и содержание работ	Сроки исполнения
1	Разработка и утверждение задач проекта	Август-Сентябрь 2021г.
2	Исследование предметной области	Август - Май 2022г.
3	Разработка архитектуры программного обеспечения	Январь — Март 2022 г.
4	Реализация программы	Март — Май 2022 г.
5	Тестирование и отладка	Март — Май 2022 г.
6	Оформление документации	Май — Июнь 2022 г.
7	Защита работы	Июнь 2022 г.

7. Техническая документация

По окончании работы предъявляется следующая техническая документация:

Техническое задание.

Расчётно-пояснительная записка.

Графический материал по проекту в формате презентации.

8. Порядок приема работы

Приём и контроль программного изделия осуществляется в соответствии с подразделом 5.2. данного документа.

9. Дополнительные условия

Данное техническое задание может уточняться в установленном порядке.